Implementing a Culling and Self-Intersection Algorithm for Stereolithography Files in Ada 95

John A. Reisner
Dept. of Comp. Sci. & Eng.
University of Connecticut
Storrs, CT 06268
(860) 486-5582
reisner@engr.uconn.edu

Zeenat Lainwala
Avid Technology, Inc.
1925 Andover St.
Tewksbury, MA 01876
(978) 640-5033
Zeenat_Lainwala@avid.com

Thomas J. Peters
Dept. of Comp. Sci. & Eng.
University of Connecticut
Storrs, CT 06268
(860) 486-5045
tpeters@engr.uconn.edu

Steven Demurjian, Sr.
Dept. of Comp. Sci. & Eng.
University of Connecticut
Storrs, CT 06268
(860) 486-4818
steve@engr.uconn.edu

1. ABSTRACT

Given coordinate information for a triangulated three-dimensional object, and given that we are able to move one of the object's vertices, we wish to determine whether or not the vertex movement has inadvertently introduced a self-intersection within the object. This problem is relevant to the stereolithography domain, where three-dimensional objects may be specified with triangular facets, and rapidly fabricated with a laser and liquid polymer. We developed and implemented an algorithm in Ada 95 that determines whether or not a self-intersection has been introduced to an object, along with a corresponding culling algorithm. The culling process that allow us to resolve the self-intersection question quickly, even in an object with thousands of vertices and facets.

1.1 Keywords

Topology, stereolithography, CAD applications, computer graphics, culling algorithms.

2. INTRODUCTION

Stereolithography is a technology used by CAD designers to rapidly build prototypes of objects. Coordinates of three-dimensional objects are specified in a computer file (a .STL file, specifically), and this file is used as input to a process where the object is fabricated by a laser which cures a liquid polymer. Stereolithography is being used by a wide variety of corporations, including ALCOA, Boeing, Pratt & Whitney, Ford Motor Company, and Spalding.

Stereolithography provides a practical motivation for solving a geometric problem, namely, given a geometric object with vertices, edges, and facets, when we move one of the vertices, determine if the topological form of the object changed. In this paper, when we say that the topological form of an object has been changed, we mean that a self-intersection has been introduced, that is, some part of the object intersects another part of the object, other than on the allowable intersection of a common vertex or edge. In stereolithography, objects are represented with triangular facets, thus, every facet has three vertices. Facets are only allowed to intersect at common vertices, thus, adjoining facets share two vertices, and the common edge between these vertices [4]. If a CAD designer is modifying a stereolithography file (possibly via some graphical interface), then it is desirable to know whether or not the topological form of an object has been inadvertently altered by these modifications [3]. Some examples of vertex movement which introduce and do not introduce self-intersection are shown in Figure 1.

In a stereolithography file, when a single vertex of an object is relocated, all of the facets which share this vertex are altered. For example, in Figure 1, when vertex A is moved, four triangular facets are altered (ABC, ACG, AGH, and AFH). We assume that the object in its initial state has no self-intersections. Thus, once a vertex is moved, we need to determine if any intersection has been formed among the *newly created*, or *modified* facets (such as ABC and ACG in Figure 1c), or between a newly created facet and a *previously existing* facet (such as between ABC and DEF in Figure 1d). Our goal is to *quickly* determine whether or not any such self-intersection has been introduced after a vertex is moved.

In [1], Needham used propagation modeling as part of a solution to this problem. In his design, he exploited a mathematical theorem (described in [7]) which allows a

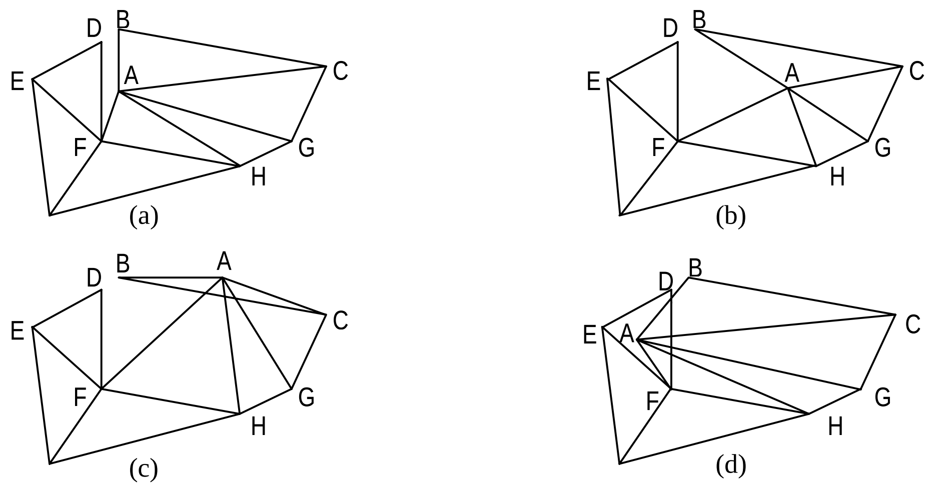

Figure 1. (a) a 3D object with triangular facets. We wish to move vertex A. (b) An example of a move which preserves the object's topological form. (c) A move which does not preserve topological form, because we have moved another facet "through" edge BC (assume ABC and ACG are on the same plane). (d) This move may or may not preserve topological form, depending upon whether or not A "pierces" facet DEF, which can not be determined from this 2D representation, since the vertices are not necessarily co-planar.

certain tolerance to be computed and associated with each vertex. If the vertex is moved in any direction within this tolerance, we can prove that topological form will be preserved. Hence, the propagation is "fired" (or "triggered") when this tolerance is exceeded; then and only then must the object's topology be explicitly checked to see if any new self-intersections have been introduced. In a file with thousands (or millions) of facets, we wish to avoid the processing penalty incurred while performing this check, hence, the mathematical theorem employed is useful, in the sense that it minimizes the cases where the check must be performed.

