CCDC DAC-TR-2020-021
JASPO-V-19-03-001
March 2020

Methodology Improvements to Non-Uniform Rational B-Splines (NURBS) Geometry Conversion in BRL-CAD

by Clifford Yapp

DISCLAIMER

The findings in this report are not to be construed as an official Department of the Army position unless so specified by other official documentation.

WARNING

Information and data contained in this document are based on the input available at the time of preparation.

TRADE NAMES

The use of trade names in this report does not constitute an official endorsement or approval of the use of such commercial hardware or software. The report may not be cited for purposes of advertisement.

CCDC DAC-TR-2020-021
JASPO-V-19-03-001
March 2020

Methodology Improvements to Non-Uniform Rational B-Splines (NURBS) Geometry Conversion in BRL-CAD

by Clifford Yapp
CCDC Data & Analysis Center

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED *(From - To)*
March 2020	Technical Report	01/2018 – 01/2019

4. TITLE AND SUBTITLE
Methodology Improvements to Non-Uniform Rational B-Splines (NURBS) Geometry Conversion in BRL-CAD

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Clifford Yapp

5d. PROJECT NUMBER
JASPO-V-19-03-001

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Director
U.S. Army CCDC Data & Analysis Center
6896 Mauchly Street
Aberdeen Proving Ground, MD 21005-5071

8. PERFORMING ORGANIZATION REPORT NUMBER
CCDC DAC-TR-2020-021

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Joint Aircraft Survivability Program Office
701 Courthouse Road
Suite 1G140, Bldg 15
Arlington, VA 22204-2489

10. SPONSOR/MONITOR'S ACRONYM(S)
JASPO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
JASPO-V-19-03-001

12. DISTRIBUTION / AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
BRL-CAD is an open-source computer-aided design and analysis software package used as the foundation for vulnerability/lethality (V/L) analysis software suites in the U.S. Department of Defense. Conversion of pre-existing commercial vehicle information into formats analyzable by BRL-CAD often introduces geometric errors, most commonly in the form of overlaps between components introduced by meshing errors. Two improvements to BRL-CAD's conversion methodology were implemented to automatically assist users in avoiding these problems. The goal is to produce analysis targets that satisfy a set of quality metrics relevant to V/L analysis in less time and with improved quality.

15. SUBJECT TERMS
Non-Uniform Rational B-Splines, NURBS, mesh, triangulation, solidity, plate mode, conversion, geometry, Joint Aircraft Survivability Program, JASP

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Clifford Yapp
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT	72	19b. TELEPHONE NUMBER *(include area code)* 410-278-1382

Table of Contents

List of Figures

List of Figures

List of Figures

List of Figures

Acknowledgements

The authors wish to acknowledge the contributions of the following individuals for their assistance in the creation of this report:

Christopher S. Morrison, Dalcom Engineering and Technical Services

Shannon K. Abel, Dalcom Engineering and Technical Services

William K. Bowman, U.S. Army Combat Capabilities Development Command Data & Analysis Center

Executive Summary

The U.S. Army, Navy, and Air Force all make extensive use of geometric models for computer analysis of the vulnerability/lethality characteristics of vehicles. Although modern production methods generally result in the availability of pre-existing data from commercial computer-aided design systems, such data must be translated from commercial formats to the formats used by U.S. Department of Defense analysis tools. In the course of such translation, geometric errors are typically introduced when approximating original commercial data with triangle meshes.

This report documents efforts to improve conversion methodologies to address a specific, common source of error: the creation of overlaps (regions of space which two or more objects both claim to occupy) introduced by replacing smooth, continuous surfaces in the original commercial data with planar approximations (typically in the form of triangles).

The implementation of two improved conversion methods is documented, including details on the approaches used, the problems encountered, and the results achieved to date.

The first methodology attempts to improve the process of triangle mesh generation to either reduce or eliminate completely the introduction of spurious overlaps. Success was achieved, but robustness and performance improvements will be needed to scale-up this methodology to large-target geometries.

The second methodology improvement implements a methodology allowing thin surfaces such as aircraft skins, sheet-metal vehicle exteriors, and component enclosures to be modeled by nonsolid shapes. Historically this methodology was supported only for triangle meshes. Such surfaces can now be represented in BRL-CAD using directly imported commercial information, without requiring translation to a triangle mesh.

1. INTRODUCTION

Vulnerability/lethality (V/L) analysis strives to predict how vehicles and other military targets will behave when exposed to a variety of threats. A key input to such an analysis is a geometric description of the shape of the vehicle that is to be subject to an attack. In recent decades the broad adoption of commercial computer-aided design (CAD) software tools has resulted in the availability of a large body of pre-existing vehicle models produced during the design and manufacturing process.

In theory, the availability of these pre-existing data speeds up the V/L analysis cycle tremendously, but in practice it is still necessary to perform a great deal of manual preparatory work before a geometric description is ready for V/L analysis. Some of this work is inherent in the nature of the differing needs of V/L and commercial production (many commercial models will have detail not relevant to an analysis, for example), but a significant additional cleanup burden is introduced by the geometry conversion process when smooth, nonplanar surfaces are approximated by triangles. This approximation can introduce interference errors in the form of multiple independent objects producing incompatible triangle sets that both claim the same volumetric space. These errors (referred to as *overlaps*) can, in turn, propagate up through an analysis by complicating the reporting of ray/shape intersections that form the geometric bedrock of those analyses. In such cases an interrogating ray will not know what "correct" volume to report for that particular volume since there is no unambiguously correct answer (Figure 1). Resolving these problems using constructive solid geometry (CSG) subtractions (i.e., using BRL-CAD's modeling support to define a subtraction volume using one of the solids that removes overlapping material from the volume of the other solid) is a manual, time-intensive process that can have significant negative implications for analysis turnaround times, volumetric fidelity, and for raytracing performance during analysis.

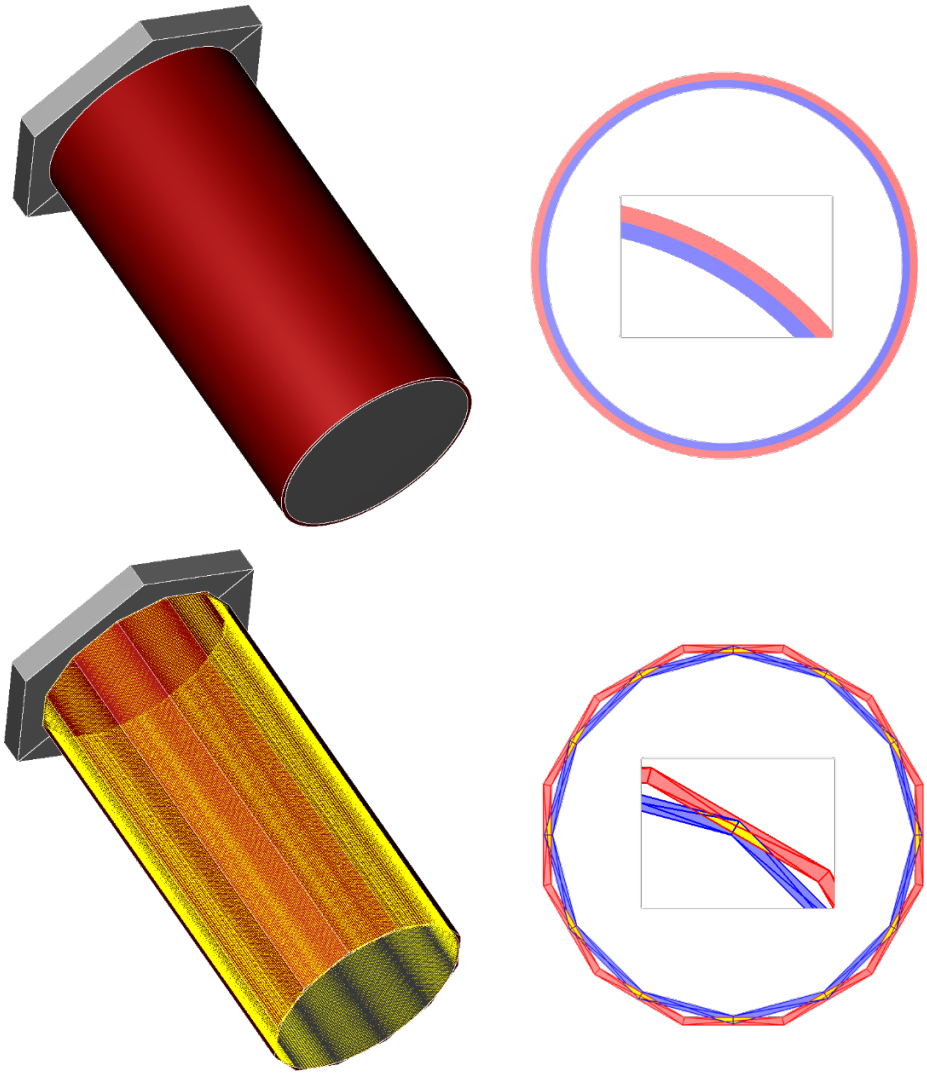

Figure 1. Example overlap introduced by triangulation of smooth solids (overlaps are yellow). The images on the right represent cross sections of the cylinders showing details of the relationships between the original and interfering cylinder faces.

Most commercial CAD geometry describing targets is described using a geometry representation know as Non-Uniform Rational B-Splines (NURBS) Boundary Representations (BReps) (Robert McNeel and Associates, 2018). NURBS are general mathematical surfaces capable of representing a wide variety of shapes, but they are also quite complex. Although a detailed description of the mathematics of NURBS BReps is beyond the scope of this report, for discussion purposes the reader must be familiar with the basics of how they define closed volumes in space. Typically a group of individual NURBS surface patches (referred to as *faces*) are combined together to form a closed volume (Figure 2).

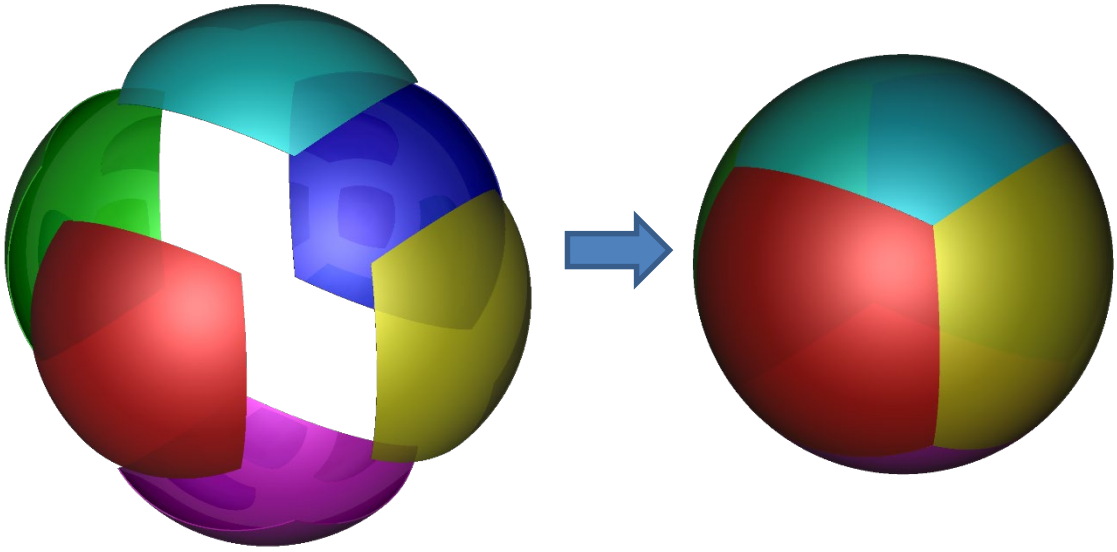

Figure 2. Individual NURBS surfaces combining to form a closed boundary representation

The 3-D curves at which those faces join are referred to as *edge curves* (Figure 3.)

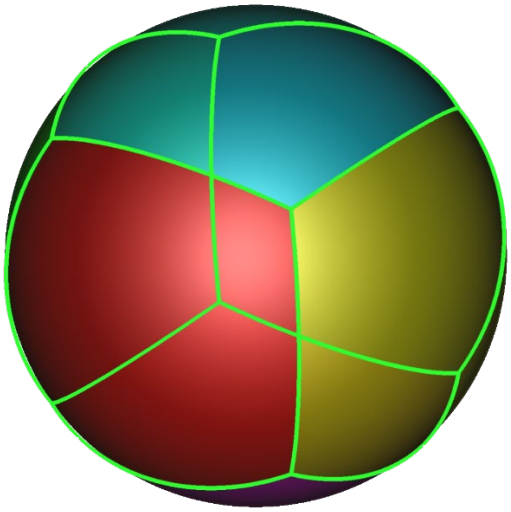

Figure 3. Edge curves (green lines) defining the meeting of BRep faces

Edge curves are typically stored independent of surfaces as NURBS curves. Because those curves are independent of the surfaces, they can be used to define "correct" 3-D edge point face edges even when the two joined surfaces do not align perfectly. These 3-D curves, in turn, correspond to 2-D trimming curves defined in the parametric surface domain of individual NURBS BRep faces, which are used to define the portion of the NURBS surface that contributes to that particular face area.

Imperfect face alignment is frequently encountered in real-world NURBS-based BRep models (in fact, alignment imperfections are both numerically and mathematically inevitable in the

general NURBS surface intersection case) (Sederberg, Anderson, & Goldman, 1984), which means processing techniques (both meshing and raytracing) have to be prepared to accommodate such imperfections. Floating-point-based computational imprecision and modeling tolerances also limit the degree of perfection obtainable in practice.

Historically, none of the geometry systems used by U.S. Department of Defense analysis tools were capable of interrogating NURBS directly with solid ray tracing. BRL-CAD has recently introduced the ability to perform such interrogations, avoiding the overlap triangles by simply not introducing any triangles at all. However, other V/L analysis technology stacks such as the FASTGEN/COVART analysis suite (Defense Systems Information Analysis Center, 2020) still rely on triangle conversion.

Even when NURBS geometry can be used directly, there are modeling techniques for thin surfaces (such as sheet metal parts on vehicles) that use only a single surface to represent the shape. Historically, FASTGEN has supported such shapes by allowing triangle meshes approximating such surfaces to define an implicit thickness. Such solids are referred to as "plate mode" solids. Thus far that interrogation mode has only been supported for triangle meshes. If a modeler wishes to convert thin surface parts for V/L purposes, the two options that have been available historically are 1) define a closed solid and 2) convert the NURBS surface to a plate-mode triangle mesh. Defining a closed solid for such a thin shape is possible but problematic (it is not hard for the surfaces to self-intersect and create holes), and converting the surface to a mesh to use plate mode again runs the risk of introducing spurious overlaps.

Ideally, BRL-CAD should be able to both take advantage of keeping NURBS shapes and interrogating them directly as well as to automatically convert NURBS solids to triangle meshes when such an operation is desirable *without* requiring manual effort.

In 2019, BRL-CAD developers undertook the implementation of an automatic triangle-meshing algorithm to improve BRL-CAD's NURBS to triangle conversion (Figure 4), plus an expansion of the plate-mode raytracing methodology to directly support NURBS-based shapes. These capabilities minimize conversion times while also improving the quality of imported target data.

For development purposes, small test cases with representative interference or surface behaviors were used to allow for rapid iterative development and clarity of visualization. For solid meshing, the National Institute of Standards and Technology (NIST) Model-Based EnterpriseProduct and Manufacturing Information (PMI) test models (NIST, 2014) were used, and for overlap resolution, specially designed inputs were generated by U.S. Army Combat Capabilities Development Command Data & Analysis Center target modeling team to represent typical interference generating cases.

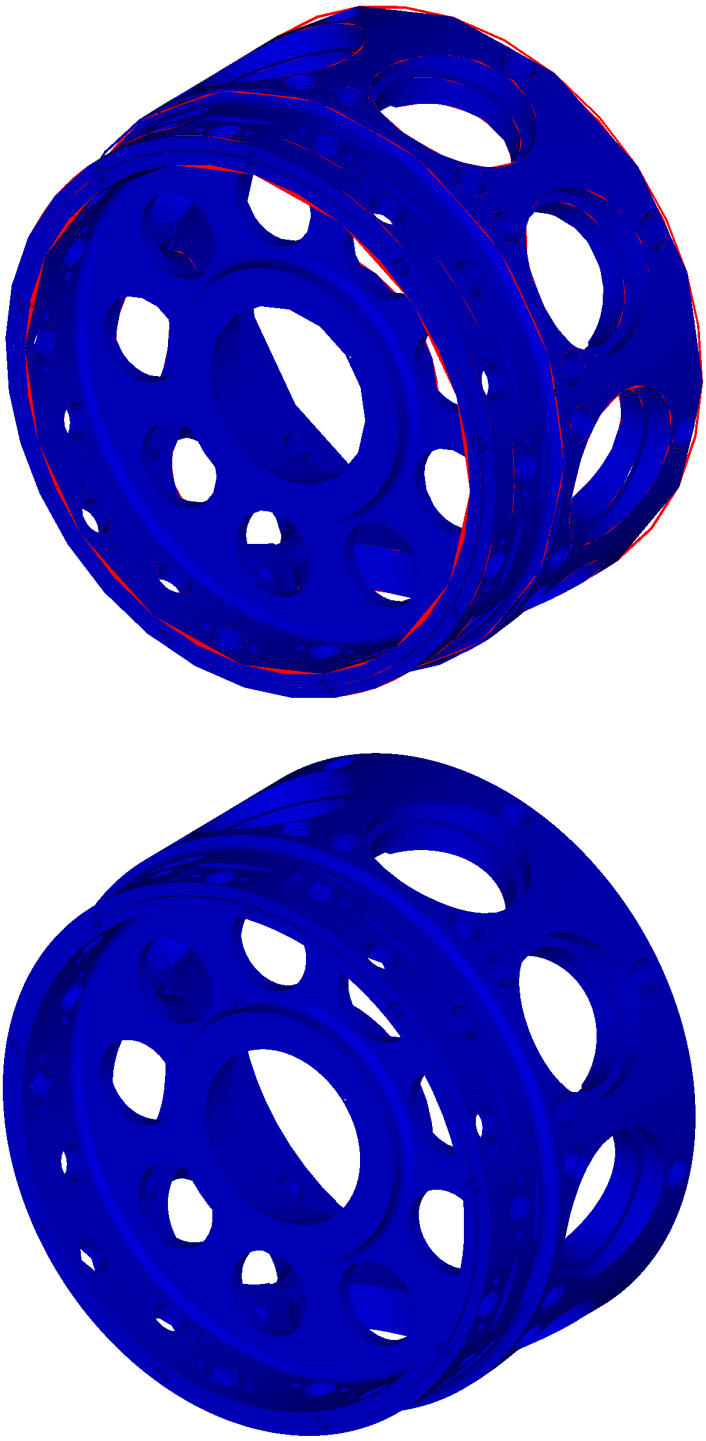

Figure 4. Example part (Elg, 2020) showing improvements in NURBS mesh generation. Top image is using original BRL-CAD logic to generate meshes; red areas highlight solidity problems. Bottom image renders the same part using the new watertight meshing logic, which eliminates the flaws.

2. METHODS, ASSUMPTIONS, AND PROCEDURES – MESHING

When attempting to generate triangle mesh output from BRL-CAD NURBS objects, it is important to define specific quality criteria that must be satisfied for analysis purposes. Avoiding overlaps is the goal, but it is also important that the mesh be *solid*: it should define a closed volume in space without "cracks" between triangles that would allow interrogating rays to slip through and report a miss. For refinement purposes, a mesh should be *valid*: it should not self-intersect (even when such a self-intersection has negligible impact on the overall mesh volume) or exhibit topological problems such as holes, unmated edges, or flipped faces. Beyond those necessities, it is desirable that a mesh have no more triangles than necessary to satisfy the user's demands of model accuracy and (where possible) avoid long thin triangles.

2.1 Existing Capabilities

The starting point for NURBS surface meshing is the sampling of a set of points along trim curves and in the surface interiors in the 2-D parametric space of each BRep face, and then integrate a Constrained Delaney Triangulation (CDT) algorithm to connect those points with a series of triangles. Those 2-D triangles are then combined with the 3-D evaluations of the 2-D parametric vertex points to define the corresponding triangles in 3-D.

BRL-CAD uses the openNURBS (McNeel, Inc., 2020) libraries for NURBS data structures and a variety of support routines used during implementation, but openNURBS itself only includes a very basic meshing capability that is not suitable for production applications. However, there has also been prior work in BRL-CAD on this topic. BRL-CAD developers implemented a meshing algorithm in BRL-CAD in 2014 using the Poly2Tri (Green, 2020) implementation of Domiter and Zalik's CDT algorithm (2008) that successfully represents BRep faces with meshes.

This work enables BRL-CAD to produce interactive shaded displays of NURBS solids and constitutes an essential piece of the NURBS BRep to triangle mesh conversion puzzle, but is not sufficient by itself to produce analysis-ready meshes. Because it treated each surface as an independent meshing problem, there was no guarantee that individual surface meshes would join up at the boundaries. Consequently, the generated triangles were suitable *only* for visualization, and even there the lack of watertight edges sometimes introduced visual artifacts that could be disconcerting to users. Without detailed knowledge of the implementation, such artifacts tend to be interpreted as indicating solidity errors in the underlying geometry. An example of this problem can be seen in Figure 5.