Although Needham's design does avoid checking for topological preservation unnecessarily, there are still moves that exceed the specified tolerance, yet do not introduce a self-intersection. Because the theorem employed by Needham does not detect every "legal" move, some alternate mechanism must be used to explicitly check for new self-intersections. We have extended Needham's design, therefore, and implemented an algorithm that will perform the necessary computations when required. This paper focuses on this extension of Needham's design.

3. CULLING STEREOLITHOGRAPHY FILES

In a stereolithography file with thousands of vertices and facets, it is not practical to check whether the modified facets remain pairwise disjoint from all preexisting facets in the object (i.e., to use an exhaustive search to determine if a self-intersection exists). Such a brute-force approach is not scalable; as the .STL file grows, the algorithm would not give a CAD designer the necessary information quickly enough. We aimed to develop an algorithm which could be used at interactive speeds [5]. Therefore, our first goal was to derive a subset of the facets in the file that *might* be affected by a given vertex movement, so that the remaining information could be excluded.

Because we don't know which vertex will be moved (nor the magnitude and direction of the movement) until run-time, we must be able to perform the culling process dynamically. In order to cull correctly at run-time, the algorithm must be able to discern which facets are "local" to the modified facets. This is difficult in that the .STL file only gives the coordinates for the facet's three vertices. An intersection might occur along an edge (i.e., between two vertices) or somewhere in the middle of the facet (away from all vertices) For example, in Figure 1d, there is a possible intersection between facets DEF and ACG; however, by looking at vertex information alone, it may not appear that these facets are even close to each other. The key idea is that the *vertices* of intersecting facets are not necessarily close to each other, making it difficult at run-time to determine which facets ought to be eliminated from consideration.

In order to overcome this, we wrote a preprocessing program which analyzes a stereolithography facet file (i.e., a .STL file), and produces various index files. These index files can be accessed at run time so that we can quickly determine which of the previously existing facets are local to facets which are newly created by a vertex movement. The preprocessing program was written in Ada 95.