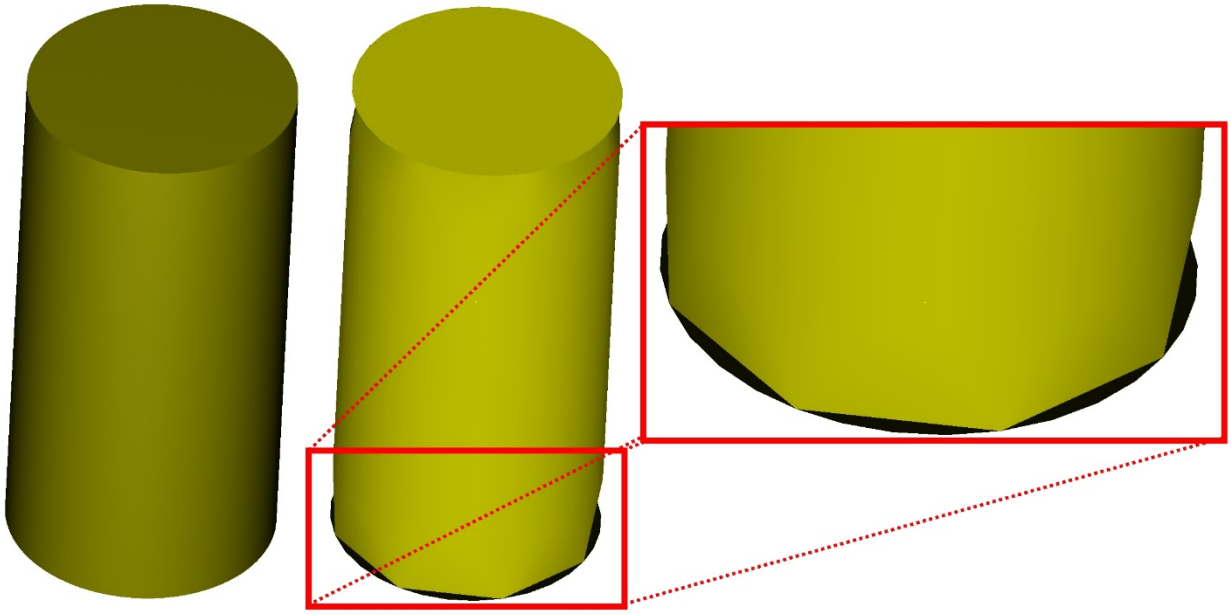

Figure 5. Triangulation of a NURBS cylinder using pre-existing BRL-CAD NURBS meshing capabilities. The left image is the original shape; the middle image is a rendering of the full mesh; and the right image highlights an area where separate BRep face meshes do not align.

Toward the end of the 2014 effort, the possibility of generating watertight meshes to correct this problem using the edge curves as common sources of points with which to terminate each surface mesh was explored. Early feasibility studies were promising, but work priorities shifted before the capability could be fully realized.

Meanwhile, subsequent work produced a "bot check" command in BRL-CAD capable of verifying the topological validity of individual Bag of Triangles (bot) primitive shapes via triangle connectivity data. This command is important because it provides an independent validity check that can be applied to any outputs generated from NURBS meshing.

Because overlap resolution of meshes requires an ability to determine what is inside and outside the mesh, the obvious first step for new work was to complete the unfinished support for generating watertight meshes started in 2014. This would allow for a well-defined inside/outside determination in subsequent processing. Once that known limitation was addressed and the generated bot meshes could pass the "bot check" validity check, BRL-CAD would be able to generate output meshes that would be a suitable platform from which to identify overlapping triangles and devise strategies to resolve them.

2.2 Creating Watertight Meshes – Sampling BRep Face-Edge Curves

The basic strategy used in this work to implement watertight meshing was the one explored by Bowman (2014): sample common points on the edge curves, identify the corresponding 2-D points in each NURBS surface, and use those points as the boundary polygon points when

performing the individual face CDTs. Most of the original exploratory work on watertight meshing had not been integrated into the BRL-CAD codebase, so it proved necessary to execute a new implementation of this methodology in BRL-CAD's libbrep library. The stages of this process are the following:

1. Apply the openNURBS (McNeel, Inc., 2020) ShrinkSurfaces routine to the BRep faces to tighten surface 2-D extents to closely align with the maximal extents of the face trimming curves. Unshrunk surfaces were sometimes found to generate problematic meshing solutions where sampling and triangulation in the parametric domain aligned poorly with the 3-D evaluation that defines the final mesh (Figure 6). Shrinking the surface size also better aligns the surface parametric domain extents to the active face size (Figure 7).

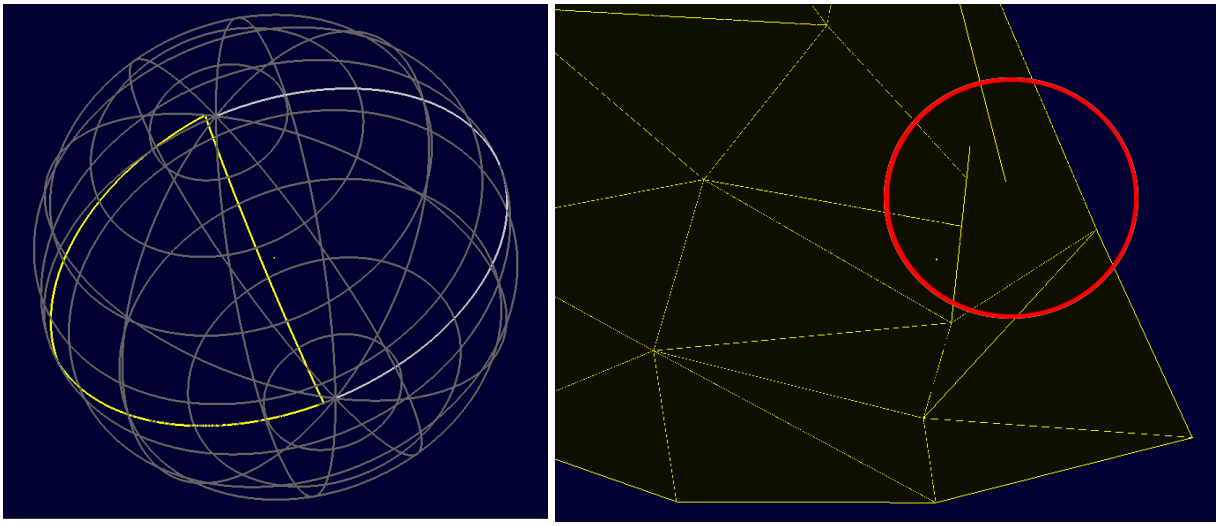

Figure 6. The initial (pre-shrunk) state of a NURBS face. The yellow line in the left image denotes the subset of the spherical surface active in the face. The right image shows a subset of the triangles produced, with the problematic area circled in red (NIST2 Face 237) (NIST, 2014).

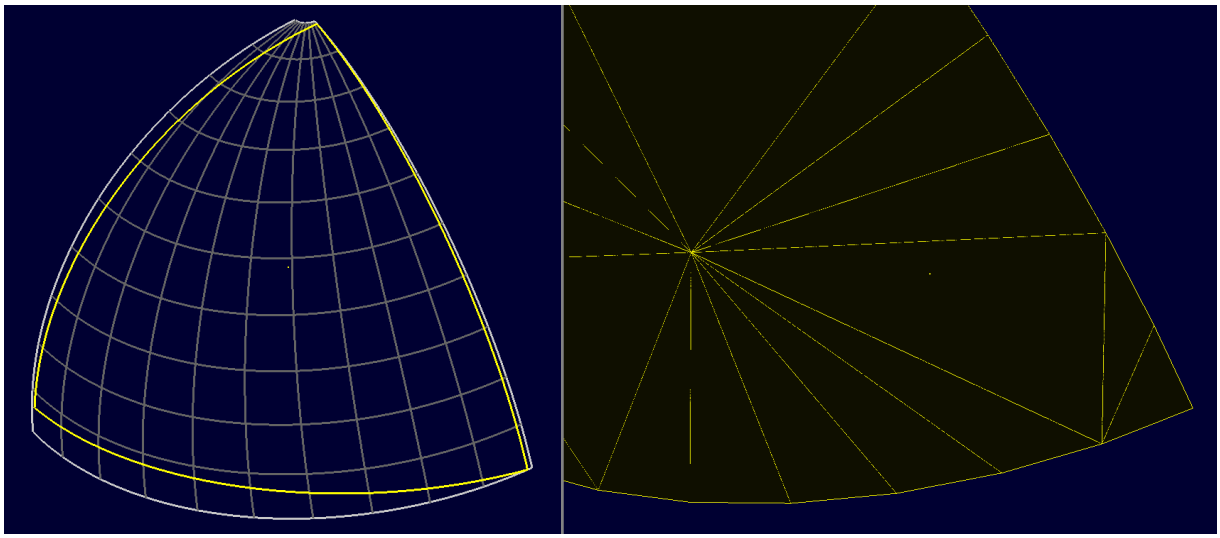

Figure 7. The same face after application of the ShrinkSurfaces routine. Note the triangle mesh no longer exhibits the problematic topology (NIST2 Face 237) (NIST, 2014).

2. Identify suitable normal vectors for vertex points. Naïve evaluation of normal vectors can produce incorrect results when working near singularities—NURBS regions where a line in the surface's parametric domain maps to a single point in 3-D space. At such a point, the surface normal is not well defined (Figure 8). It is necessary to define a locally reasonable triangle normal to avoid visual artifacts when rendering the generated mesh.

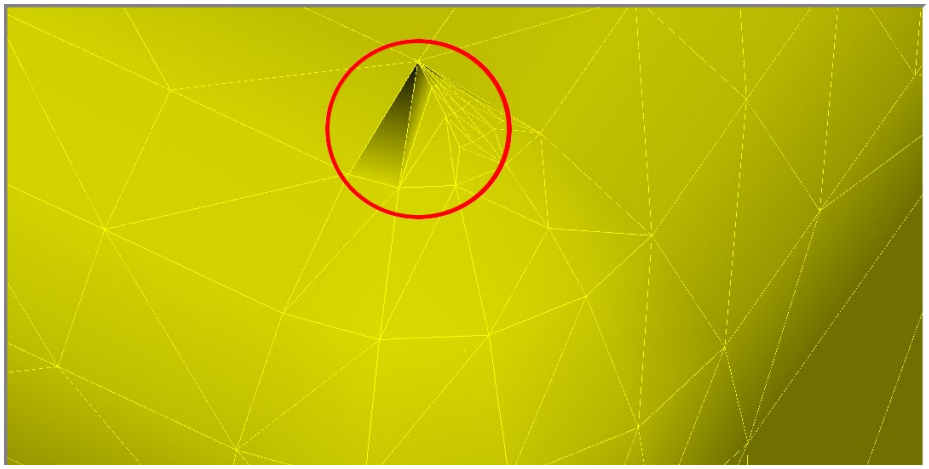

Figure 8. NIST2 exhibiting an incorrect triangle normal at a singular vertex (NIST, 2014)

3. Split edge curves into polyline segments based on tolerance and interference criteria (the latter will be covered in more detail in a later section.) While splitting, it is necessary to identify and track the corresponding trim curve points that most closely evaluate to each 3-D splitting point used to refine the edge curve approximation. It was found that simply aligning the points using the normalized curve parameters was not sufficient to produce

meshes without severe distortions near edge curves (Figure 9). Correcting this necessitated the implementation of a searching routine to locate the nearest 2-D edge curve points.

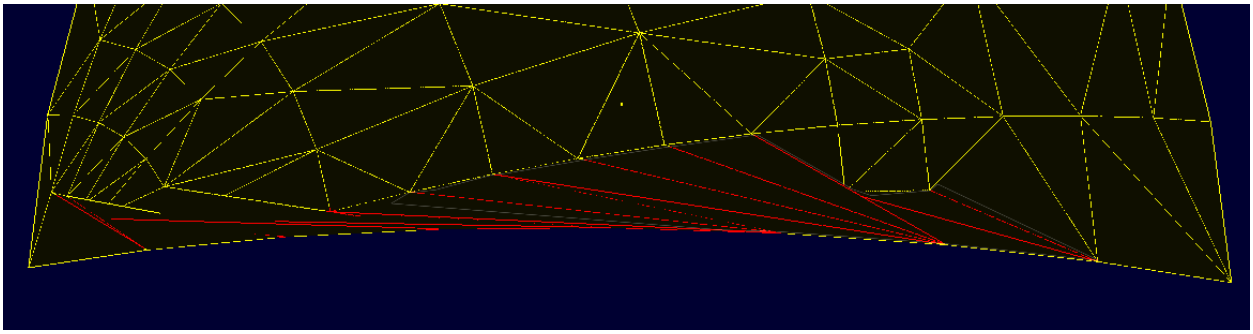

Figure 9. This triangulation an example of the distortion that can be introduced without performing the work to identify the closest 2-D point associated with a 3-D edge point. Distorted edges are highlighted in red (NIST2 Face 246) (NIST, 2014).

Once the splitting of the edge curves is complete, with corresponding 2-D points identified in the various faces, the necessary information is in place to define outer bounding polygons for the CDT algorithm in such a way that those boundaries are forced to line up in 3-D.

The original logic for generating edge polylines in BRL-CAD walked the edge curves rather than splitting them. However, the need to coordinate 3-D and 2-D curve splitting for watertight meshing ended up requiring detailed understanding of the splitting behavior of the curve routines at each point of the process. It proved advantageous for debugging purposes to adopt a splitting approach that kept all curve relationships valid at all steps of the process. Unfortunately, this change introduced a complication when it came to splitting edges that caused quite a bit of difficulty during implementation; edge curves, whether in 2-D or 3-D, have a directionality associated with them (i.e., each curve has a start and an end point). For the 2-D curves, this directionality is used to define inside or outside status for the portions of the NURBS surface they enclose. However, that means that 2-D curve directions may not agree with the direction of the 3-D edge curve they are associated with. Thus, when splitting 3-D curve segments, it is necessary to check on the corresponding 2-D curves whether their directions are reversed relative to the 3-D curve being split and identify the correct splitting points accordingly.

2.3 Creating Valid Meshes – Sampling BRep NURBS Surfaces

While watertight meshing was addressed via processing the edge curves and defining polygons, application of the "bot check" command revealed that there were still topology flaws in the meshes being produced. Simply resolving the edge gap problems of the existing meshing logic was not enough to reliably produce valid output meshes. This was an unanticipated obstacle. Real-world examples were demonstrating that a Poly2Tri CDT mesh in the 2-D parametric space

of the NURBS surface did not always manifest as a topologically valid mesh when mapped into 3-D space.

After some initial experimentation and alterations to the meshing code, it became clear that it would be necessary to incorporate validity testing directly into the meshing workflow to determine which logic was responsible for introducing errors. After completing this work, attention focused on the surface point sampling code.

The original 2014 surface sampling implementation used a quad tree overlaid on the surface's parametric plane to identify sampling points. Those boxes were subdivided recursively until they reached a small enough size according to current surface tolerance settings. Points from these boxes that proved to be directly on a 2-D edge curve need to be rejected for further processing. To find such points, an RTree (Guttman, 1984) spatial acceleration is used (Figure 10) to quickly find any 2-D curve segments close to the sampled point. Those segments are then checked using a point-on-line-segment test to determine if the point in question is in fact directly on a segment.

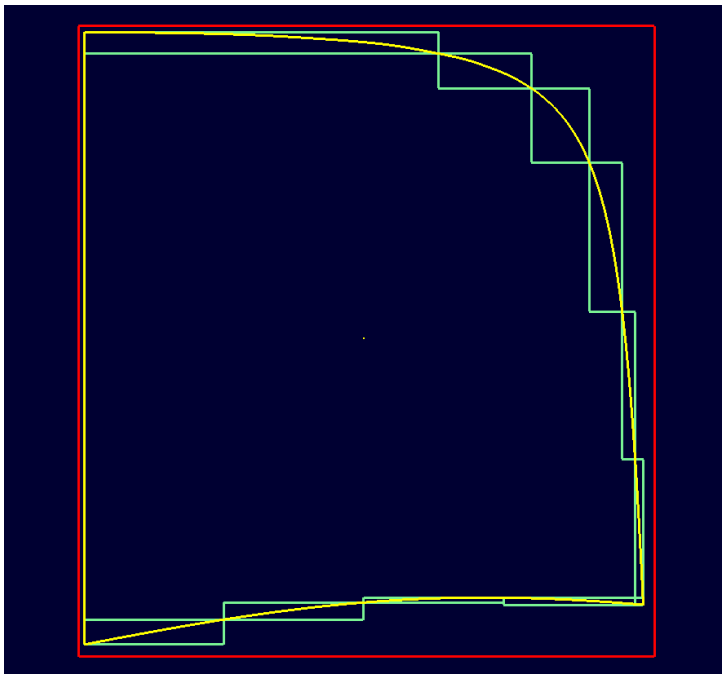

Figure 10. RTree leaves (green) drawn with the 2-D trimming curves (yellow) and surface bounds (red). This structure defines local regions in the 2-D parametric space of the surface used to guide point sampling (NIST2 Face 246) (NIST, 2014).

This check avoided points directly on the boundary polygon during CDT (avoiding such points is a necessary precondition for the algorithm to succeed), but a number of additional specific problems were identified as more-sophisticated analysis and debugging routines were incorporated into the processing logic, as follows:

1. In the neighborhood of singularities, the surface was being heavily oversampled in 2-D, resulting in highly distorted 3-D meshes near those points (Figure 11).

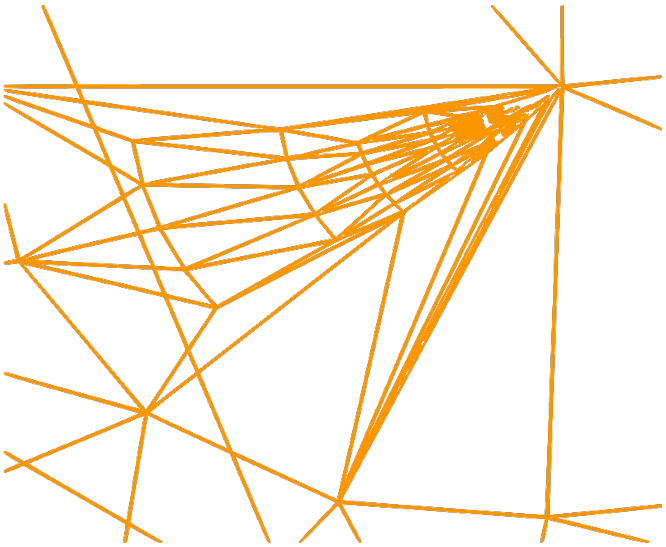

Figure 11. Example of oversampling in the vicinity of a singular NURBS surface point

2. The use of a rectangular grid for surface point sampling was causing occasional problems for the CDT algorithm (Figure 12). (Collinear points are not considered valid CDT inputs.)

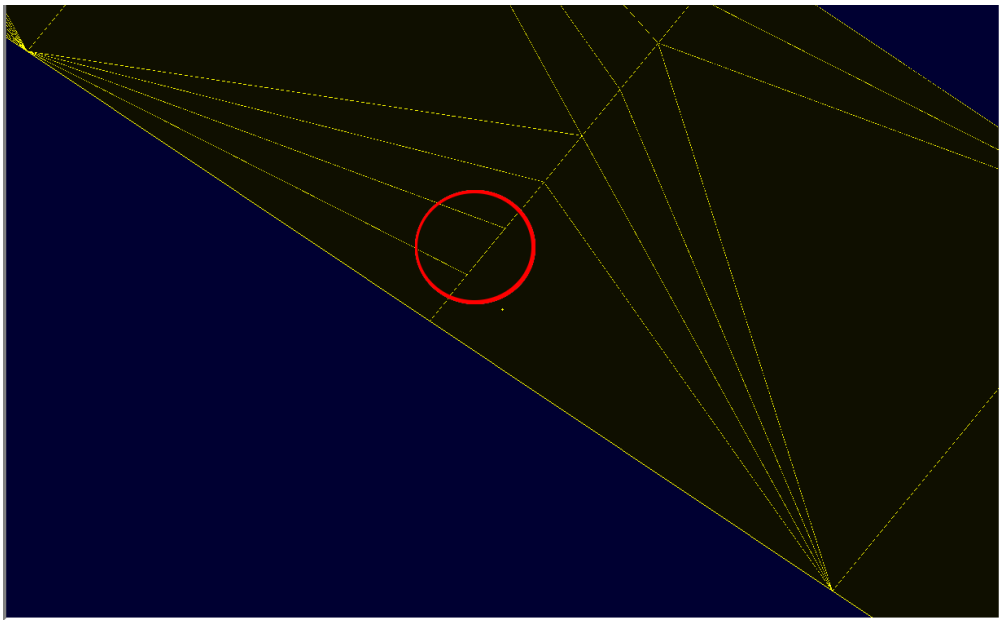

Figure 12. Example of incorrect triangulation due to linearity of sampled points

3. Differences between surface and edge-curve sampling tolerances sometimes resulted in very large triangles next to very small triangles, which often produced flipped triangles when comparing BRep surface normals with triangle normals.
4. Valid triangulations in 2-D could still produce flipped normal triangles in 3-D, occasionally even in surface areas that appeared otherwise unremarkable (to visual inspection, those surface regions were not near singularities or other known problem areas).