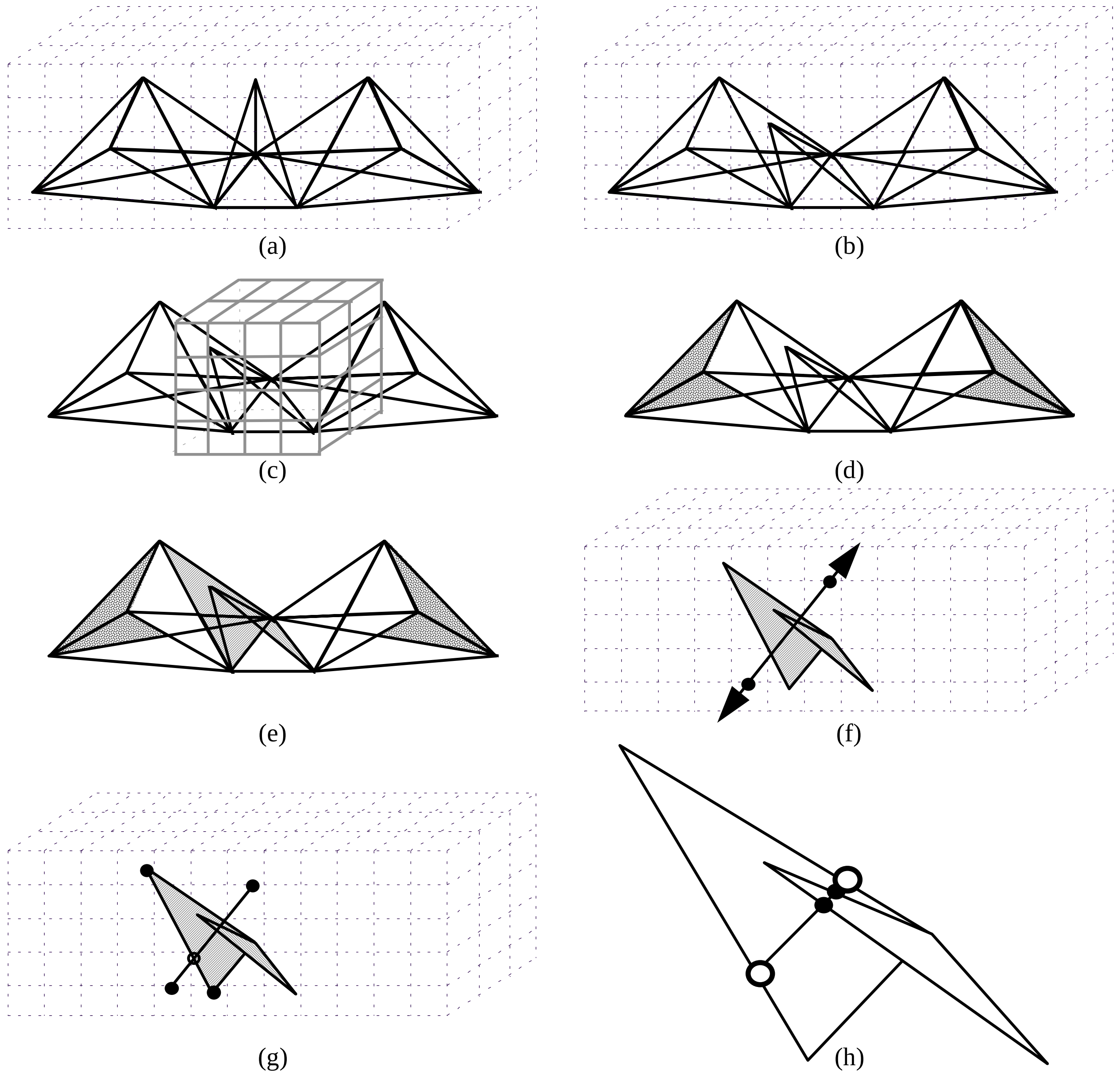

Figure 2. Process for determining self-intersection. Parts (a) - (d) depict the culling process, while parts (e) - (h) illustrate testing for self-intersection. **(a)** A triangulated object partitioned into cubic spaces. **(b)** The same object with a vertex moved. **(c)** The $4 \times 4 \times 2$ set of boxes that completely contain the three facets affected by the move. **(d)** The shaded facets are not in the space defined in part (c), thus, will not be considered anymore. In a small object like this, an abnormally large portion of the file remains for the second part of the algorithm. **(e)** Two facets are selected for consideration; these are the two newly shaded facets. **(f)** The line of intersection for the two planes is found, and bound by the original set of bounding boxes. **(g)** The bound line of intersection is compared with an edge of the old facet. The point of intersection is shown with an open circle. This step is performed for all three edges of both facets. **(h)** The points of intersection from the previous step are used two create two new line segments. If these intersect, then the facets intersect along the length of the intersection; otherwise, the facets do not intersect.

The preprocessing program divides the object's space into equal sized cubes. The size of the cubes can be set when the preprocessing program is run. With predefined cubes filling the object's bounding box, we can efficiently and dynamically cull the input file, if we have readily available answers to the following three questions:

(1) Given the coordinates of any vertex, what is the set of facets use (are connected to) this vertex?

(2) Given the coordinates of any facet, what is the set of cubes completely containing the facet?

(3) Given any cube in object's three dimensional space, what are all the facets that intersect this cube?

The preprocessing program writes a set of index files which allow these three questions to be answered efficiently. At run time, the user selects a facet and moves it. By answering the question (1), we can determine which facets have been modified. By answering question (2) for each of these modified facets, we get a set of boxes which completely contain the area affected by the movement. Then, by using this set of boxes and the answer to question (3), we can create a list of all the previously existing facets which *could* intersect the new (i.e., modified) facets. Culling is performed by checking for intersections solely among these facets and the newly created ones; all other facets are eliminated from consideration. This process is depicted in Figure 2, parts (a) - (d).

4. CHECKING FOR SELF-INTERSECTION

We now focus on this problem: given the coordinates of the vertices for two triangular facets, how can we tell if these facets intersect? This section briefly describes our algorithm; accompanying illustrations are found in Figure 2, parts (e) - (h).

Step 1. For each facet, determine its planar equation using its three vertices.

Step 2. For the pair of facets we are examining, determine the intersection of their containing planes as either the empty set (when the planes are parallel), as the equation of a line (we refer to this line as the *line of intersection*), or as being co-planar (we shall discuss this special case later).

Step 3. Trim the extents of the line of intersection relative to the union of all cubes delimited by the preprocessing program; that is, we define a line segment with endpoints at the line of intersection's "entry" and "exit" points through the bounding box formed by the union of all cubes delimited by the preprocessing program. The purpose of this step is to use a method that, given two line segments, returns the point of intersection if the two segments intersect. Trimming the line of intersection allows us to input a line segment into this routine.

Step 4. Find the points of intersection between the line segment defined in Step 3 and the three line segments defined by the facet edges. This step has four possible outcomes, as shown in Figure 3. If one (or both) of the facets does not intersect this line segment, then the facets do not intersect. If *both* facets intersect the line of intersection, then the facets *may* intersect; we proceed to the next step.

Step 5. The results from Step 4 are used to determine if the facets indeed intersect. Specifically, we check to see if the two line segments (formed by the points of intersection calculated in Step 4) intersect each other. The facets intersect if and only if these line segments intersect.