The first three difficulties were addressed with a change to how surface points internal to the mesh were sampled. A new strategy was introduced that tested all quad-tree boxes at each stage of the refinement process for their 3-D length and width, and did not split them in any direction where their 3-D length was less than thresholds established based on tolerances.

Once those boxes are sufficiently split according to triangulation tolerances and surface dimensions, they are then compared with the RTree holding the trimming-curve segments to see if any particular box is close to the edge of the BRep face. If the box is close, the refinement process for the surface box continues but uses a new set of termination criteria driven by the closest edge segment. When the surface box is split in this mode, each of the new boxes is also checked to see which ones are close to the edges. For any that are no longer overlapping the RTree of the edge segments, their refinement ceases. For boxes that are still close to the edge, if the surface bounding box dimensions are comparable to the edge segment length, the refinement is halted. Otherwise, refinement continues. Then, using those boxes, a pseudorandom sample point is selected near the center of each box to provide inputs for CDT (Figure 13).

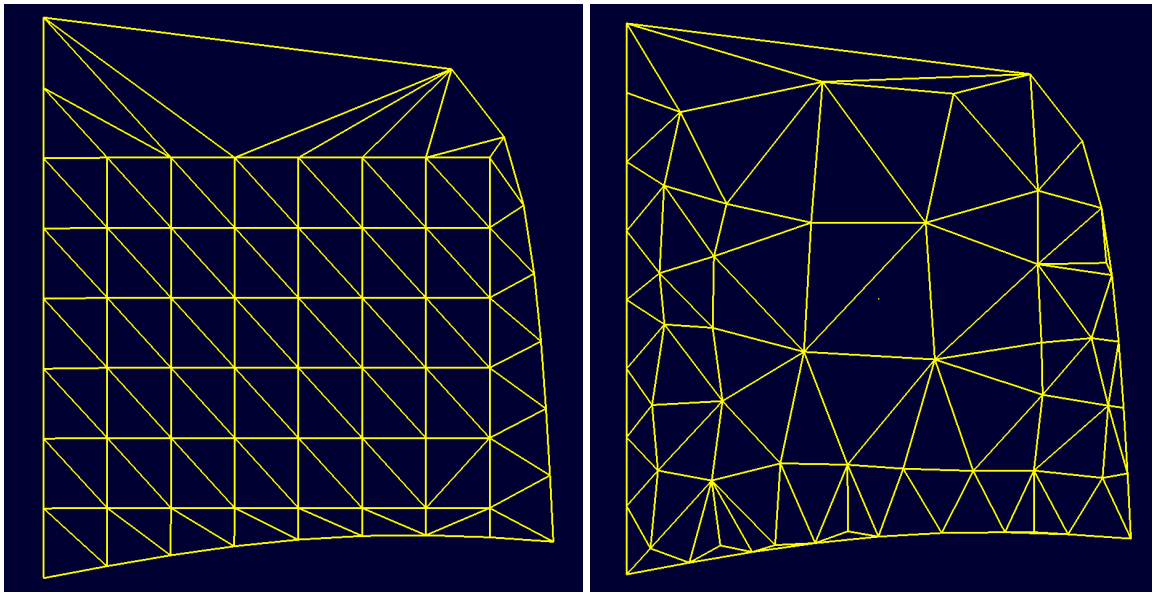

Figure 13. 2-D triangulation outputs with (left) original grid sampling and (right) the new pseudorandom adaptive sampling (NIST2 Face 246) (NIST, 2014)

In addition to these concerns, surface sampling near edges holds another peril in the form of selecting a surface point too close to the 3-D BRep face edge for the local tolerance used to linearly approximate the closest edge (Figure 14). In such a case the only valid triangle that can be formed in 3-D will have a normal direction opposite that of the NURBS surface normal. In effect, the triangle will be incorrectly representing the surface in that vicinity.

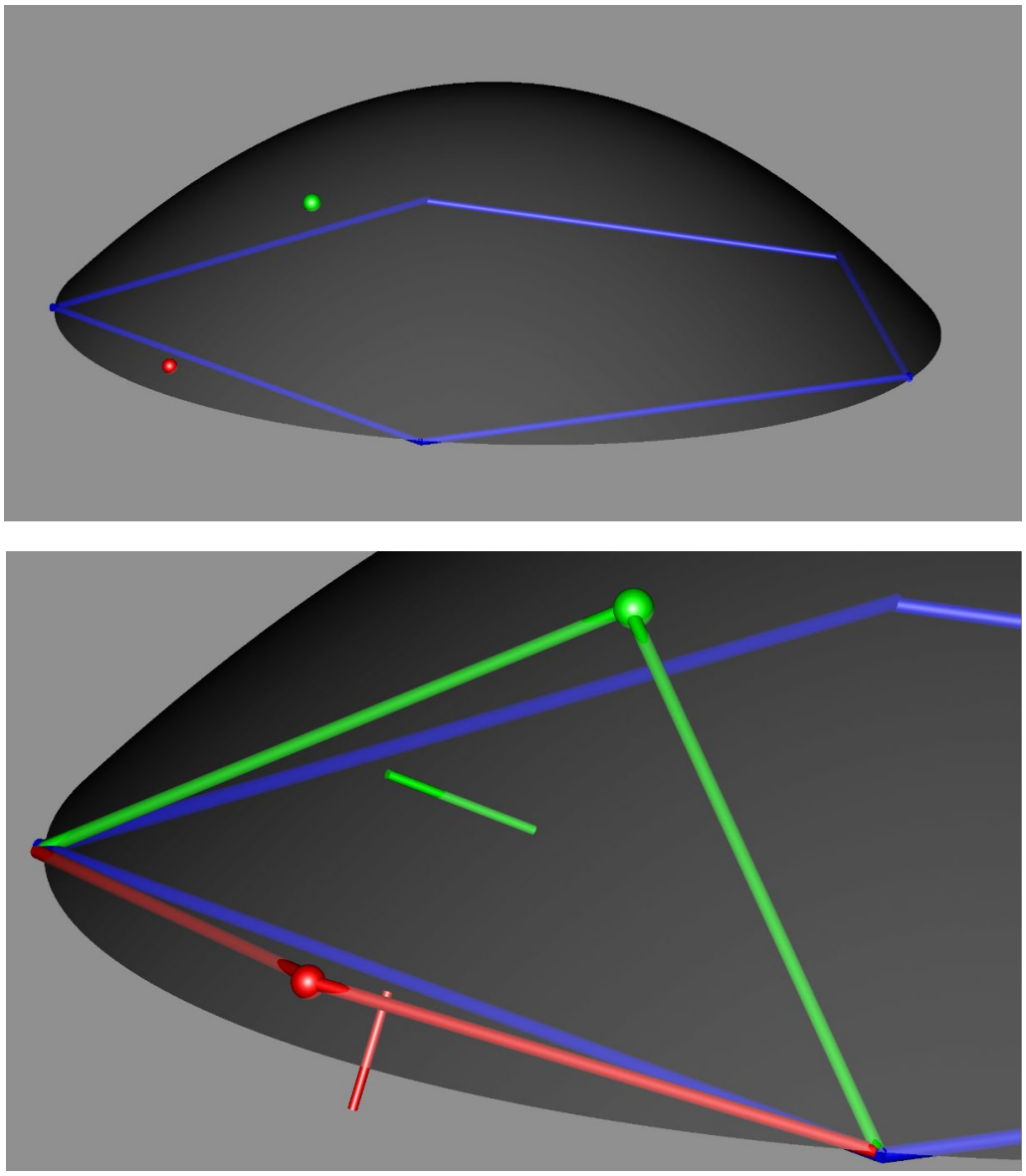

Figure 14. Example illustrates the problem of flipping triangle direction relative to the surface if points are sampled too close to an edge curve with a coarse approximation. Top image is an overall view of the original surface (gray), the edge sampling polyline (blue), and two candidate sampling points (green and red). Surface normals are pointing up (toward the top of the image). The bottom image draws two triangles connecting each point to the nearest edge curve segment, and shows the two normals (green and red lines). For the green sample point, the normal of the triangle is relatively close to the original surface's normal at that point. For the red sample close to the edge curve, the triangle normal is

becoming very different from the original surface normal, making it a poor approximation of the original surface.

To avoid introducing such surface points (which cannot be readily detected in 2-D), 3-D bounding boxes are defined that enclose the 3-D edge segments and additional boxes bounding the neighborhood of vertex points. These boxes derive their dimensions from either the source edge-curve segment or (for vertex boxes) the two attached curve segments. They collectively form a bounding volume enclosing all edge curves, defining a region of space in which no surface points should be added by random surface sampling (Figure 15).

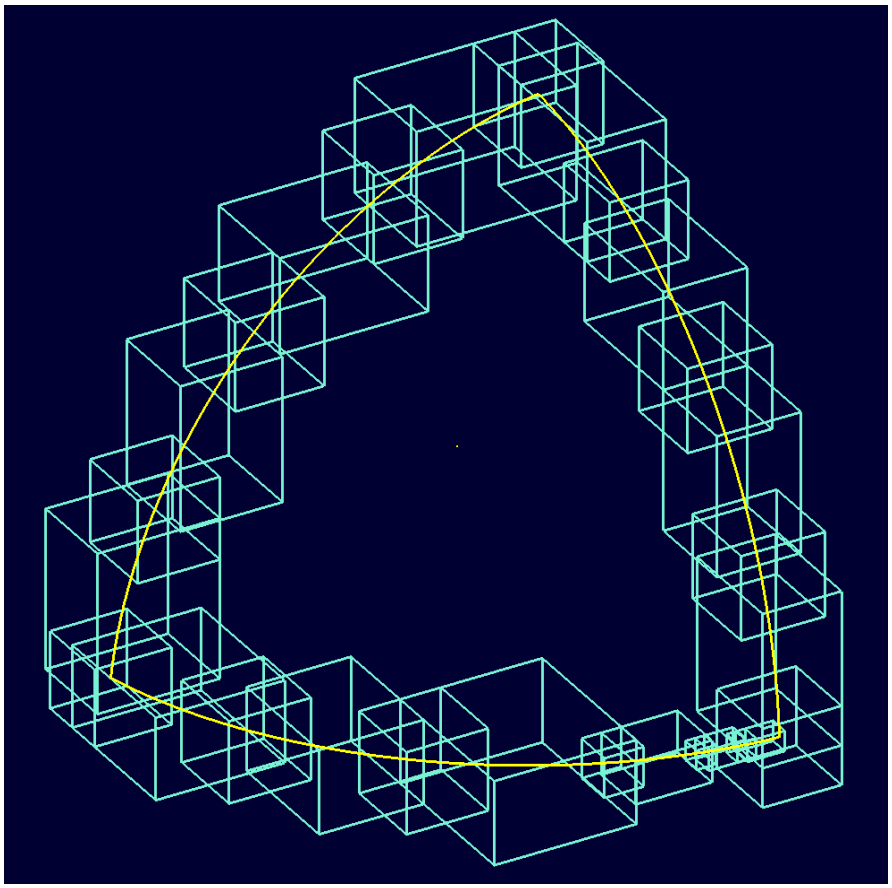

Figure 15. 3-D trim curve RTree box leaves, used to avoid sampling surface points too close to edge curves in 3-D (NIST2 Face 246) (NIST, 2014)

This new approach improved results, but it still did not resolve all validity issues. Attempting to micromanage the 2-D CDT algorithm to select "appropriate" triangles based on 3-D criteria would have required a deep study of the details of the Domiter–Zalik triangulation algorithm, with no way of knowing in advance if such guided selection would prove practical. Before attempting such an introspection, it was decided to first implement and test a more straightforward approach: locally removing connected patches of problematic triangles from the mesh; reprojecting the points from those triangles into a new plane fit the points in question; and

retriangulating those points to generate new mesh patches in a parametric domain more appropriate to the local neighborhood of the 3-D mesh. The proposed repair method also needed to be automatic; otherwise much of the labor saving advantages of the proposed conversion workflow improvements would be nullified. This task proved to be one of the most significant challenges of the project.

Mesh validity information identified various categories of triangles associated with one or more problematic aspects of the mesh (for example, Figure 16 identifies triangles in the neighborhood of a singularity in NIST2 Face 4 based on connectivity and face angles), but that information by itself is not enough to be able to effect a local mesh repair. To generate new triangles locally, a new bounding polygon on the mesh must be found that contains the area in need of repair and does not itself incorporate edges from problematic mesh triangles. This polygon must be closed, projectable to a plane and not self-intersecting. By definition, the topological information from "bad" triangles is suspect and thus not a reliable basis for creating such a polygon.

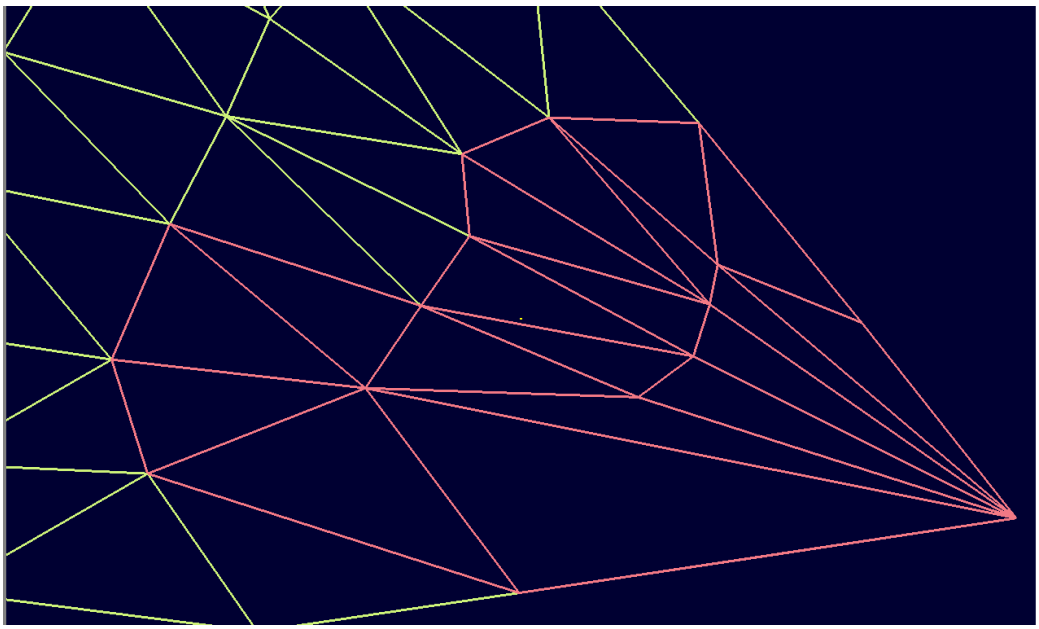

Figure 16. Identification of triangles near a singular vertex (the rightmost vertex in this image) (NIST2 Face 4) (NIST, 2014)

After attempting and discarding a number of approaches, a practical approach was found that starts with a known "good" triangle near the region of the problem, using its edges to define a valid polygon, and then grows that polygon out along the mesh surface (walking over any bad triangles and continuing until they were interior to the polygon). Angle- and validity-based termination criteria, as well as a series of specific criteria for which triangle should be incorporated into the patch area next to avoid infinite loop conditions, eventually resulted in an ability to generate bounding polygons and interior point sets. (For more details, refer to the

BRL-CAD source code in src/libbrep/cdt. The loop building logic is rather involved and needs to handle a number of different problem cases.)

Once the bounding polygon and interior points are successfully identified, the next step is to find a best-fit plane and project the points in question onto it (Figure 17). Once in the 2-D parametric space of the planar projection, CDT can be applied again to just the points and boundary of the patch in question. Once CDT is complete, the original mesh triangles are removed and the new ones evaluated into 3-D and substituted (Figure 18). Because the polygon was formed using edges and vertices shared with other triangles in the mesh and the CDT incorporated all polygon edges into the new triangle set, the new patch matches up with the older portions of the mesh and produces a valid result.

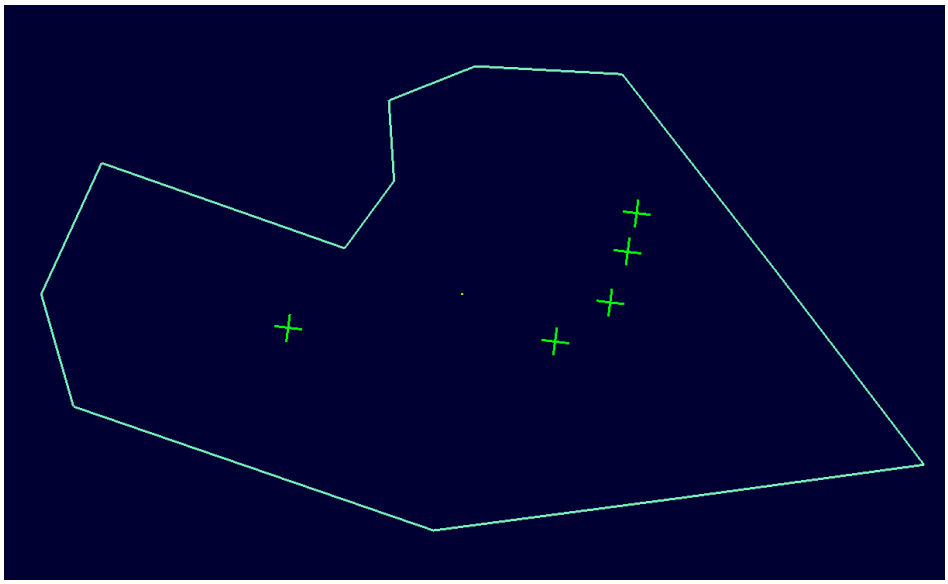

Figure 17. Projected bounding polygon with interior points. This is a 2-D data set suitable input for triangulation that can be mapped back to the original 3-D points (NIST2 Face 4) (NIST, 2014).

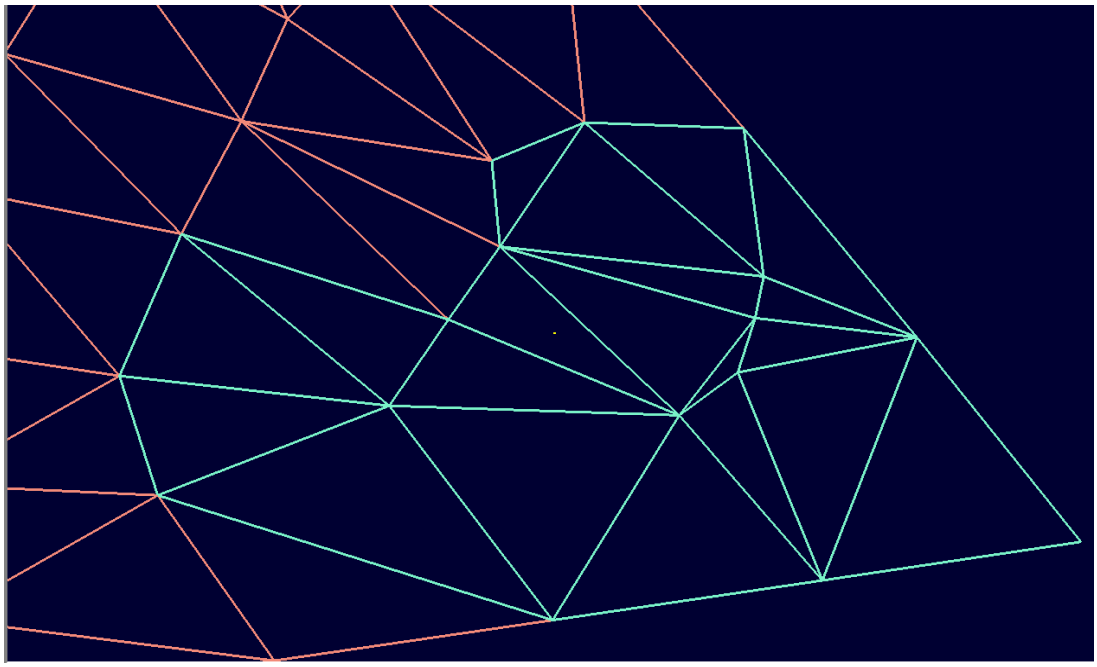

Figure 18. Polygon triangulation reassembled into the original mesh (NIST2 Face 4) (NIST, 2014)

A comparison of the original and new mesh triangles makes the topology changes clear (Figure 19). Because this parameterization is a local planar projection of the 3-D data of the mesh (as opposed to the original NURBS parameterization), the triangulation answer tends to be more optimal for the 3-D mesh once the new triangles are translated back into 3-D. In particular, for this example, triangles that were narrow and crowded toward the singularity have been removed.

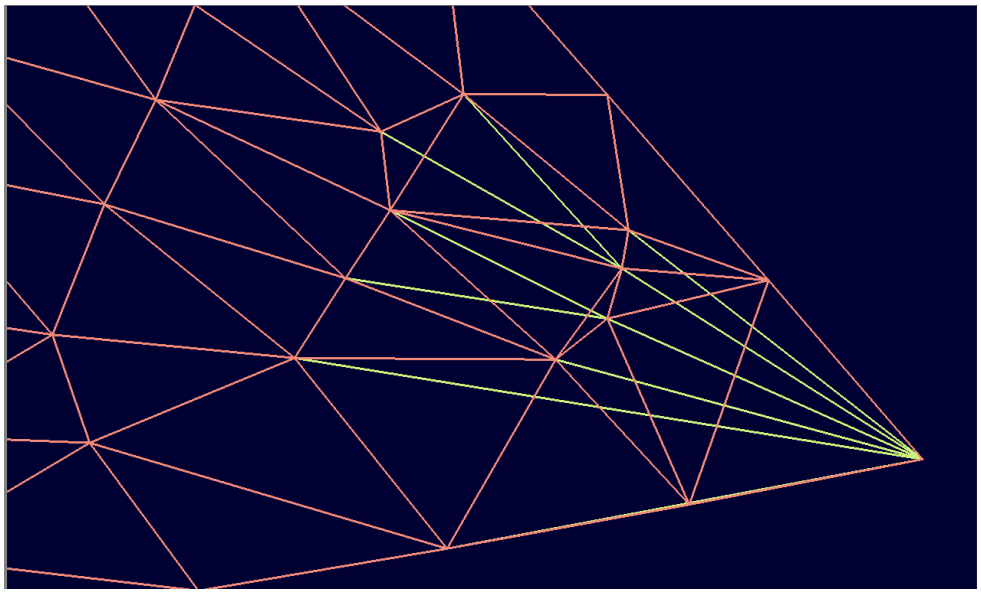

Figure 19. New mesh triangles (orange) and original triangles (yellow) (NIST2 Face 4) (NIST, 2014)

2.4 Point Sampling – Ensuring Sufficient Density

In V/L applications it is common to want relatively sparse meshes to represent objects, but vehicles often contain many thousands of highly detailed parts, and working with high-density meshes can quickly exhaust computational resources. The countervailing concern is that successfully sampling NURBS BReps sparsely while producing valid meshes is something of a challenge. A sampling that is too sparse can produce a mesh that will be degenerate or invalid regardless of any mesh repair routines. Specific examples include the following:

1. A half-cylinder sampled using only edge points. Without any surface sampling at all, the linear edges triangulate to a plane and the end caps incorrectly form triangles with other points on the same end cap.
2. Long, thin curved surfaces with insufficiently fine sampling will produce triangles whose distance and normal vectors are far enough from the original NURBS surface to introduce visual artifacts.

Clearly there exists (at least for some surfaces) a minimally sufficient sampling density below which unacceptable triangulation results are assured as well as specific sampling locations (dependent on tolerance settings) that need to be either included or avoided. To practically address these limitations without requiring user input, a heuristic was devised using wide bounding boxes to the curve edge segments in 2-D. These boxes are assigned to each segment, but unlike the tighter boxes used for checking if points are on bounding polygons in 2-D, these boxes have their width set at half the segment length. As in the surface point sampling

application, these trim curve boxes are then stored in an RTree to support fast nearest-neighbor lookup.

During the initial curve breakdown that precedes surface interior point sampling, each curve segment 2-D box is checked to see if it overlaps with any boxes other than its immediate neighbors. If it does, and if a maximal depth limit has not been reached, that segment is flagged for splitting. This process repeats either until all boxes are clear of overlaps or until a depth limit is reached (Figure 20). In this fashion, an initial set of curve segments can be identified that are much more likely to produce a reasonable visualization of the mesh.

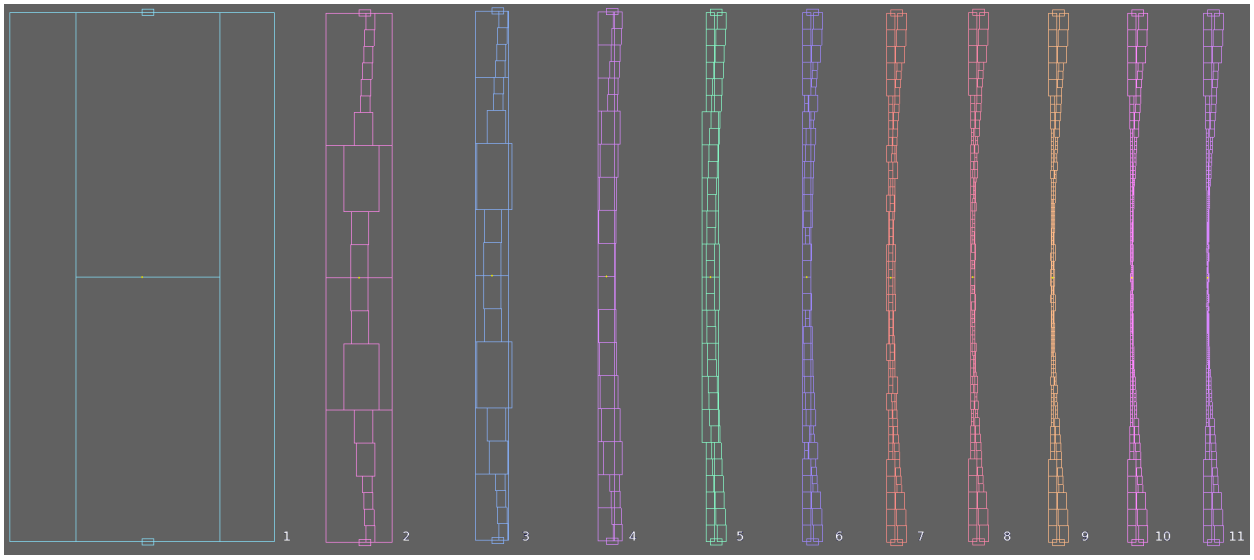

Figure 20. Visualization of the 2-D parametric-space edge-curve bounding-box refinement process for NIST2 Face 229. Left to right: the initial breakdown begins with coarse bounding boxes, with overlap testing being used to identify necessary refinements. Starting at refinement Step 6, boxes toward the top and bottom of the face stabilize as they clear their neighbors. Refinement toward the center continues to deeper levels, where the face edges are closest together.

These 2-D boxes also allow for the selection of interior points near edge curves that do not have the problem of a poor relationship with the edge. The local neighborhood of the box in the direction interior to the segment's box can be sampled at increasing distances from the segment until a suitable interior point is found (or it is determined that the edge is too close to another segment to be able to define an interior point). This point (and only this point) then becomes the representative point for that portion of the surface and other surface points sampled inside the box are rejected. This ensures that there will always be some interior points to force the mesh to represent the interior surface in from the edge curve while avoiding points that could produce flipped triangles.

2.5 Solid Meshing of NURBS BReps

In combination, these techniques were able to produce valid meshes for all of the solid NIST sample geometries (NIST, 2014). To illustrate the difference in meshing outputs, Figure 21 compares the original triangle mesh generated by the original NURBS meshing logic in BRL-CAD with the results generated by the new routines. Note in particular that even the coarse triangulation performed using the new techniques (2) is fully watertight.

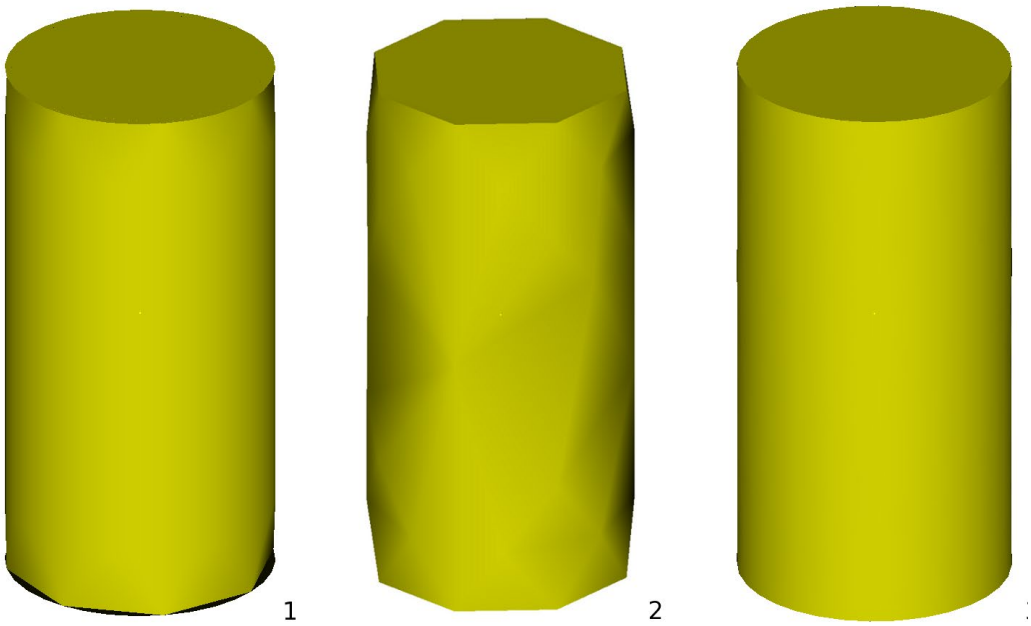

Figure 21. Example triangle meshing outputs for the original example cylinder: 1) the original nonwatertight output; 2) demonstrates the coarsest possible triangulation using the new logic. In particular, note that the top and bottom faces of the cylinder now align with the center mesh and respect the coarse tolerance used for that face; and 3) represents the output from the new routines using a tighter tolerance on the triangle normals. This uses more triangles, but results in a smoother mesh that is still watertight and valid.

With valid meshes available, work then shifted to overlap detection and processing.

2.6 Detecting Overlapping Triangles

By definition, any solid overlapping mesh will have triangles that intersect with some of the triangles in the mesh with which it is overlapping. The basic input for overlap resolution is thus an array of valid mesh objects generated from their corresponding NURBS BRep solids. Because these meshes are generated in isolation, their meshes do not take cognizance of their local neighborhood and overlaps may exist between them. The work of the overlap resolution routine is thus to either 1) prove no such overlaps exist or 2) identify those overlaps and apply selective refinement of the meshes to clear them. Because the meshes generated for these

purposes have carefully retained information that associates them with their specific source NURBS BRep data, refinements using that allow for a degree of improvement not possible using generated meshes in isolation. The NURBS BRep objects do not overlap (a precondition of this process) and thus constitute a ground truth that may be relied upon when local mesh alterations are necessary.

The concern here is with volumetric overlapping between meshes; that is, nonzero volumes in space claimed by one mesh inside the volume of another mesh. Triangles aligned along surfaces but not intruding into their neighboring mesh are not considered overlapping for these purposes even though such triangles may report overlaps to interrogating rays intersecting both triangles in the shared plane of such configurations. Such tangent ray overlaps could also occur when interrogating the original NURBS BRep models. Interrogating applications must either develop multi-ray based testing to identify and handle such issues or develop conventions for how to process such rays. The goal for this work is to achieve meshes where *only* such tangential rays will produce overlap results. Any query testing for volumetric interference should find nothing introduced due to meshing effects.

The basic triangle/triangle overlap test implemented in BRL-CAD uses the well-known implementation by Möller (1997) and tests using meshes confirmed by BRL-CAD's rtcheck tool to have overlaps demonstrate successful identification of interfering triangles (Figure 22).

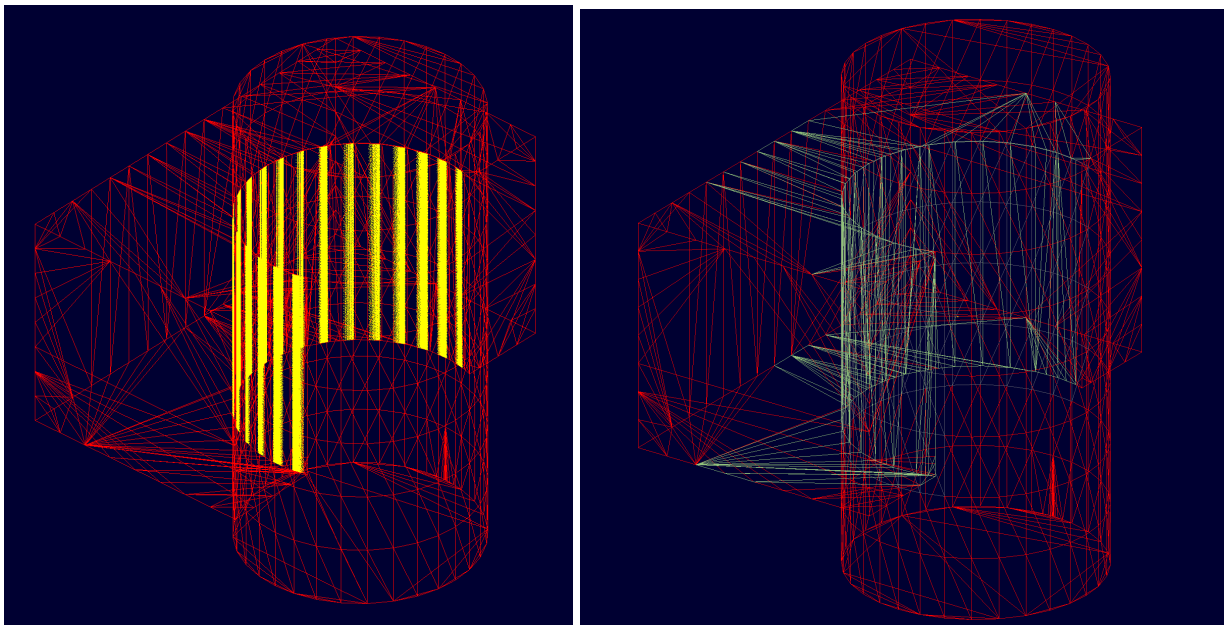

Figure 22. Overlap detection test case: (left) BRL-CAD's rtcheck report on what volumes are overlapping, and (right) triangles colored green are detected as overlapping. The results are consistent, indicating a successful integration of basic triangle level overlap detection.

For testing purposes, we use the meshes from the BRep object pair in Figure 23 as the primary input data set. These meshes are representative cases of classic triangle-induced overlapping but also constitute a simple enough input to make rapid iterative testing possible.

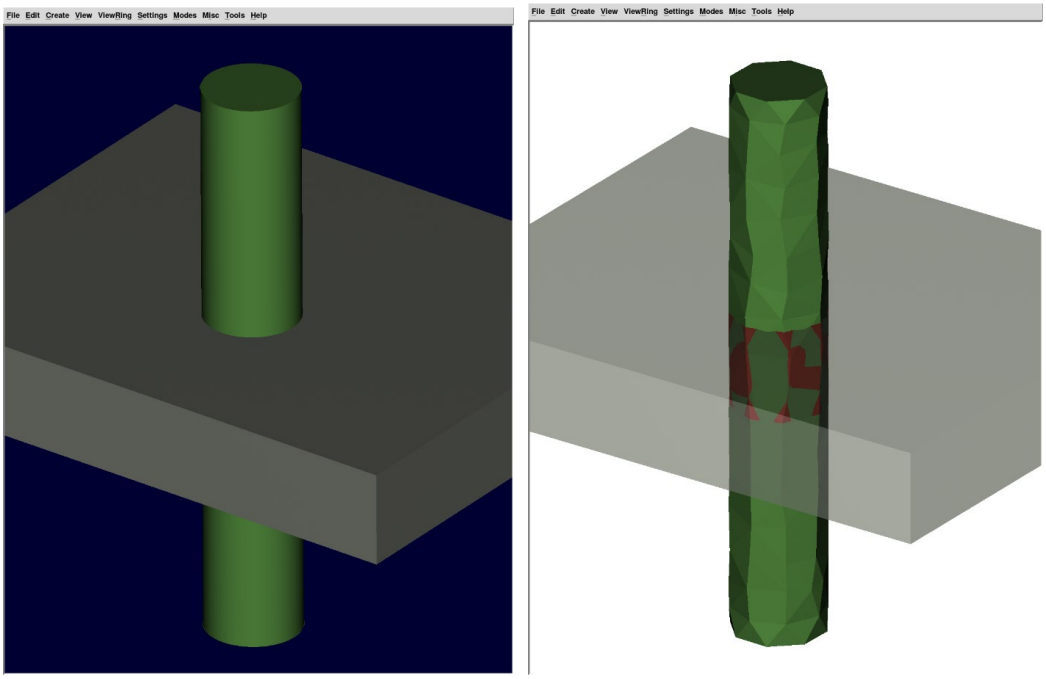

Figure 23. Example illustrating a typical triangulation induced overlap: (left) a rendering of the original NURBS based model, which does not have geometric overlapping, and (right) a basic triangle mesh generated from this example, without any refinement due to overlaps. Red represents volumetric overlap between the meshes.

With Möller (1997) overlapping triangle detection working, the next question became its efficient application to input mesh sets.

The highest-level operation when processing a set of mesh objects is to identify pairs of overlapping BRep face bounding boxes. If the bounding boxes of the faces are disjointed, there is no information to be had from their triangles from a refinement standpoint and no further testing is necessary. It is only when the bounding boxes overlap that we need do any further work. This allows for the elimination of a great many face/face tests and is the first optimization step applied in overlap processing.

Once potentially interfering face pairs are identified, it is necessary to determine if overlaps exist at the triangle level. However, for large meshes, testing every triangle against every other triangle [the naïve $O(n^2)$ algorithm] quickly becomes prohibitively slow; a more refined approach is needed. The RTree data structure already in use for surface point selection was a logical starting point for a more-optimized workflow since it is designed specifically for such

operations. However, for overlap detection of triangles, the 3-D edge trees already created were not enough. Trees containing all active mesh triangles that might potentially interfere with another mesh were needed.

After some development and testing iterations, it was decided to refactor the mesh data structures to use the RTree container as the primary container defining which triangles are active in a given mesh, maintaining the RTree's state during triangle insertion and remove operations on the mesh itself. This avoids any need to worry whether a given mesh's RTree correctly reflects the state of the mesh at any particular point in the process. Any change to the mesh will automatically update the RTree.

Given current RTrees for the meshes, those trees can be compared to find pairs of overlapping leaf bounding boxes (or rule out such overlaps, if the meshes do not in fact interfere). Those pairs are in turn tested with the Möller intersection routine to make a final determination of whether they do or do not overlap (Figure 24).

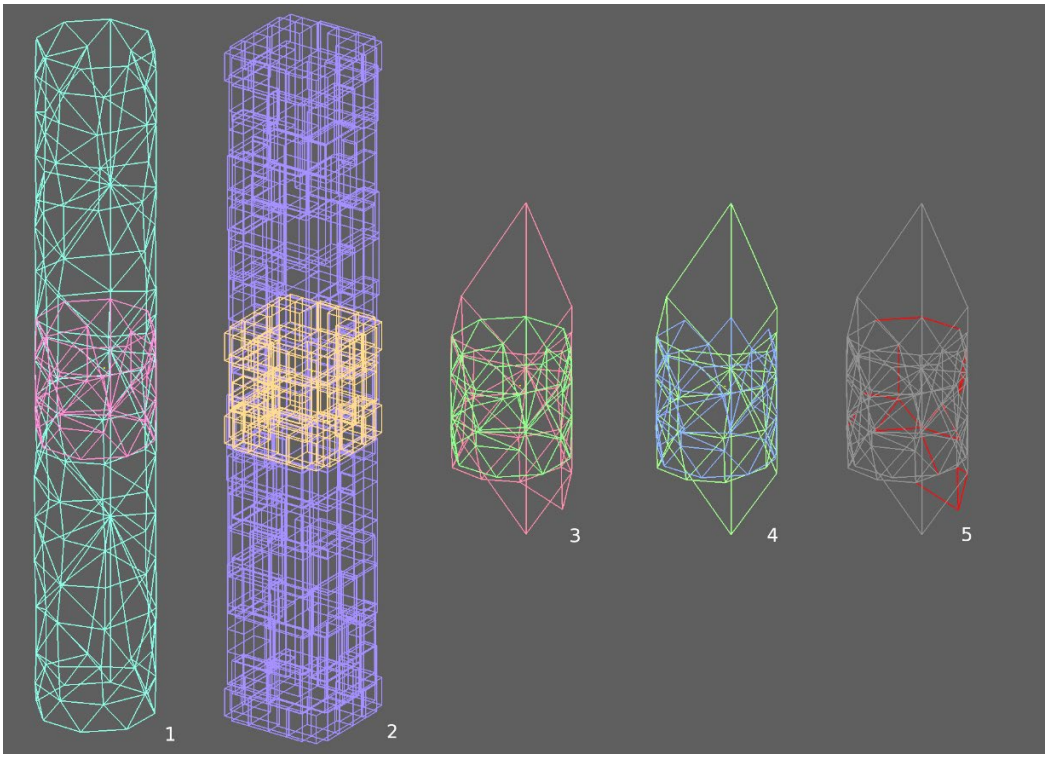

Figure 24. Illustration of the initial overlap detection process. 1) Two mesh faces have their 2) RTrees compared to identify 3) nearby triangles. 4) Those nearby triangles are then compared to identify which triangles genuinely have overlapping relationships per Möller (1997). 5) The rightmost illustration highlights differences between the initial triangle test set and those with genuine overlaps. Triangles rejected by Möller are colored red.