In the case where the two facets are co-planar, we will be unable to obtain a line of intersection, so the preceding algorithm is not used. A simple test to see if two co-planar facets intersect is to see if any of their edges intersect. This can be easily performed using methods available from the previous algorithm, however, it does not provide a conclusive answer, since one of two intersecting co-planar facets could be completely contained inside the other. For the initial version of the software, we omitted this case from consideration. The percentage of times that a vertex movement would result in two co-planar, intersecting facets, where one was completely contained inside the

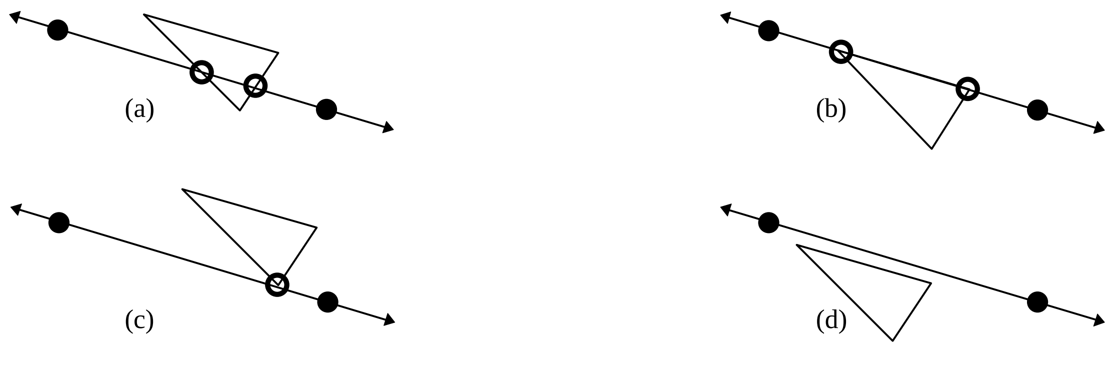

Figure 3. The four possible outcomes of the intersection between a facet and the line segment formed from the line of intersection: (a) Intersection at two points; (b) Intersection along an edge; (c) Intersection at a vertex; (d) No intersection. The line segments used in Step 5 are shown with open circles. In case (c), we use a point instead of a line segment in Step 5.

other, was not considered high enough to justify implementing this special case at this stage of development. Our main emphasis at this time was to determine whether or not we could develop a practical culling algorithm. However, when co-planar facets are detected, this is reported to the user, along with a message that indicates that the current version of the software is unable to conclusively determine whether or not a self-intersection was introduced.

5. RESEARCH MOTIVATION

A propagation is one way to represent a dynamic relationship between classes. Needham proposed propagation modeling as a way to more accurately capture and enforce interobject behavior [2]. The stereolithography domain is one area that a propagation can be utilized to dynamically enforce consistency constraints across objects at runtime. As mentioned in Section 2, the culling and self-intersection detection algorithms were designed to be executed as a result of a triggered propagation.

We are continuing to define and explore improvements to object-oriented design models and methodologies, such that they more accurately model dynamic behavior and interobject consistency requirements. Throughout this effort, we find that detailed, real-world examples (like stereolithography) are valuable when demonstrating the feasibility, utility, and effectiveness of any proposed modeling improvement. This paper is the culmination of an experiment in propagation modeling, as a prelude to proposing extensions to other object-oriented design models and/or methodologies. From these proposed extensions, our eventual goal is to be able to accurately model interobject behavior, and to automatically generate Ada95 code that implements the modeled behavior and specified consistency requirements.

Stereolithography has proved to be a fitting application domain for these efforts. Among the advantages of this domain are: (1) it is complicated enough that it is not trivial; (2) it forces us to confront real-world problems such as the accumulated inaccuracies of floating-point arithmetic; and (3) it has potential application in and benefit to the real-world.

6. EFFECTIVENESS OF THE CULLING

When a vertex is moved, we wish to see what, if any, self-intersections have been introduced. With no culling algorithm available, we would need to compare all of the newly created facets with all of the previously existing facets (i.e., use an exhaustive approach). The time needed to perform these calculations could be expressed as:

$$(N - n) \times n \times t_i$$

where N = the number of facets in the object, n = the number of facets connected to the moved vertex, and t_i = the time it takes to perform the intersection algorithm for two facets. Similarly, let C = the number of facets that are culled (eliminated from consideration) by a culling algorithm, and t_c = the culling time (that is, the time it takes for the algorithm to determine which facets can be eliminated from consideration). The time needed to determine what, if any, self-intersections have been introduced by a vertex movement can be expressed as:

$$t_c + (N - n - C) \times n \times t_i$$

Although both equations are $O(n^2)$ algorithms, there is potential for a large performance gain when both N and C are very large, that is, when we can cull a large percentage of the facets. Algebraically, using the culling algorithm will be the better option when:

$$t_c + (N - n - C) \times n \times t_i \ < \ (N - n) \times n \times t_i$$

or, equivalently:

$$t_c \ < \ C \times n \times t_i$$

For example, consider an object with 100 facets. We move a vertex that is part of six facets (a very typical vertex). Of the 94 remaining facets, let us assume that about one-fourth (22) are eliminated by the culling process. Our formula is now:

$$t_c \ < \ 132 \times t_i$$

However, if the culling algorithm eliminates about three-fourths of the remaining facets (say 70, for example) the formula becomes:

$$t_c \ < \ 440 \times t_i$$

With a similar vertex (i.e., n = 6), which is part of an object with more facets (say 1000), if we eliminate one-fourth of the facets (say 250), then our formula is:

$$t_c \ < \ 1500 \times t_i$$

And if we manage to eliminate 800 facets from consideration:

$$t_c \ < \ 4800 \times t_i$$

Thus, the effectiveness of the algorithm depends on how many facets actually get eliminated from consideration. As more facets are eliminated by the culling process, the potential gains realized by culling increase.

The culling algorithm uses preprocessed index files, which describe the space occupied by the object; this space has been partitioned into cubic boxes. The size of the boxes can be customized at preprocessing time. One question of interest, which influences the algorithm's effectiveness, is what size the boxes should be. If the box is too big, the algorithm will cull a smaller fraction of the facets. Consider, for example, the extreme case, where the entire object fits into one cubic box. Since all of the facets are in the one box, zero facets will be culled. In this case, culling will always take more time than the exhaustive search approach, because $t_c < 0$ is never true. On the other hand, if the cubes are too small, then they will not provide an effective granularity. Too frequently, the set of facets which intersect one box will be the same as the set of facets which intersect the neighboring boxes. One of the steps of the culling algorithm is to obtain the list of all facets intersecting the boxes contained in the area affected by the moved vertex. Too many boxes will increase t_c, and will hurt the effectiveness of the algorithm, even though the information is indexed. For example, consider the ideal case, where the area affected by the move is completely (but just barely) contained in one box. Now imagine that instead, we build cubes with an edge length that is one-tenth of this ideal size. The culling algorithm will now obtain a list of facets from 1000 cubes instead of one, however, the resulting list will be the same. Hence, one way to keep t_c small is to choose an ideal box size at preprocessing time. This is not always possible, since the magnitude of the vertex movement is unknown. Yet a box size can be chosen according to what would be a typical move for the object.

It is theoretically possible to select a vertex at the outer edge of an object, and move it to the opposite side. However, in a design environment where self-intersections are prohibited, such a sweeping move is very unlikely. This truth works to the advantage of the culling algorithm; specifically, since the algorithm was designed for an environment where large, indiscriminate vertex movements are unlikely, while small, exact vertex movements are more typical, a large percentage of facets will usually be culled. Moreover, this information can also be used to obtain a more ideal cube size at preprocessing time. We assert that a good way to set the cube size is to examine the density of the object (in terms of triangular facets) and try to get an idea of how large a move would be possible before it became very unlikely that the move would not nearly invariably cause a self-intersection. This value could then be considered when choosing a box size.

7. EXPERIMENT

We set up an experiment to determine what percentage of facets are likely to be culled, and to determine whether or not we could predict an optimal cube size at preprocessing time. To do this, three .STL files were used, `VALVE.STL`, `SHAFT.STL`, and `BALL-BOX.STL`. These objects were provided by our industrial collaborator Pratt & Whitney. Some information detailing the size and shape of these objects is provided in Table 1.

The `VALVE` object is extremely flat in the y-direction, hence, there are fewer cubes in that direction than in the x- and z-directions. Conversely, the `BALL-BOX` is spherical in shape, thus, the union of the cubes themselves also form a larger cube. In this larger cube, the smaller cubes in the corners are sparse or empty; furthermore, the inside of the object is hollow in some areas. Thus, some cubes are empty or sparsely populated, while others are densely populated. The spherical shape of the `BALL-BOX`, coupled with its larger size, made it the most interesting object to analyze.

The following data excerpt of 12 vertices from the file `BALL-BOX.STL` gives an indication of the close proximity of vertices to each other. The column on the right indicates the number of facets connected to each of the vertices. Thus, exactly six facets share the vertex specified in the second row.

Row	x-coord	y-coord	z-coord	Num Facets
1	5.66173	5.42500	5.90381	5
2	5.66173	5.42500	6.09619	6
3	5.66173	5.52500	5.90381	5
4	5.66173	5.52500	6.09619	5
5	5.66173	6.47500	5.90381	5
6	5.66173	6.47500	6.09619	5
7	5.66173	6.57500	5.90381	6
8	5.66173	6.57500	6.09619	5
9	5.66396	7.68940	5.47749	8
10	5.66396	7.68940	6.52251	7
11	5.67114	4.24106	5.82357	6
12	5.67114	7.75894	5.82357	4

The next step was to preprocess each of the .STL files with varying cube sizes, select certain vertices, and move them various distances from their original positions. The results

Triangulated Object	Number of Vertices / Facets	Dimensions of Object's Bounding Box	Volume of Bounding Box	Density (Facets/Unit Vol)	Vertex Connectivity (Average / Max)
Valve	1356 / 452	$2.22 \times 0.06 \times 1.94$	0.268	1686.58	6.05 / 16
Shaft	2982 / 994	$0.88 \times 0.88 \times 2.94$	2.248	400.81	6.02 / 26
Ball Box	17136 / 5712	$3.52 \times 3.52 \times 3.52$	43.561	131.13	6.04 / 29

Table 1. Triangulated objects (.STL files) used in the experiment, with size data. Vertex connectivity tells the average and maximum number of facets connected to each vertex. For example, in the `SHAFT` object, the "average" vertex is connected to six facets, although there is one vertex that is connected to 26 facets (the latter is termed the *most connected vertex*).