Having identified the sets overlapping triangles, the next question is what to do with the information to improve the meshes. It was immediately clear that triangles associated with BRep edge curves would pose a problem for any sort of refinement operation, as changes to the shared edge would necessitate changes not simply to the face associated with the triangle in question but also to triangles in the adjoining face. This nonlocality is necessary to maintain the watertight property of the mesh during refinement, just as the original meshing of individual faces needed to be aware of shared edge points to establish that property in the first place.

At an implementation level it proved necessary at this stage to add mappings connecting triangle edges with their associated face-edge segments and, vice versa, to allow local mesh operations to identify the local triangles and vertices they would have to manipulate for any given operation.

With those preliminaries established, the stage was set for the most challenging aspect of the overlap problem: devising and implementing strategies for minimizing or (ideally) clearing overlaps. There are a number of possible strategies, most of which have different tradeoffs in terms of complexity of implementation, quality of output, and robustness. Full exploration of these issues was not possible in the time available. This report details explorations made to date and the current working capabilities in BRL-CAD, with some thoughts on what future improvements might be made.

2.6.1 Edge-Length-Based Overlap Refinement

The simplest approach available for this purpose is to directly use the overlapping triangle sets identified by the RTree comparisons to create smaller triangles from the larger triangles until the longest edge length of an overlapping triangle is smaller than the user-specified tolerance. Since the input NURBS surfaces are assumed not to overlap, the deepest possible triangle penetration into an opposite mesh after refinement to that dimension would have to be less than the user's specified tolerance.

When performing mesh alterations, it is always good practice to start with any BRep face edge segments that require modification. Because changes to those edges impact two faces rather than one, they must be dealt with first in order for subsequent processing steps to assume safely that no other operations in the refinement cycle will change a given face's mesh. (This property, although useful simply for the sake of clarity when debugging, becomes *very* important in potential future work to parallelize the processing of individual mesh faces.)

Because a simple splitting of all edges longer than the specified tolerance at their midpoints would not converge the mesh toward the original NURBS surface if the midpoint of the edge line segment is used, the refinement operation instead uses an implementation of the closest-point-on-NURBS-surface routine (originally created for use in the BRL-CAD step-g converter work in 2014) (Figure 25). The edge midpoint on a long triangle edge is calculated from the line

segment, and that point is used as the query point for the closest-point-on-surface routine to identify a suitable point that is actually on the original NURBS surface. Once that point is identified, the two triangles associated with the original edge are replaced with four triangles using the original vertices and the new vertex at the new surface point. It is crucial that all edges interacting with the original mesh neighboring triangles be in their original directions to ensure the mesh remains topologically valid. Unordered edge-based triangle look-ups must work correctly after a splitting operation for refinement to continue, and also to ensure the final output mesh will be valid.

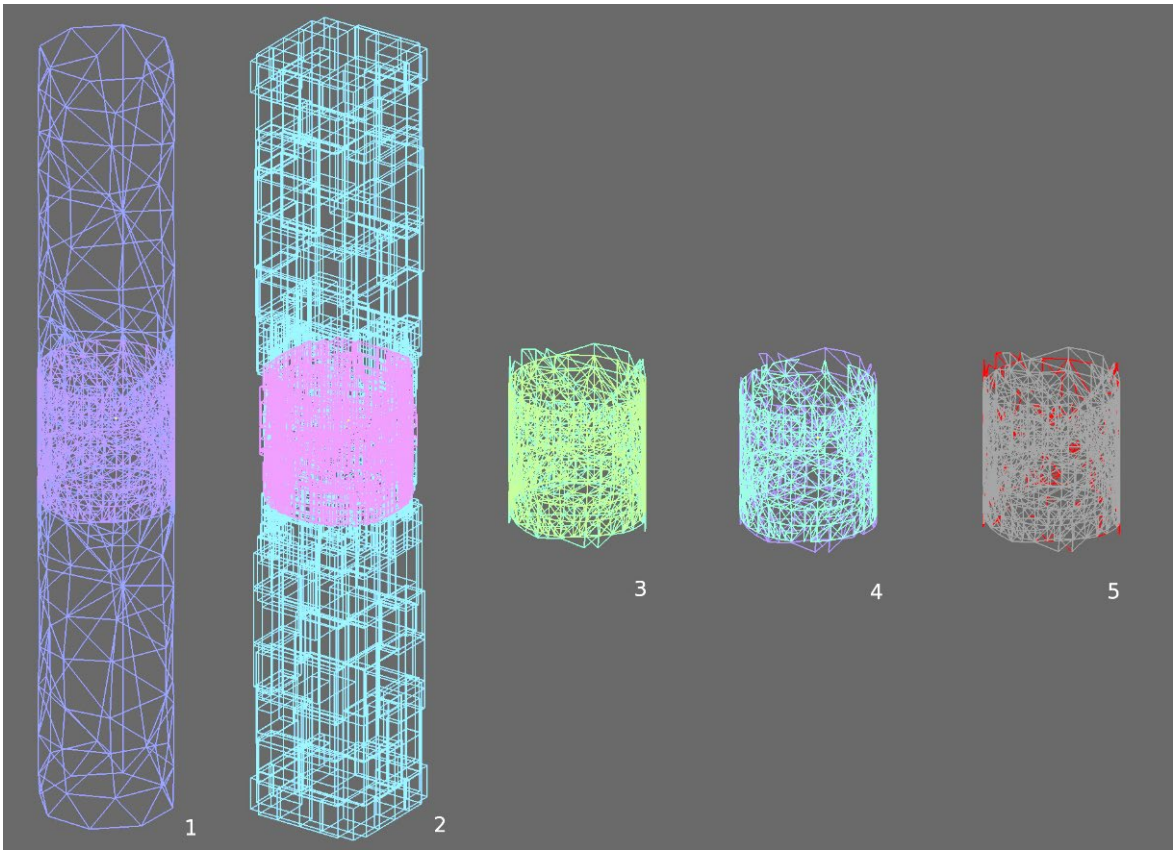

Figure 25. Snapshot of mesh refinement process after several detection and splitting cycles. Both meshes are being refined only in the area of mesh overlap. Non-interfering areas remain in their original configuration.

A naïve splitting of all edges longer than the user specified tolerance proved less than ideal in terms of the mesh quality produced. To converge more toward equilateral triangles, the overlapping triangle sets were processed to build up two sets of triangle edges: 1) all those longer than the specified tolerance, and 2) those containing any edges that were the shortest edge on one or more triangles. Once the initial sets were built, the set of shortest edges was removed from the long edge set to identify the final set of edges to be split. This procedure produced better triangle shapes as the mesh converged. Figure 26 shows the results of several overlap detection and processing cycles on the Figure 23 example.

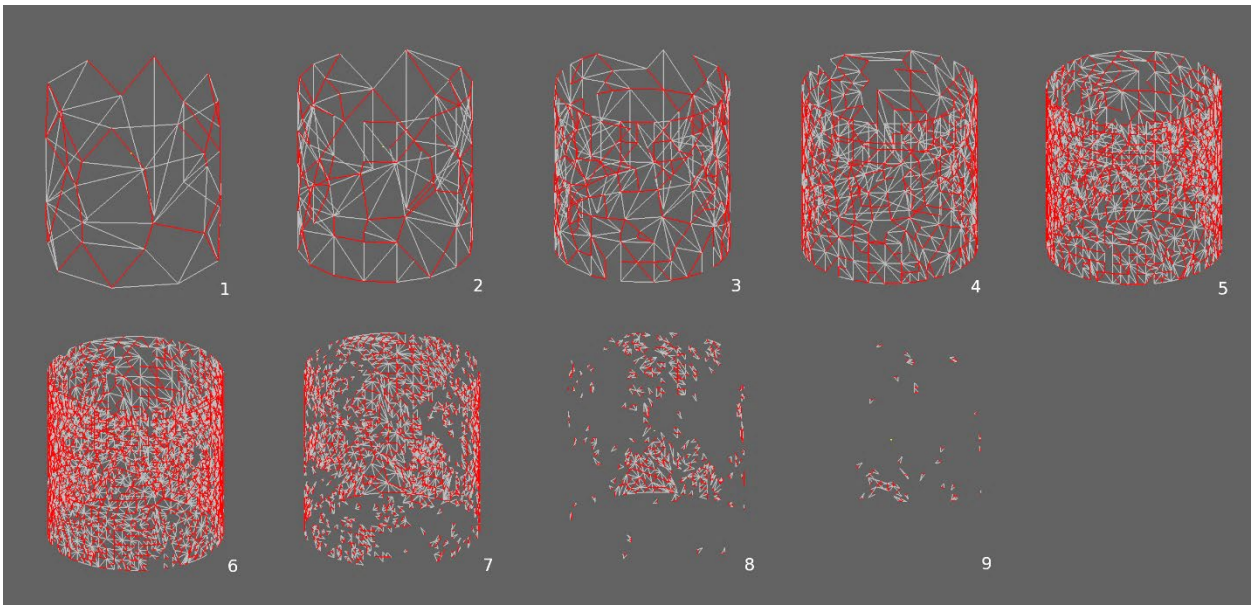

Figure 26. Edge refinement on one of the example BRep face meshes. Gray edges are to be split. Red edges were initially flagged as edges to split, but were filtered out due to being the shortest edge on a triangle. As the cycles progress, triangles become smaller and are filtered out of the processing set due to their edge lengths falling below the user-specified threshold length.

A byproduct of this approach to mesh refinement is that it also tends to reduce gaps between meshes. As triangles are refined toward the surfaces, they tend to pull in additional triangles from the opposite mesh in overlap testing, which has an overall effect of reshaping the local mesh to more closely match the surface (Figure 27). Gap removal is not currently a specific goal of this logic, but such a result is definitely a desirable property when preparing V/L meshes.

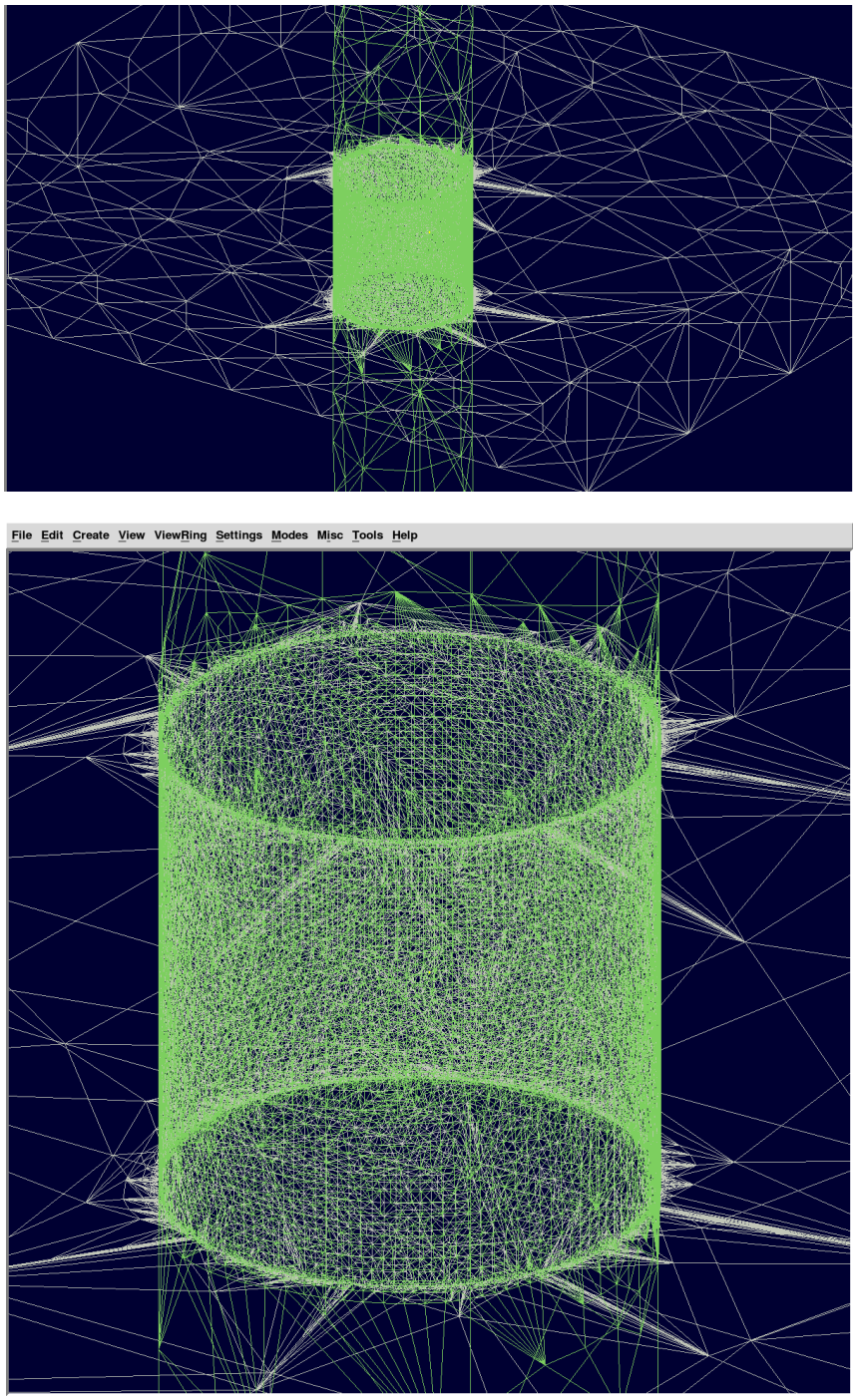

Figure 27. The final meshes output by the conversion process. Due to a tight tolerance specification, the mesh has been extensively refined in the mesh overlap area. (top) An overall view showing the contrast between refined and unrefined mesh areas, and (bottom) a close-up of the refined area.

In the event that an input has a genuine overlap in the original BRep input set, this method of refinement will simply continue splitting triangles on the boundaries where the faces overlap until the user specified limits are reached. It will not reduce overlap volumes. A closer inspection of the overlapping cylinder from Figure 28 illustrates that the refinement can still provide a visual indicator of where the BRep faces are interfering (Figure 29).

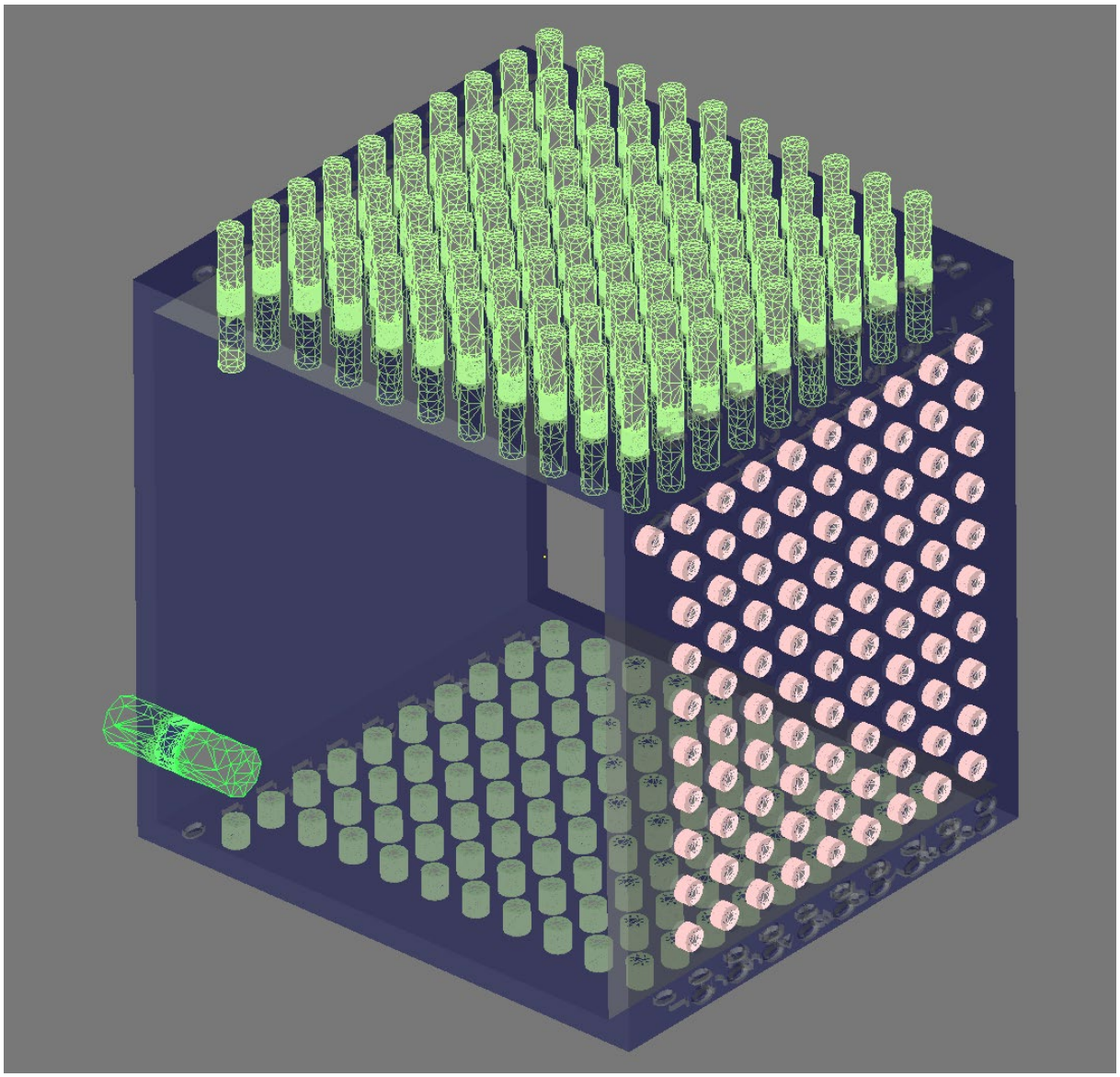

Figure 28. Example refinement with a more-complex multi-object test case. All BRep cylinders except the large one in the lower left fit precisely into holes within the larger matrix and have been refined in those interference areas.

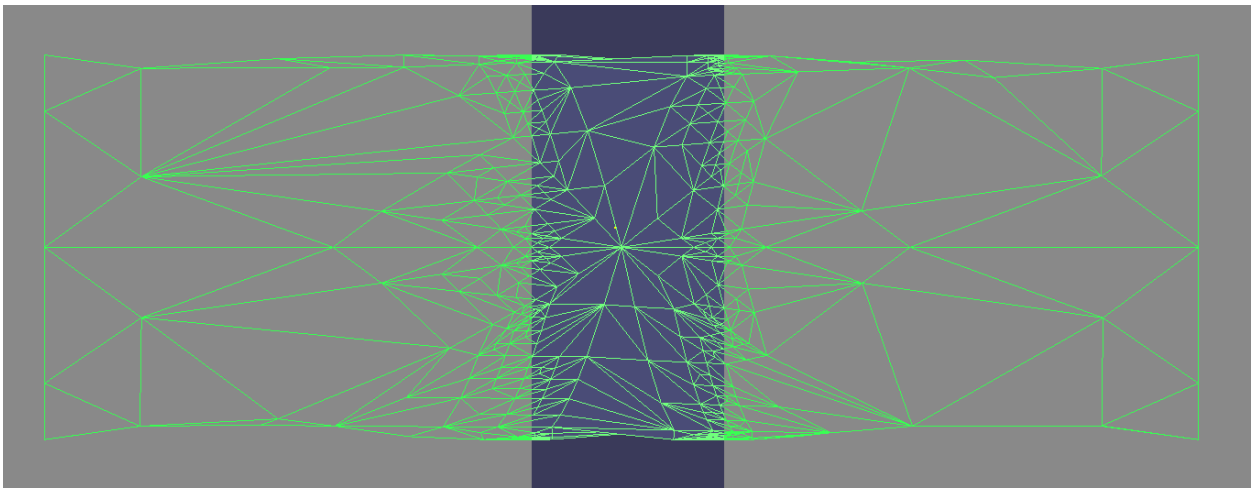

Figure 29. Example refinement on an overlapping set of BRep objects where the overlap is in the original NURBS objects and not due to the mesh. The refinement clusters around the face edges and does not extend to the interior of the interference volume. This is because within the overlapping volume, the faces are not close enough for their triangles to interfere, even though they are inside the overall mesh.

2.6.2 Intruding Vertex-based Overlap Refinement

While the edge-splitting refinement has the advantage of strongly converging the mesh toward the original NURBS surfaces, it also has some drawbacks. In particular, tight tolerance specifications generate very large numbers of triangles. It is quite easy to inadvertently specify tolerances that will attempt the creation of meshes too large to successfully generate or process. Another drawback appears when rays are fired nearly tangent to overlapping mesh. Faces may still find long overlap shot lines very close to the interfering mesh faces, since edge refinement reduces the volume of overlaps rather than fully eliminating them. A more ideal solution would be to more selectively introduce mesh triangles only where needed to clear specific overlaps. An attempt was made to implement such a refinement structure, but as of this writing those efforts have not yet achieved results generally applicable to all input meshes. Efforts to date are outlined in the following, and subsequent sections will discuss measures that can potentially be undertaken to improve conversion yields.