Cube Size	Number of Cubes	Layout of Cubes
0.100	3240	$9 \times 9 \times 40$
0.200	500	$5 \times 5 \times 20$
0.500	81	$3 \times 3 \times 9$
1.000	5	$1 \times 1 \times 5$

Table 2. Varying cube sizes used for the SHAFT object.

Cube Size	Number of Cubes	Layout of Cubes
0.200	6859	$19 \times 19 \times 19$
0.500	729	$9 \times 9 \times 9$
1.000	250	$5 \times 5 \times 5$
1.500	27	$3 \times 3 \times 3$

Table 3. Varying cube sizes used for the BALL-BOX object.

of the culling were then analyzed for each of these moves. We will provide results for the two larger objects, that is, SHAFT and BALL-BOX. The cube sizes tried for these two objects are shown in Tables 2 and 3. Three vertices were chosen for each of the objects. Vertex 1 is a vertex near an outer extremity of the object; it was moved toward the object's center. Vertex 2 is a vertex near the object's center. Vertex 3 is the object's most connected vertex, that is, the vertex with the most facets connected to it. Vertex 3 was intended to give a worst-case scenario, because, in general, moving vertices connected to more facets tends to affect a larger volume in the solid, since more facets and edges are being moved along with the vertex. Each vertex was moved twice--once with a large movement and once with a small movement. For both objects, the smaller movement's magnitude was 0.173 (comprised of a movement of 0.1 in the x-, y-, and z-directions). The larger movement's magnitude was 0.866 for the SHAFT (Δ 0.5 in the x-, y-, and z-directions), and 1.732 for the BALL-BOX (Δ 1.0 in the x-, y-, and z-directions).

When the software was initially designed, the goal was to determine whether or not any self-intersection was introduced as quickly as possible. Hence, our algorithm would search until the first self-intersection was found, and alert the user that a self-intersection had been detected. Therefore, the algorithm would completely test *all* of the facets in the affected area only when a very small movement was made, since large moves almost invariably introduce multiple self-intersections. In order to see how long the testing would take, we changed the software such that it would test all remaining facets after the culling, and then report how many self-intersections were found.

Additional vertex/movement combinations were tried, but are not further discussed in this paper. The reason that movements in all directions are used in this report is that moving a vertex in only one direction tends to "flatten" (and thus minimize) the affected area of the move, favoring the culling algorithm.

8. RESULTS

Tables 4 - 6 show the results of the experiment for the SHAFT object, while Tables 7 - 9 show the results for BALL-BOX. Each of these tables corresponds to one vertex used in the experiment. Graphical depictions of these results are found in Figures 4 and 5.

As expected, more facets must be checked when the magnitude of the movement is greater. Other factors which affect how many facets remain after the culling algorithm completes include where the facet is located, the direction of the movement, and the size and shape of the connected facets.

Magnitude of Movement	Cube Size	Number of Cubes Containing the Newly Created Facets	Number of Previously Existing Facets in Containing Cubes	Percentage of Facets Eliminated
0.866	1.0	4 $(1 \times 1 \times 4)$	842	15.292
0.173	1.0	4 $(1 \times 1 \times 4)$	842	15.292
0.866	0.5	54 $(3 \times 3 \times 6)$	702	29.376
0.173	0.5	24 $(2 \times 2 \times 6)$	396	60.161
0.866	0.2	104 $(2 \times 4 \times 13)$	613	38.330
0.173	0.2	26 $(1 \times 2 \times 13)$	337	66.097
0.866	0.1	1274 $(7 \times 7 \times 26)$	629	36.720
0.173	0.1	312 $(3 \times 4 \times 26)$	342	65.594

Table 4. Results of Vertex 1 (near object's edge) for SHAFT. 7 facets are connected to Vertex 1.

Magnitude of Movement	Cube Size	Number of Cubes Containing the Newly Created Facets	Number of Previously Existing Facets in Containing Cubes	Percentage of Facets Eliminated
0.866	1.0	2 $(1 \times 1 \times 2)$	629	36.720
0.173	1.0	1 $(1 \times 1 \times 1)$	491	50.604
0.866	0.5	30 $(2 \times 3 \times 6)$	653	34.306
0.173	0.5	30 $(2 \times 3 \times 6)$	653	34.306
0.866	0.2	80 $(4 \times 4 \times 5)$	446	55.131
0.173	0.2	18 $(2 \times 3 \times 3)$	192	80.684
0.866	0.1	504 $(8 \times 7 \times 9)$	421	57.646
0.173	0.1	72 $(3 \times 4 \times 6)$	134	86.519

Table 5. Results of Vertex 2 (near object's center) for SHAFT. 6 facets are connected to Vertex 2.

Magnitude of Movement	Cube Size	Number of Cubes Containing the Newly Created Facets	Number of Previously Existing Facets in Containing Cubes	Percentage of Facets Eliminated
0.866	1.0	4 $(1 \times 1 \times 4)$	842	15.292
0.173	1.0	4 $(1 \times 1 \times 4)$	842	15.292
0.866	0.5	18 $(3 \times 2 \times 3)$	431	56.640
0.173	0.5	8 $(2 \times 2 \times 2)$	337	66.097
0.866	0.2	126 $(3 \times 3 \times 14)$	527	46.982
0.173	0.2	78 $(2 \times 3 \times 13)$	398	59.960
0.866	0.1	945 $(5 \times 7 \times 27)$	426	57.143
0.173	0.1	728 $(4 \times 7 \times 26)$	410	58.753

Table 6. Results of Vertex 3 (most connected vertex) for SHAFT. 26 facets are connected to Vertex 3.

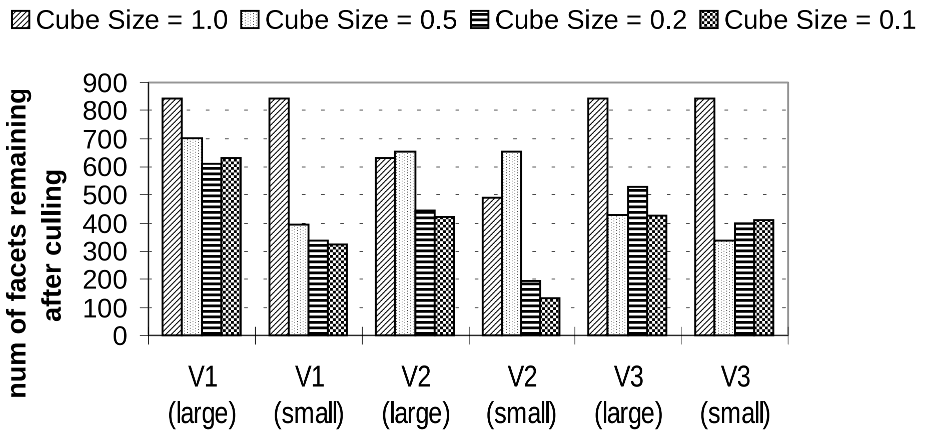

Figure 4. Results for vertex movements in the 994-facet SHAFT object.

In terms of percentage, the culling algorithm was more effective with the BALL-BOX. The SHAFT object contains several elongated triangles which traverse the long z-axis of the object. Large facets explain why, for example, the affected number of cubes remained constant between both the large and small vertex movements, and why there is little change between the number of eliminated facets for the small and large movements of Vertex 3 (see Table 6). Because moving these relatively large facets affect a greater percentage of the object's overall volume, fewer facets are able to be removed from consideration.

Magnitude of Movement	Cube Size	Number of Cubes Containing the Newly Created Facets	Number of Previously Existing Facets in Containing Cubes	Percentage of Facets Eliminated
0.866	1.0	4 $(2 \times 1 \times 2)$	1673	90.237
0.173	1.0	1 $(1 \times 1 \times 1)$	552	96.779
0.866	0.5	12 $(2 \times 3 \times 2)$	1145	93.318
0.173	0.5	6 $(3 \times 2 \times 1)$	330	98.074
0.866	0.2	48 $(4 \times 3 \times 4)$	910	94.690
0.173	0.2	8 $(2 \times 2 \times 2)$	198	98.845
0.866	0.1	294 $(6 \times 7 \times 7)$	839	95.104
0.173	0.1	24 $(2 \times 4 \times 3)$	100	99.416

Table 7. Results of Vertex 1 (near object's edge) for BALL-BOX. 11 facets are connected to Vertex 1.

Magnitude of Movement	Cube Size	Number of Cubes Containing the Newly Created Facets	Number of Previously Existing Facets in Containing Cubes	Percentage of Facets Eliminated
0.866	1.0	4 $(2 \times 2 \times 1)$	1520	91.130
0.173	1.0	1 $(1 \times 1 \times 1)$	448	97.386
0.866	0.5	8 $(2 \times 3 \times 2)$	1251	92.700
0.173	0.5	2 $(3 \times 2 \times 1)$	420	97.549
0.866	0.2	36 $(3 \times 4 \times 3)$	837	95.116
0.173	0.2	4 $(1 \times 2 \times 2)$	164	99.043
0.866	0.1	448 $(6 \times 7 \times 7)$	394	97.701
0.173	0.1	18 $(3 \times 3 \times 2)$	64	99.627

Table 8. Results of Vertex 2 (near object's center) for BALL-BOX. 5 facets are connected to Vertex 2.

Magnitude of Movement	Cube Size	Number of Cubes Containing the Newly Created Facets	Number of Previously Existing Facets in Containing Cubes	Percentage of Facets Eliminated
0.866	1.0	2 $(1 \times 2 \times 1)$	887	94.824
0.173	1.0	1 $(1 \times 1 \times 1)$	448	97.386
0.866	0.5	12 $(3 \times 2 \times 2)$	1869	89.093
0.173	0.5	6 $(3 \times 2 \times 1)$	938	94.526
0.866	0.2	45 $(3 \times 5 \times 3)$	883	94.847
0.173	0.2	27 $(3 \times 3 \times 3)$	456	97.339
0.866	0.1	420 $(7 \times 10 \times 6)$	614	96.417
0.173	0.1	168 $(7 \times 6 \times 4)$	426	97.514

Table 9. Results of Vertex 3 (most connected vertex) for BALL-BOX. 29 facets are connected to Vertex 3.