To identify when and where we need to refine a mesh, the obvious starting point is to characterize which vertices from one mesh are intruding into or otherwise interfering with another mesh. However, early experiments proved this information was not sufficient by itself. More than just the intruding vertices would be needed to achieve a properly aligned non-overlapping mesh, and care would be needed to avoid introducing vertices in such a way that pathologically small triangles were created. Forcing a mesh to incorporate multiple points

extremely close together is a source of problems for mesh quality and subsequent processing, and a number of measures are needed to prevent this.

Once the problem of close vertices was clear, the first and simplest way to avoid the need for such vertices was to attempt to adjust existing vertices' positions on the NURBS surfaces so they aligned with each other rather than interfering. This could not only alleviate the need for close vertex insertion in many cases, but also would clear a number of overlap cases by simply shifting existing triangles into a non-overlapping configuration. The questions were 1) how to identify close vertices and 2) how to know how much distance was "safe" when it came to moving a vertex; any vertex move that might result in invalid triangles in the mesh is a nonstarter.

The answer to both of these concerns was to generate a triangle vertex bounding-box hierarchy stored in an RTree, with the bounding-box dimensions based on a fraction of the smallest edge length connected to that particular vertex (Figure 30). This meant that two vertex RTrees could be intersected, just as the triangle RTrees were intersected earlier, and nearby vertex pairs identified by their overlapping bounding boxes. If a vertex had very short edges associated with it, its bounding box would be small and only a very close point on the other mesh (or a point with very long edges and a lot of freedom to move) would register as being "nearby" in such a comparison. Since multiple vertices might be found that satisfied these criteria for any given vertex, in these cases the actual vertex distances were checked and the closest vertex used for the pairing.

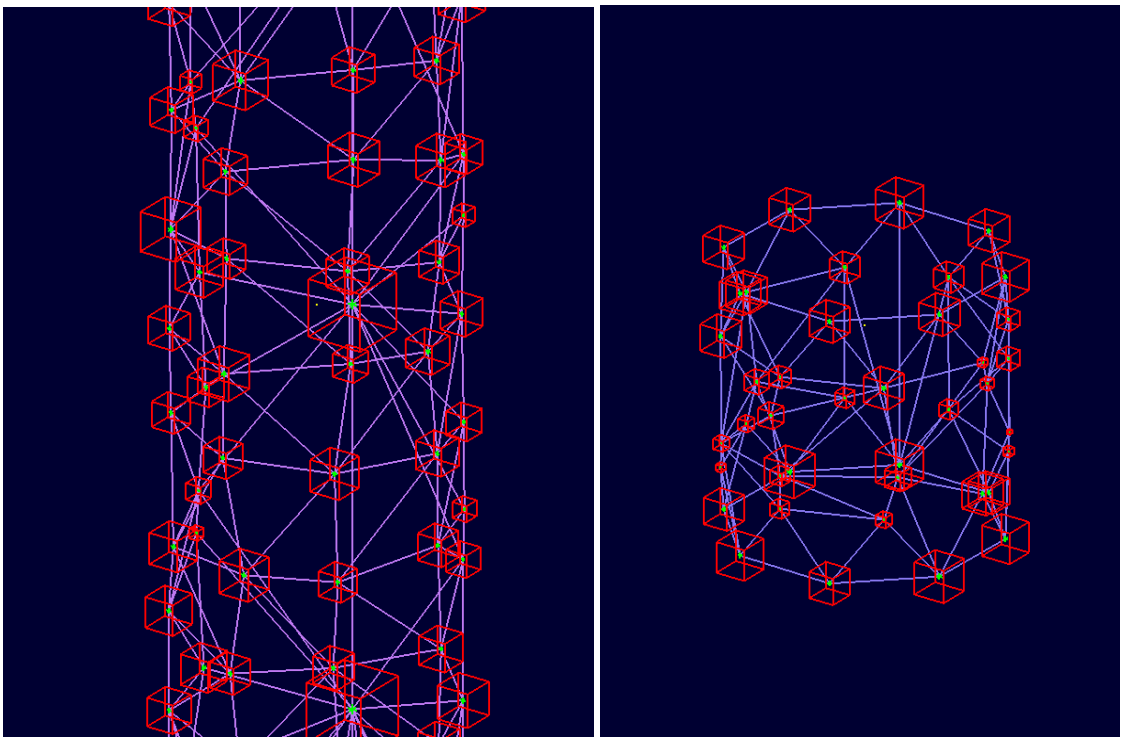

Figure 30. Example leaf bounding boxes from two mesh vertex RTrees, with dimensions of each vertex's bounding box based on connected edge lengths

After pairs of vertices were identified, a weighted average point was calculated between them, with the weighting factor being a function of how large a bounding box was associated with each vertex. In the case of one large and one small box, the weighted average point would be much closer to the vertex with the smaller bounding box since it has less ability to shift without mesh damage. Once that average point is found, the closest surface point on each NURBS face is calculated and the two vertices in question are shifted to these new positions (Figure 31). (This also necessitates a lot of updating to the triangle and vertex RTree data structures to keep them current.)

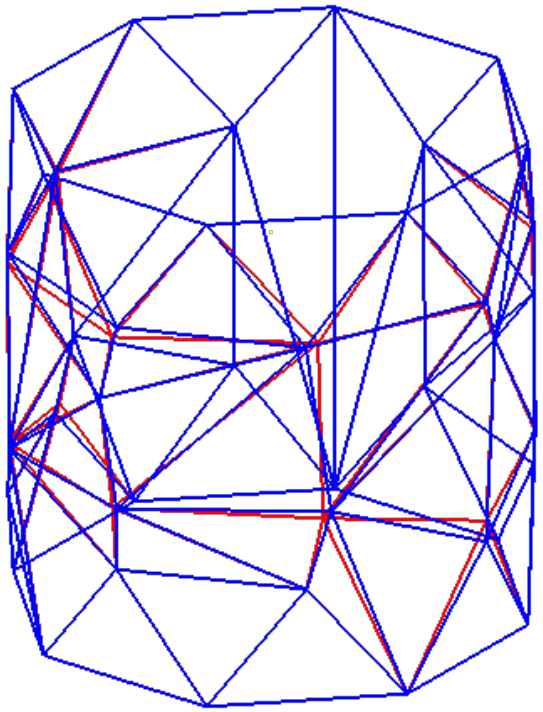

Figure 31. Example mesh change pre- and post-vertex alignment. Original edges are shown in blue and new edges in red. Often these adjustments are enough to clear previously overlapping triangles.

Ideally, this adjustment should be done for edge curve points as well, but the constraint in that case is that the points must remain on the curve and between the segment neighboring points. The BRL-CAD data structures are currently not set up to handle edge point location adjustments, as those points are treated as fixed in space. This limitation is thought to be a likely source of problematic points in subsequent stages of processing, but to date that has not been confirmed.

Vertex adjustments are a useful first step, but of course they are not sufficient if intruding mesh vertices are not close to a vertex in the opposite mesh. For these vertices, it is necessary to introduce new vertices in the opposite mesh to accommodate the intruders.

The question of when to introduce a new mesh vertex is actually more subtle than it might first appear. There are vertices that are not themselves intruding into the opposite mesh that may still need to be accommodated to avoid triangle overlaps. The following heuristics were eventually devised for the first refinement stage (red points in the example appear in Figure 32):

1. If a triangle overlaps with another triangle only at a single point, it is not considered to be an overlap for these purposes. A single point interference cannot be volumetric.

2. If a vertex is associated with two or more triangles that intrude into the opposite mesh, it is flagged for processing.

3. If a vertex is close to one of the edges of an intersecting triangle, flag it for processing. In particular, this is needed to deal with what would otherwise be nearly parallel edges in overlapping triangles.

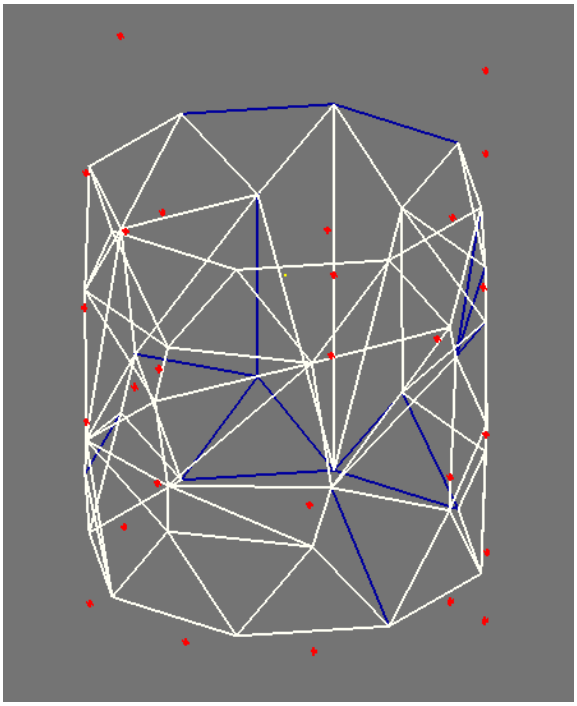

Figure 32. View of face meshes showing points of interest from other meshes (red) and overlapping triangles (yellow). Blue triangles are those not reporting overlaps.

For all vertices identified as being refinement vertices, RTree lookup is used to find the closest point on the mesh into which the vertex in question intrudes. In the first pass, reject any vertex that is too close to an already existing vertex in the opposite mesh per the bounding-box comparisons to avoid introducing two vertices close together on the mesh. If the vertex is not rejected, find the closest point on the opposite NURBS surface and add that point to the opposite mesh (using the bounding box of the originating vertex in the intruding mesh as an initial bounding box for the new vertex.) Once all new points have been added, for each new point find the closest edge in the mesh that will be accepting the new vertex. (This is currently done by doing an RTree lookup of the triangle tree, finding the close triangles, and then for each of those triangles characterizing the distance between the triangle edges and the vertex.)

If the closest edge to a vertex is also a BRep edge, that edge is split at the point on the curve closest to the vertex in question if that closest point is not already an edge segment start/end point. (This necessitated implementing a closest-point-on-curve routine for NURBS curves, translating the algorithm from the verbnurbs library [Boyer, 2020].) This step is performed to ensure the outer mesh boundary can properly enclose the new mesh points. After this step the closest edge to the new vertex point is updated to refer to the new interior edge added by the

triangles introduced into the mesh to accomplish the BRep edge-splitting operation. At this point all new vertices now have an interior edge as their closest edge (Figure 33) and triangle replacement can begin.

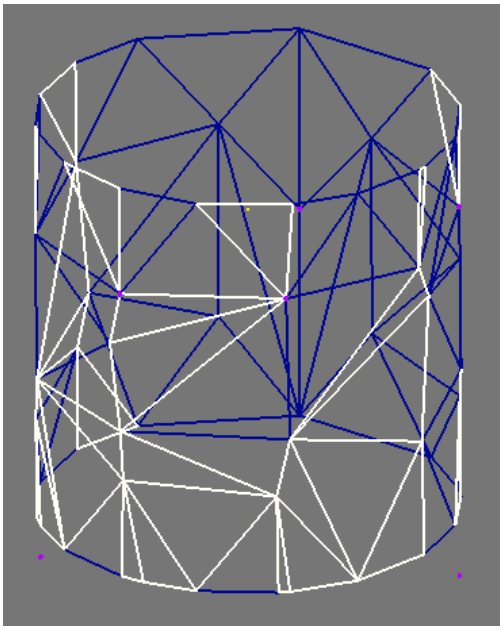

Figure 33. Triangle overlap state after edge refinement. Note the increase of face-edge points around the upper and lower edge curves of the cylinder.

For interior-edge-associated vertex points, the two triangles associated with the closest edge to a set of new vertex points are used to define a bounding polygon. That polygon and the new vertex points are projected to a local best-fit plane, and the mesh is locally re-triangulated to incorporate the new points into the mesh topology (Figure 34). Conceptually, this replacement procedure is identical to the repair operation used during the initial mesh creation, although in this case the interior points are new rather than being reused from the original mesh. Once new triangles are introduced, closest edge assignments for any remaining new vertices are reconfirmed to make sure those assignments have not been changed by the introduction of new edges.

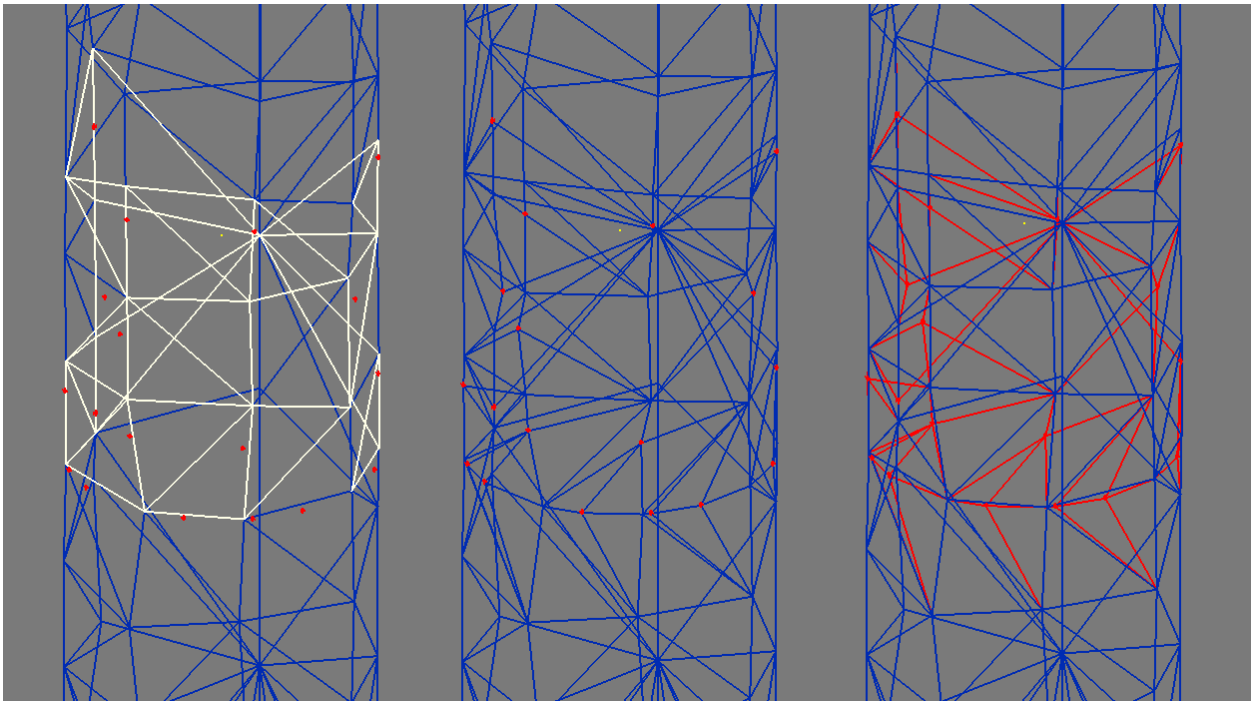

Figure 34. Triangle refinement driven by local re-triangulation incorporating nearest points to intruding vertices: (left) yellow triangles are overlapping triangles, and red points are vertices of interest; (middle) updated mesh and the points that drove the updates; and (right) visualization of mesh changes; blue are original edges, and red are updated/added edges

Once all new vertices are incorporated, the mesh is locally re-triangulated to optimize the new triangle edges locally around the adjusted mesh areas. Since only the two triangles associated with the original edge were replaced, the insertion can sometimes leave the mesh with locally sub-optimal triangles.

The previous process is run iteratively until no new triangles are being introduced. At that point overlaps are again checked for. If any triangles remain that are still involved in overlaps, subsequent refinement operations will need to ensure the closest points on the opposite meshes are present as vertices. To accomplish this, remaining vertices associated with overlaps are incorporated into the opposite meshes even if this means adding vertices close to existing vertices in the mesh; there are triangle interaction combinations that can require such insertions.

After vertex insertion is complete and we have aligned nearby vertices, triangle overlap detection is faced with a new challenge, or rather it must now address a situation that was previously overshadowed by other mesh problems. When overlapping triangles overlap only at their edge (which is now a probable interaction after working to align vertices), the overlap itself is not useful in characterizing the interaction. Some pairings can be eliminated based on the projection of the triangles into the average plane of the two triangle normals. If the triangles do not overlap

there, they are pointing in "opposite" directions and cannot define a volume. If the pairings do overlap in projection, they may (but are not guaranteed to) be part of volumetric overlaps. Figure 35 shows a case where two edge-only intersection triangles are part of an overlap, while Figure 36 shows two similar triangles that are *not* part of an overlap.

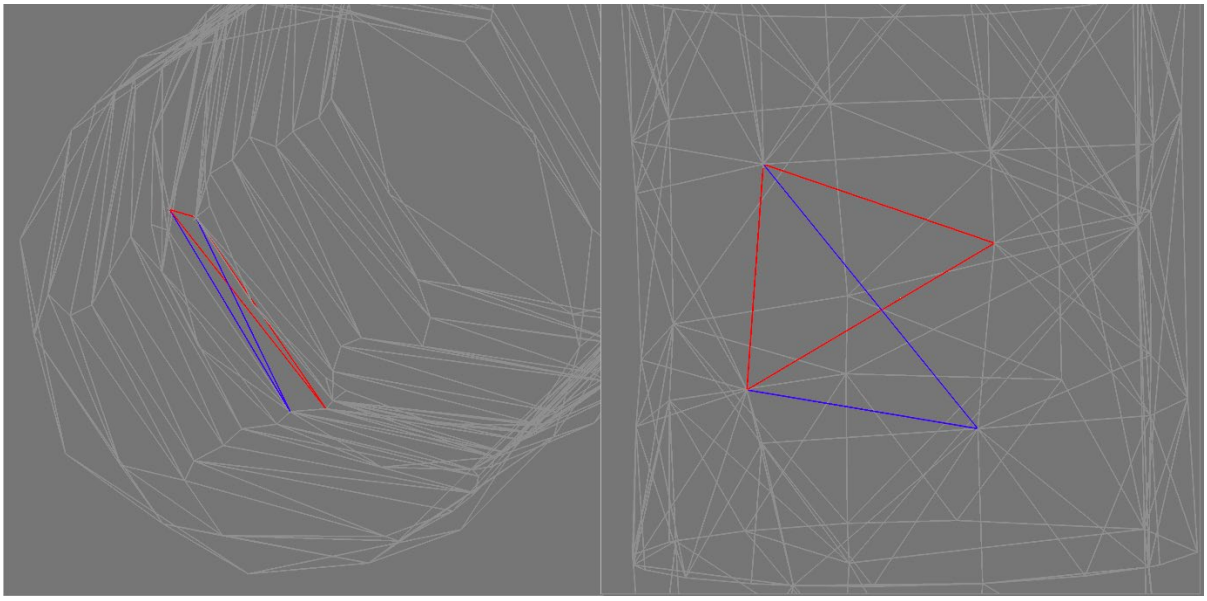

Figure 35. Edge-only triangle intersection pair (top view left and side view right) that is part of a volumetric overlap

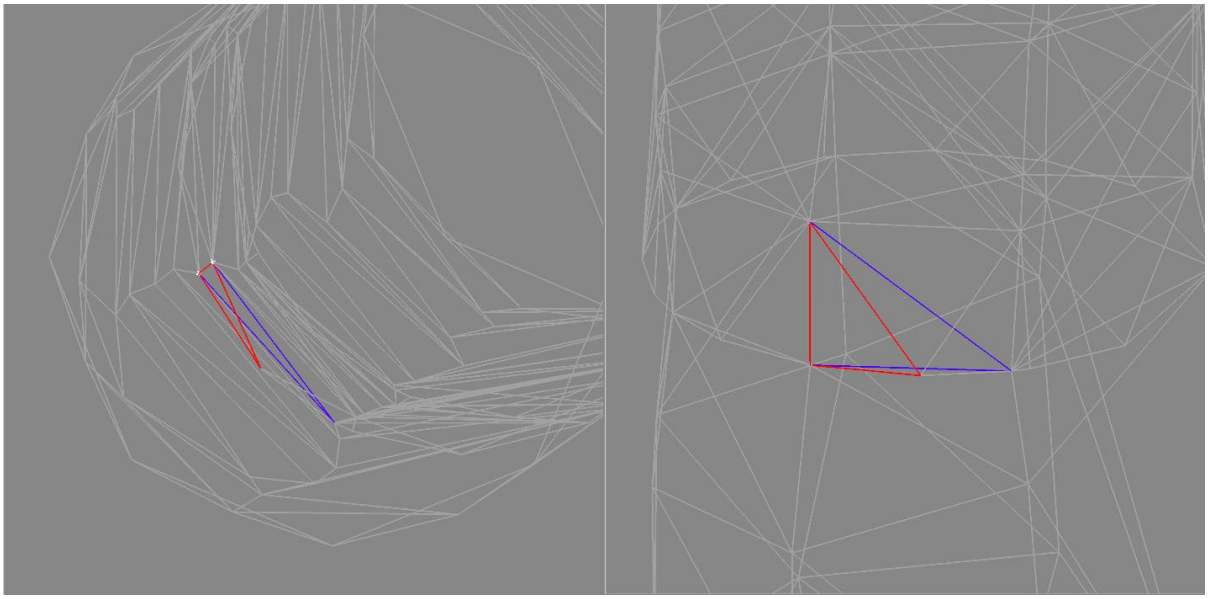

Figure 36. Edge-only triangle intersection pair (top view left and side view right) that is *not* part of a volumetric overlap

This situation is the most difficult of the triangle intersection cases to characterize. Currently a full-fledged point inside/outside polyhedron test based on Dickinson's (2020) implementation is

used for robustness. The test characterizes the triangle edge midpoints on the triangles' edges that are not the aligned edges from the triangle/triangle intersection test to determine if the triangle is or is not intruding into the other mesh. Moreover, because we do not want to report overlapping if the midpoints are on the surface of the mesh (to within tolerance), not only do we need the inside/outside test but we also need to find the closest point on the mesh (not the local face but the overall object mesh) and determine how close the midpoints are to being on the surface. This again uses the triangle RTree lookup, as well as the closest point on triangle routine from Geometric Tools (Eberly, 2020). In combination, these routines can be used (with considerable cost in time) to decide if a given triangle is inside or outside the opposite mesh.