Also as expected, the cube size affects the effectiveness of the culling process. In general, as the cube size is reduced, there are fewer facets remaining after the culling process. However, this is not universally true. One of the more interesting findings is in Table 9, where a reduction in cube edge size from 1.5 to 1.0 more than doubled the

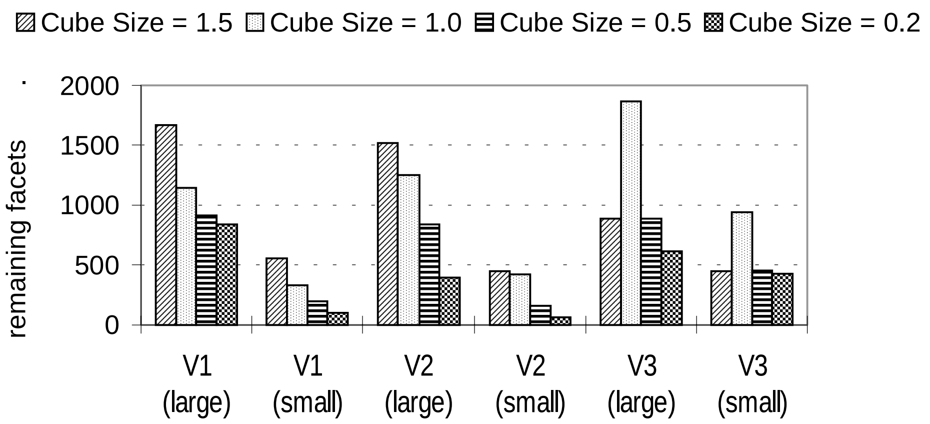

Figure 5. Results for vertex movements in the 5712-facet BALL-BOX object.

remaining facets after culling. This is because of where the cube boundaries are after the preprocessing is finished. The movement of this particular vertex was more neatly contained in the larger cubes. Hence, with the smaller cubes, many facets which were located just beyond the space affected by the movement also happened to intersect the boxes which contain that space, and thus could not be eliminated by the culling process.

9. CONCLUSION

We have designed and implemented in Ada95 an algorithm which determines if a vertex movement in a triangulated object has changed the topology of the object, that is, has introduced a self-intersection. The algorithm requires some preprocessing, where the object space is partitioned into cubes. Index files are created which allow the software to quickly establish that certain facets are so far away from the space affected by the movement, that these facets need not be checked for self-intersections. This process could be used to verify that design changes modifying a stereolithography file do not inadvertently altered the object's topology. Because of the sorting and indexing that occurs at preprocessing time, the algorithm is fast enough to be used interactively. Knowing the magnitude of typical editing session vertex movements would be valuable when selecting the granularity to use when preprocessing an object.

The overall effectiveness of the algorithm fluctuates depending upon factors such as how many facets are connected to the vertex being moved, the shape and area of these facets, the magnitude and direction of the movement, and how neatly the resulting affected volume fits into the predefined borders of the preprocessed cubes. Nevertheless, the algorithm works well, especially in cases where a large, fairly uniform object with relatively small facets (such as the BALL-BOX described in this paper) is being modified with small vertex movements. In such cases, the algorithm consistently eliminated over 90 percent of the object's facets.

Further research could be done on how to more efficiently partition the triangulated object, such as larger boxes in more sparsely triangulated sections of the object. Furthermore, once an update is made, the index files would either need to be updated, or else a log maintained of all of an object's edits, in the case where more than one vertex would be moved. Implementing such an algorithm, which would allow moving multiple vertices, was not attempted at this time.

10. REFERENCES

[1] D. Needham, S. Demurjian, and T. Peters. An Ada95 Basis for Propagation Modeling. Proc. of the 1997 TriAda Conf., St. Louis, MO, 1997, pp. 263-272.

[2] D. Needham, "Object Oriented Propagation Modeling to Support CAD/CAM and Software Engineering" PhD Dissertation, The University of Connecticut, 1997.

[3] T. Peters, S. Demurjian, D. Needham, R. Peters, and S. Dorney, "Propagating Topological Tolerances of Rapid Prototyping", Proc. of the Int'l Mechanical Eng. Conf. and Exposition, MED-Vol. 4, Atlanta, Nov 1996, pp. 487-498.

[4] P.F. Jacobs, Stereolithography and Other RP&M Technologies, Society of Manufacturing Engineers, 1996.

[5] I. Fudos and C.M. Hoffman. "Systems of Geometric Constraints", ACM Trans. on Graphics, Vol. 16, No. 2, Apr 1997, pp. 179-216.

[6] D. Needham , S. Demurjian , K. El Guemhioui , et al., "ADAM: a Language-Independent, Object-Oriented Design Environment for Modeling Inheritance and Relationship Variants in Ada 95, C++, and Eiffel", Proc. of the 1996 TriAda Conf., Page 99-113.

[7] Anderson, S.M. Dorney, T.J. Peters, and N.F. Stewart, "Polyhedral Perturbations that Preserve Topological Form," CAGD, Vol. 12, 1995, pp. 785-799.