If these routines determine that overlaps are still present in the meshes, the next step to attempt is to see if selecting different triangles can clear the overlaps. Even if vertices are all aligned, the choice of mesh edges alone can result in interfering triangles on nonplanar faces (Figure 34). As a preliminary step, the remaining intersections are analyzed and grouped into common sets: overlapping pairs are used as seeds to form the groups, and then grown until they cease increasing their triangle counts based on the following criteria:

1. All triangles in the group (starting with the seed pair) that intersect with at least one triangle in the group in the other mesh are added to the group.
2. All triangles overlapping with any triangle in the group in the projection plane are added to the group.
3. Steps 1 and 2 are performed repeatedly until no new triangles are added and all pairs are assigned to groups.

Visually this procedure's output manifests as "clusters" of related overlaps grouped in local areas on the mesh (Figure 37). Once groups are identified, there are a number resolution strategies to try:

1. For simple two-triangle four-vertex cases where all vertices can be mapped to close vertices on the opposite mesh, replace one set of triangles with a mapped version of the other set.
2. If the interaction is more complicated, locally re-triangulate the meshes in the area of the overlaps using a common projection plane fitted to the vertices from both meshes.
3. If Steps 1 and 2 cannot clear the mesh, identify mesh edges with nearby edges in the other mesh, find the closest point to the two edge segments (where such a point is not a segment end point) and introduce new mesh points near those intersections.
4. Repeat these steps iteratively, tracking improvements to the mesh until a stable state is reached.

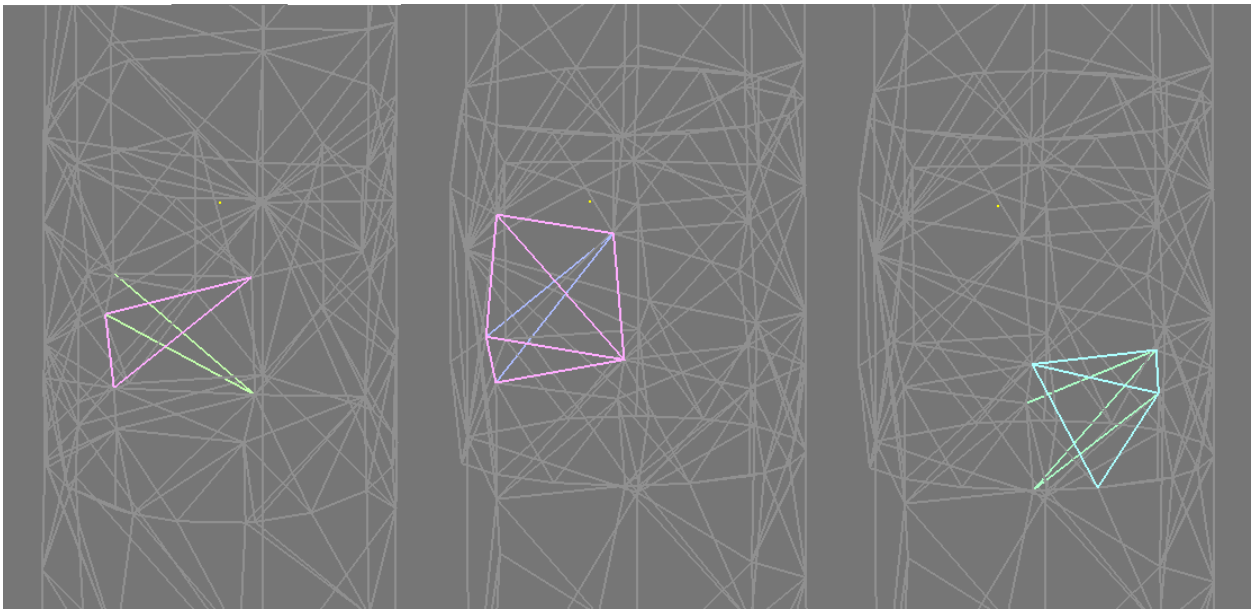

Figure 37. Group characterization of remaining overlapping triangles. These three sets of triangles may be processed locally to resolve overlaps.

Steps 1 and 2 have been implemented, and in the example case used up to this point, iterative application of those measures is sufficient to achieve a completely overlap-free meshing while keeping the mesh sparse (Figure 38). This is visually apparent when inspecting a shaded view of the output mesh and an rtcheck run in BRL-CAD along the cylinder's long axis (tangent to the interfering faces, which is the worst direction for overlaps) confirms a clean conversion (Figure 39).

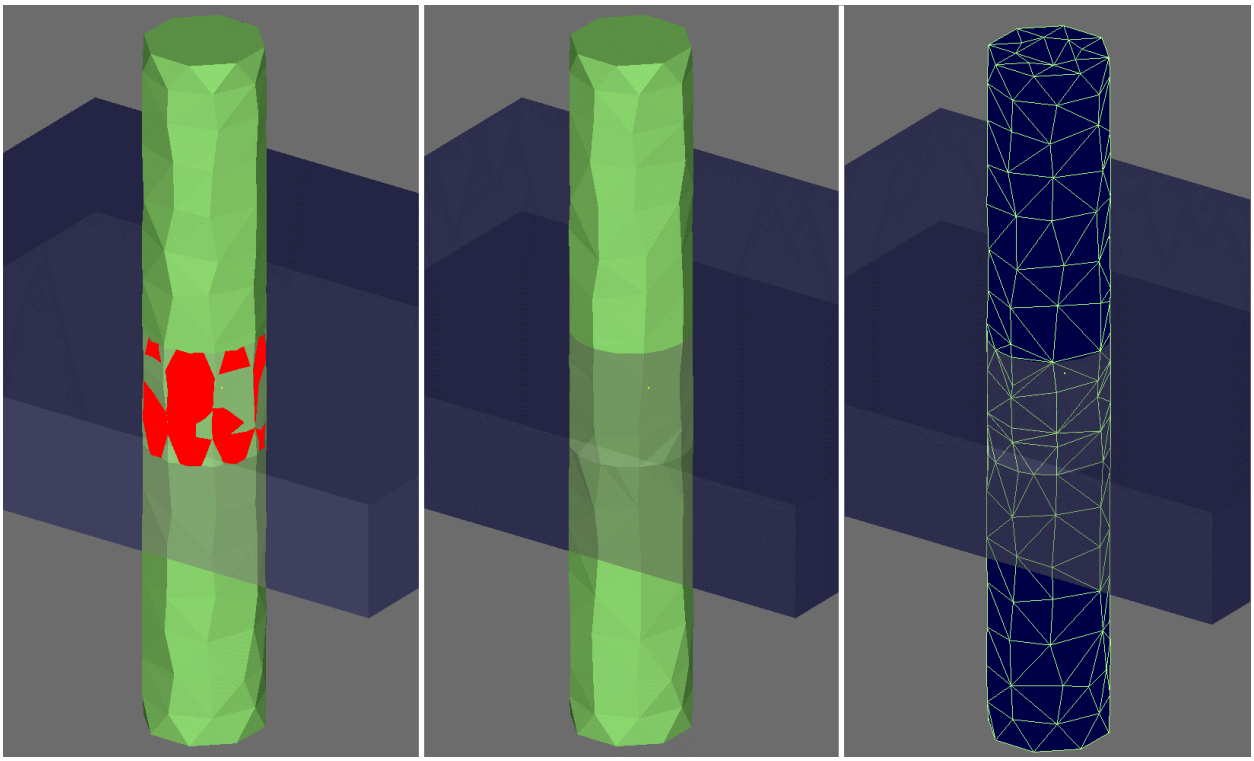

Figure 38. (left) Visualization of original overlaps, (middle) post-overlap resolution mesh, shaded view, and (right) wireframe. While the overlap volumes are gone, the mesh is much closer to its original mesh density. It was not necessary to introduce large numbers of new triangles.

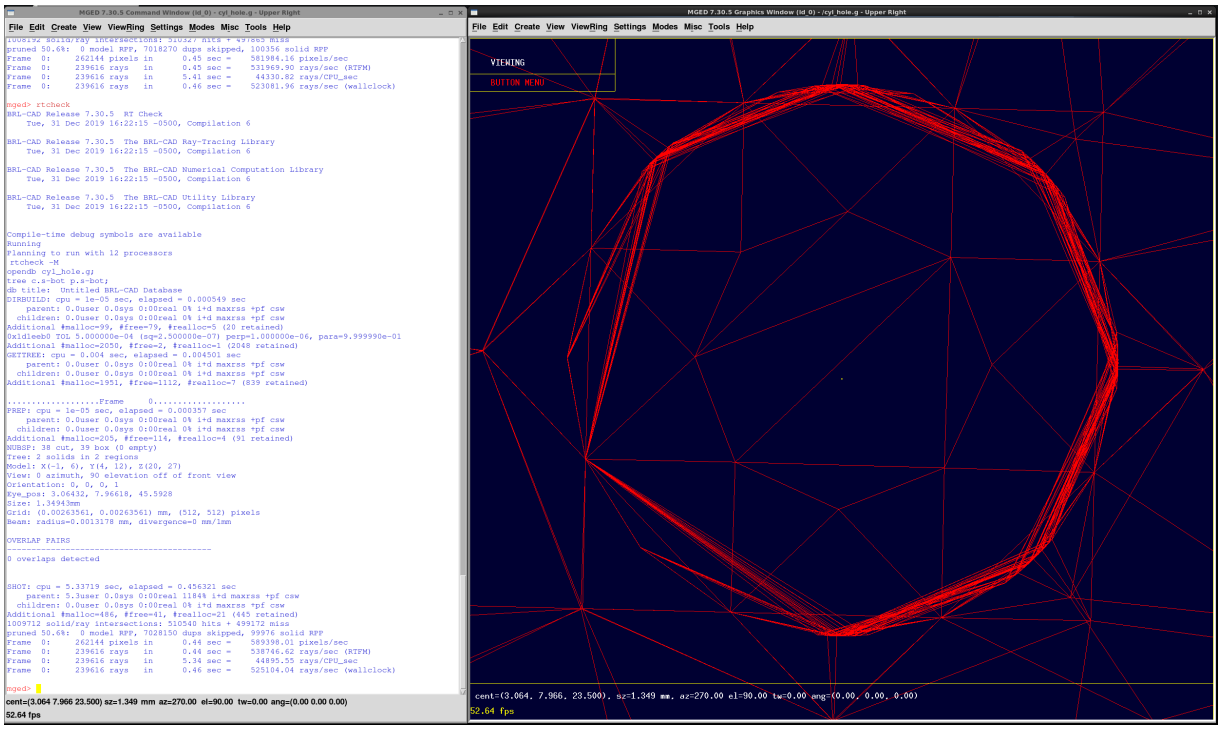

Figure 39. Successful clean (overlap-free) rtcheck test of the refined mesh in BRL-CAD's MGED tool, looking down the cylinder axis

However, while this result is promising, trials with other test cases demonstrate that these steps by themselves are not sufficient in all cases (Figure 40), and the implementation of Step 3 is not complete as of this writing. More work remains to achieve generally robust resolution results, and it is not yet known if additional measures beyond those already outlined will also prove necessary.

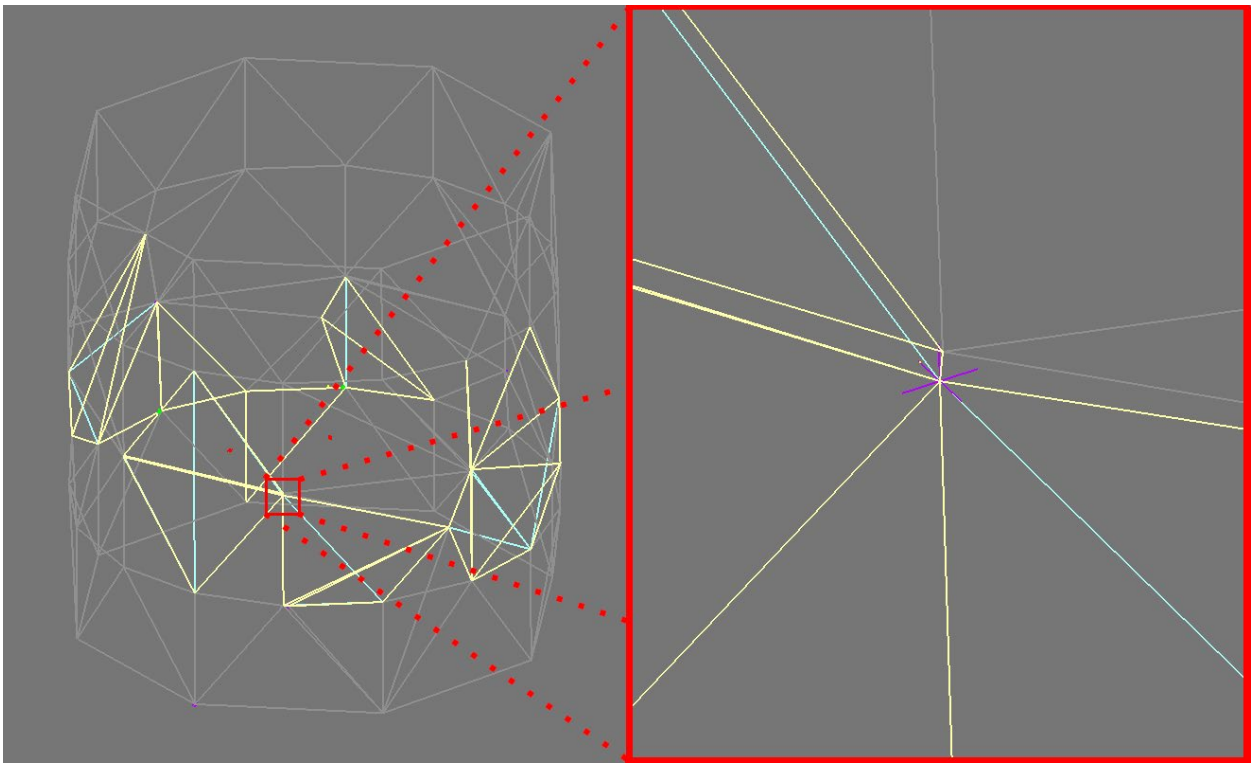

Figure 40. Overlaps from an example that cannot be fully resolved with currently implemented intruding vertex methodology

This approach to refinement also does not do as much as edge splitting to resolve gaps between meshes. Indeed, it is actually quite minimal in that respect, as coarse triangles that do not interfere with the opposite mesh do not tend to get "activated" for processing as smaller triangles constrict the meshes toward each other. It may be that a specific gap test (and refinement stage) is also in order to improve overall output quality from this method.

3. METHODS, ASSUMPTIONS, AND PROCEDURES – PLATE-MODE NURBS RAYTRACING

The basic plate-mode raytracing methodology was defined in FASTGEN (Aitken, Jones, & Dean, 1993), and BRL-CAD's implementation follows that approach. Plate mode uses an implicit thickness to create a solid shot-line segment using a single ray-intersection point. The plate-mode thickness is used to move the hit point along the interrogating ray back toward the origin point of the ray and forward past the hit point. In this fashion a line segment can be returned from a single hit, avoiding the need to explicitly define a closed volume to generate a solid response. There are two primary methods for calculating these start and end points: 1) shifting the hit point forward and back along the ray by half of the plate thickness, or 2) calculating a distance to shift along the ray that uses the cosine of the obliquity angle of the hit—how steep the incoming angle of the ray is relative to the tangent surface at the hit point—to reflect the fact that a shallow angle will result in longer intersection lengths through a given solid. These two methods are referred to as NOCOS and COS, respectively (Figures 41 and 42).

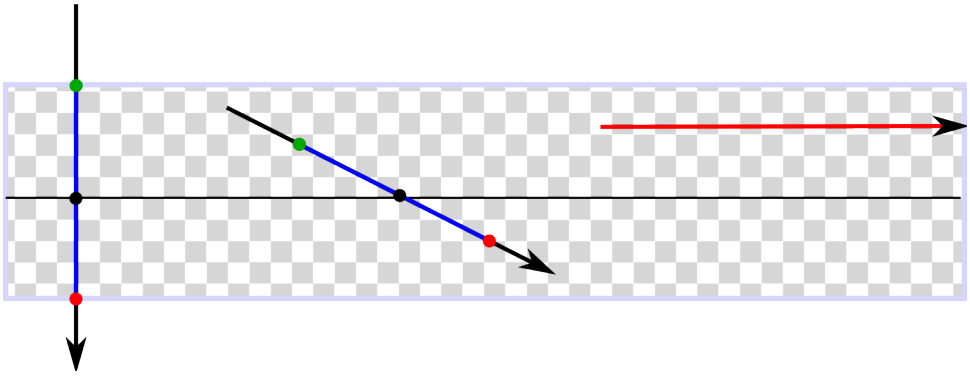

Figure 41. Cross-sectional view of three rays traversing near a NOCOS plate mode surface (black line) with an implicit thickness defined by the checkboard pattern. The leftmost ray, intersecting tangent to the surface, returns a thickness that matches the implicit shape. The middle ray, at an angle, reports a thickness, but that thickness is less than would be expected if the implicit solidity of the plate-mode surface were modeled explicitly. The right ray, with no surface intersection point to use, reports a miss even though it passes through the implicit volume.

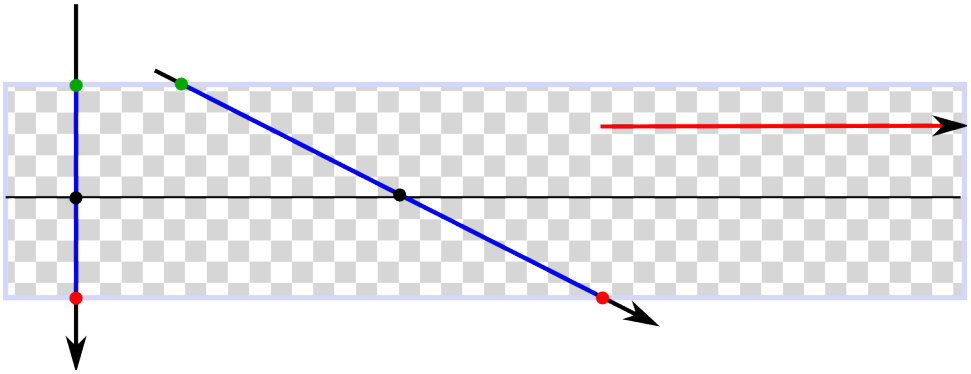

Figure 42. Cross-sectional view of three rays traversing near a COS plate mode surface (black line) with an implicit thickness defined by the checkboard pattern. The leftmost ray (intersecting tangent to the surface) and the middle ray (intersecting at an angle) report thicknesses consistent with the implicitly defined solid shape (an improvement over the NOCOS result, but requiring increased computation time per ray). The right ray, with no surface intersection point to use, still reports a miss even though it passes through the implicit volume.

Initial work on this problem in 2016 (Wu, 2016) demonstrated the basic feasibility of applying plate-mode methodology to NURBS surfaces (Figure 43).

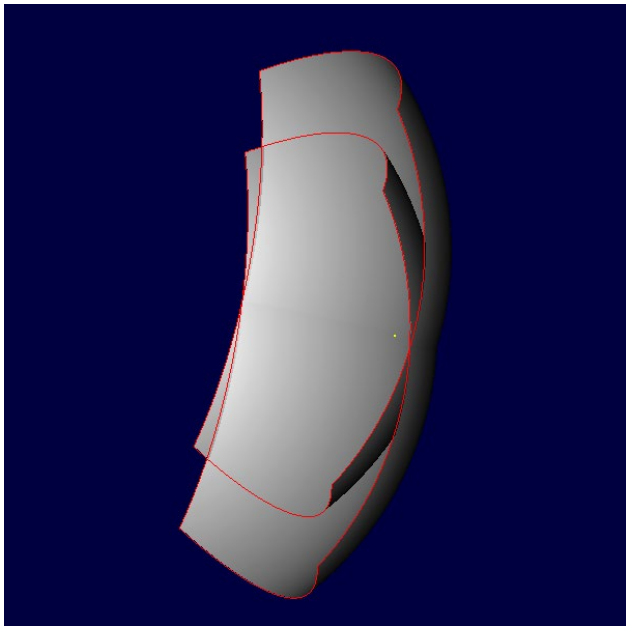

Figure 43. Example image from the Google Summer of Code plate-mode NURBS project, demonstrating surface-only renderings of objects in BRL-CAD (Wu, 2016)

However, an implementation challenge appeared when attempting to integrate this work into the primary BRL-CAD NURBS raytracing workflow. The simplest approach conceptually to adding this support is to apply the plate-mode behavior whenever the raytracer encounters unpaired hit points. Unfortunately, when this approach was attempted, it introduced artifacts in the output

generated for solid models (Figure 44). As of this writing, the reason for these artifacts has not been fully root caused. Fortunately, the nature of openNURBS BRep definitions allows for an alternative integration approach that more selectively applies the plate-mode logic, allowing solid NURBS-based objects to be raytraced using the existing code without modification.

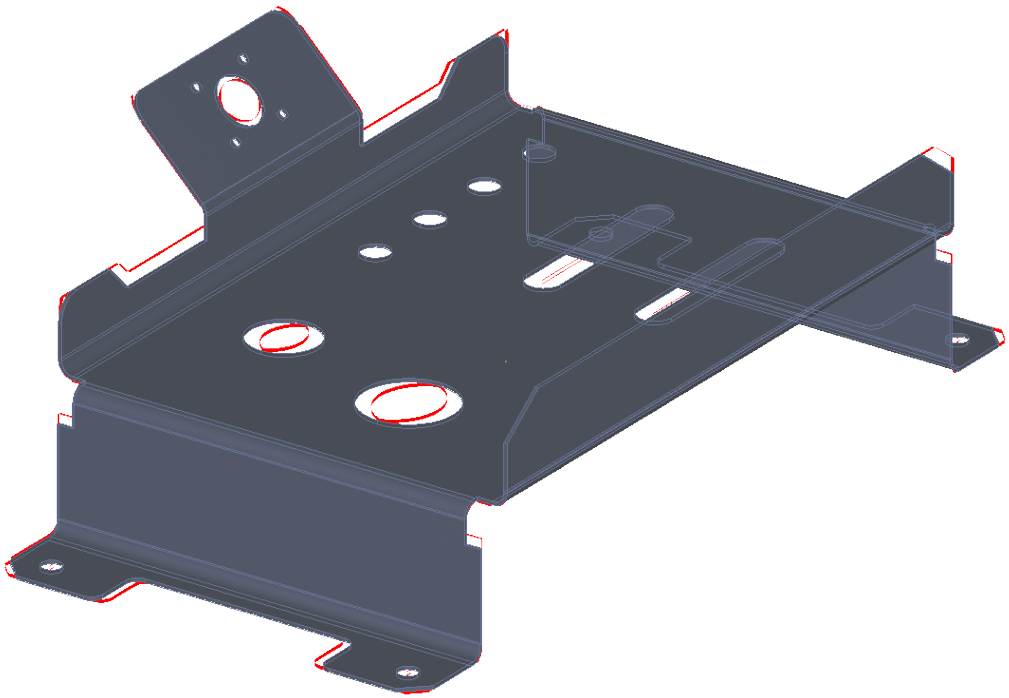

Figure 44. NIST3 problematic results with plate mode enabled. NIST3 is a solid object and thus should not change when raytracing with plate mode enabled. Red areas indicate unexpected hits returned by RT when plate mode is enabled.

OpenNURBS BRep objects track the type of trimming curves used to define objects, and the number of trimming curves associated with particular edges. For valid objects that are single surface definitions, they will exhibit one or more edges associated with a single trimming curve of the "boundary" type. When loading NURBS-based objects, those that are valid according to openNURBS and found to match this data signature are tagged as being "plate mode" NURBS objects. In this fashion, plate-mode logic can be applied only in cases where it is needed.

To validate the behavior of this code, example geometric shapes were defined by the target modeling team that expressed known simple primitive surfaces: a partial box, open cylinder, half sphere, and open cone (Figure 45). These shapes can also be expressed using traditional BRL-CAD implicit primitives, allowing for behavioral comparisons to ensure sane behavior of the plate-mode intersection method.

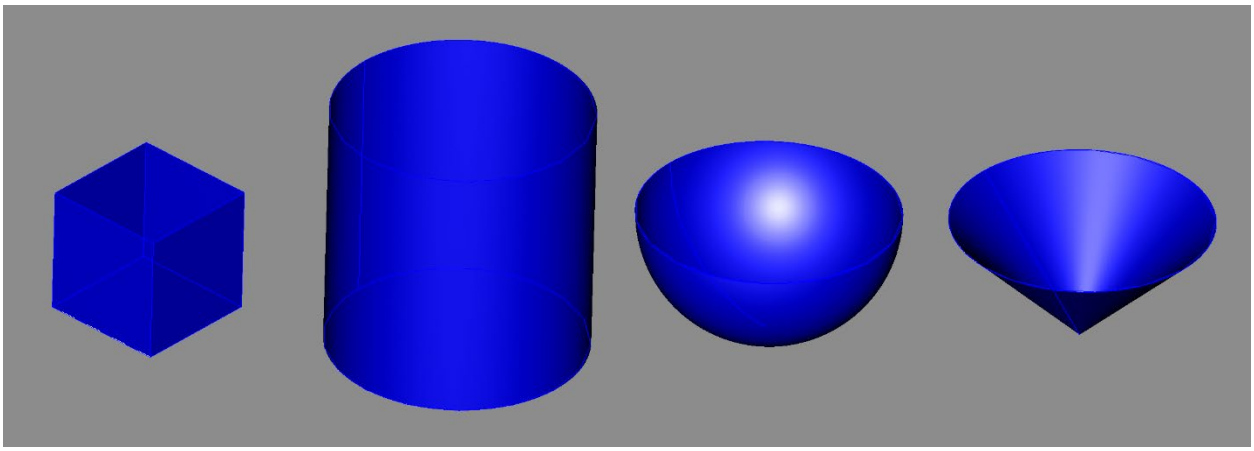

Figure 45. Plate-mode NURBS BRep example objects in BRL-CAD

By setting the plate-mode thickness to a large value, it is possible to visualize the behavioral difference of COS versus NOCOS intersection to confirm the logic is operating as expected (Figure 46). This visual confirms both the addition of implicit solidity at intersection points and the length differences between the two methods.

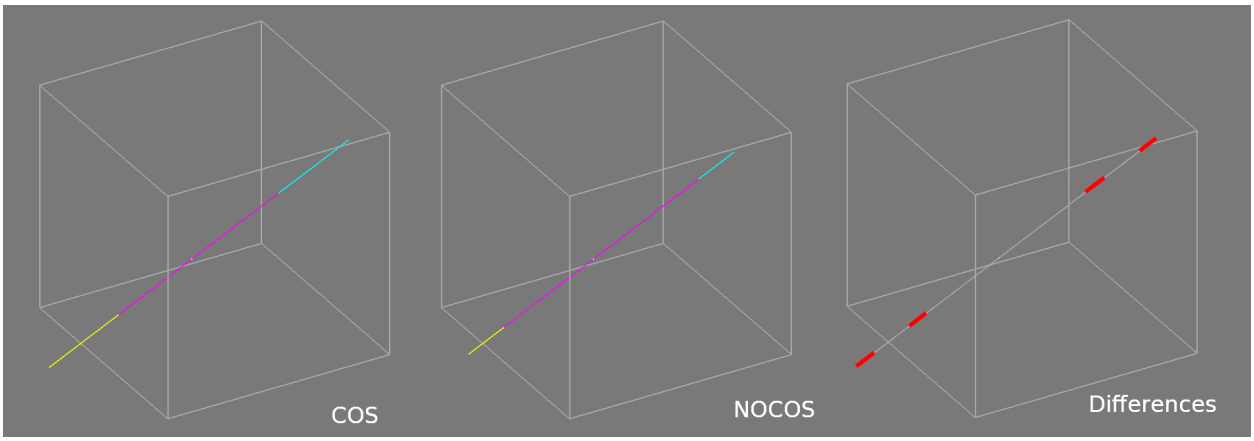

Figure 46. Partial-box plate-mode NURBS object in BRL-CAD showing intersection with a single Natalie's Interactive Ray-Tracer (NIRT) ray. Yellow and blue are solid segments, and purple denotes a gap. The figure on the right highlights in red the difference in reported segments, confirming that the COS segments are longer when the incoming rays intersect at a steep angle relative to the surface.

To confirm that the expected segment lengths are being defined at the expected places, the box, half-sphere, and cylinder objects were paired with CSG hierarchies defining what should theoretically be exactly the same shapes. These sets were then drawn simultaneously in MGED, and the NIRT command was used to determine if the solid lengths along the shot lines were in agreement. If they were, the expected result would be fully overlapping solids. Shots from multiple directions produced the expected fully overlapping results in this configuration, demonstrating that the CSG and plate-mode NURBS intersections agree (Figure 47).

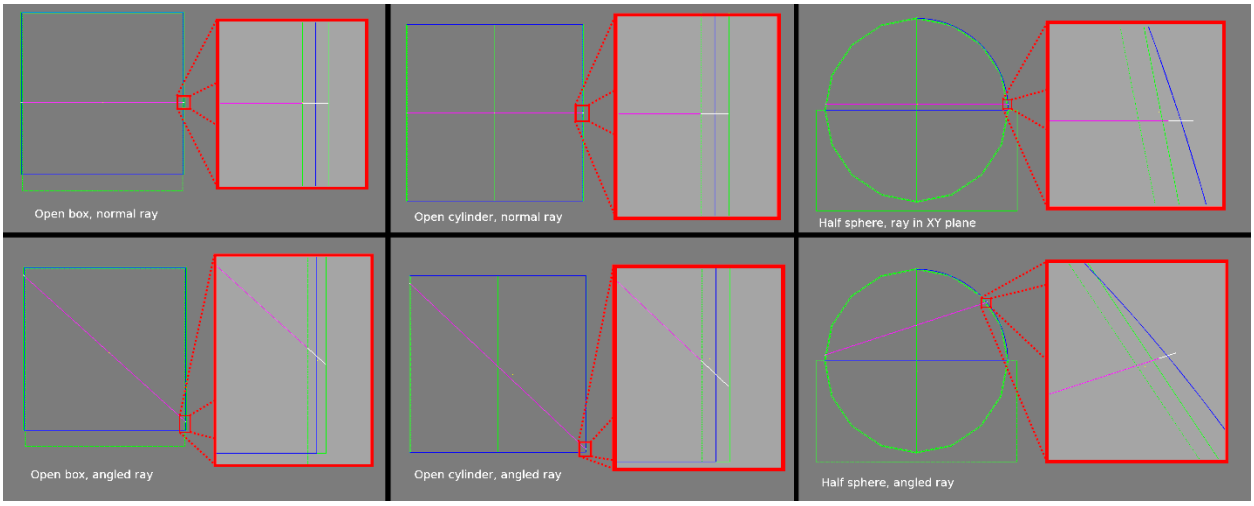

Figure 47. Interactions between deliberately overlapped CSG and plate-mode NURBS objects as tested with NIRT. For each of the six ray/object combinations, an imperfect match between the plate-mode answer would result in blue or yellow line segments near the white overlapping portion of the shot line. For all cases, the results are those expected for a correct plate-mode implementation. Only overlap and gap segments are observed.

An example of a valid NURBS geometry modeled using "plate mode" surfaces encountered in the wild is the teapot example included in the openNURBS Sample Models data set (McNeel, Inc., 2020). This model was previously not supported by BRL-CAD's raytracing implementation but can now be rendered successfully as a hierarchy of plate-mode NURBS BRep solids (Figure 48).

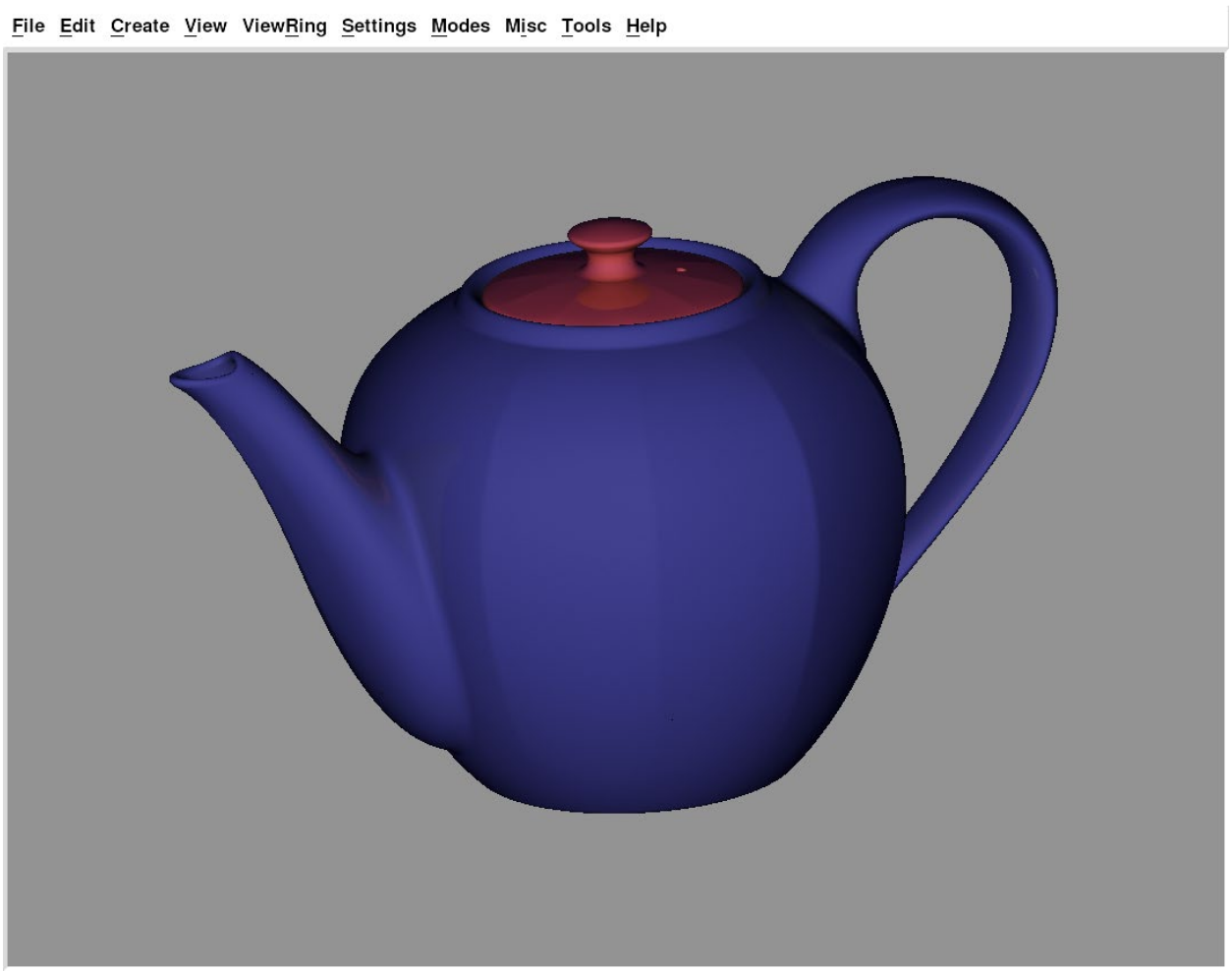

Figure 48. BRL-CAD raytraced rendering of the openNURBS Sample Model v5_teapot.3dm plate-mode geometry model

4. CONSTRAINTS, LIMITATIONS, AND ASSUMPTIONS

Both overlap resolving mesh conversion and plate-mode NURBS BRep raytracing have a number of limitations on the types of inputs and operations they support.

4.1 Limitations on Meshing Inputs

When generating solid, valid meshes from BReps, the following conditions must be met:

1. The original BRep object must be valid (i.e., define a closed solid volume and pass the openNURBS validity checks).
2. The original BRep object must have no unmated trim curves (curve connectivity is needed for defining closed solids).
3. During processing, repair zones cannot become so large that they fail to project to a plane without self-intersecting in that projection. (Such cases would require a more-sophisticated projection strategy, and currently no such strategy has been implemented.)
4. Tolerance settings must not be so fine as to produce triangle counts that are prohibitive in terms of memory or runtime.

There are also a number of restrictions on what this meshing approach can handle when it comes to refining overlapping triangles:

1. The original NURBS geometry is must be non-overlapping (i.e., any mesh overlaps present are a consequence of the triangulation process).
2. There are no Boolean operations other than unions in the geometry tree being processed.
3. There are no BRL-CAD primitive shape types other than brep and comb/region present in the tree being processed.
4. Tolerances for overlap resolution cannot be too small relative to the size of the objects being triangulated. Otherwise, both memory and runtime requirements will become impractical. Users are advised to specify coarse tolerances and gradually work down to finer tolerances based on experience with specific models.

Currently all processing is implemented serially, without use of multi-threading. At least some stages of this process are amenable to parallelization, which would offer significant speedups.

There is a tradeoff between mesh quality and processing speed: water tightness, validity, and overlap clearing involve successively more-expensive mesh operations. The original 2014 meshing implementation is fast enough for quick shaded visualization of NURBS solids. Currently none of the higher-quality options are fast enough to generate visualization meshes on the fly.

4.2 Limitations of Plate-Mode NURBS Inputs

The NURBS plate-mode methodology inherits the same weaknesses exhibited by mesh-based plate-mode raytracing: view dependency. A thickness setting that is thick compared with the surface area or large compared with the overall dimensions of interest to the analysis will produce a "volume" with solidity in some directions but not in others. (Grazing rays near the surface will "miss" the plate-mode volume that a tangent ray would see there, because there is no hit point to use to assign the implicit thickness.)

Plate-mode NURBS raytracing is currently limited in BRL-CAD to objects whose definitions indicate they are plate-mode objects. Unlike triangle meshes, there is currently no way to "force" a brep primitive to be interpreted as a plate mode object by the raytracer. A plate-mode NURBS object must be created with modeling intent and pass validity checks to be so regarded in BRL-CAD. Invalid solid NURBS models cannot currently be "forced" into plate mode within BRL-CAD.

5. RESULTS

The software results from these efforts are a series of new capabilities and command options in BRL-CAD exposing those capabilities.

5.1 Meshing Results

The basic pieces of the meshing algorithm as outlined are in place in BRL-CAD and producing geometry successfully. These new capabilities are available in BRL-CAD via the "facetize" command in MGED (Figure 49). Specifically, the "-B" option enables the new BRep specific processing routines and validates that the specified inputs are suitable. A "-t #" tolerance value also needs to be specified to control how far the meshing algorithms should refine any overlapping triangles before halting. The "--max-time" option to facetize can be used to cap how long the overlap resolution step runs. After each refinement pass, if the overall processing time for overlap resolution is greater than the specified maximum time the remaining iterations will be skipped. In that case the mesh will be generated at whatever resolution was reached when the last completed refinement step finished.

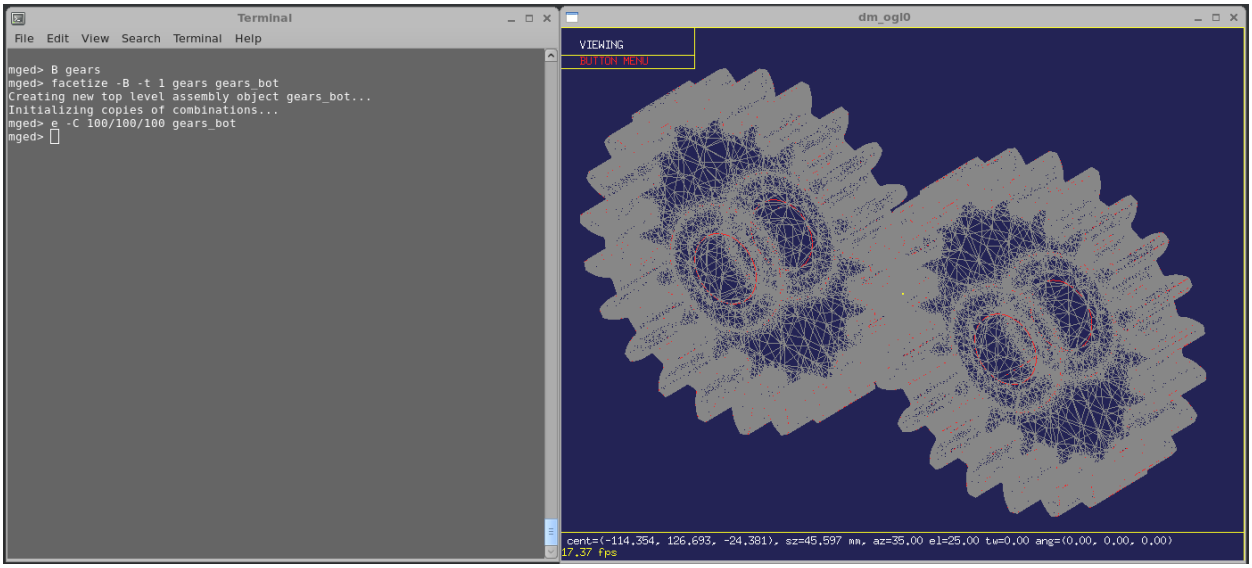

Figure 49. Set of gears (Elg, 2020) processed with the facetize command in MGED

Work remains to stress the implementation with larger inputs. Watertight meshing seems to be working well, and the repair logic responsible for clearing local flaws and singularity areas has demonstrated successful conversions with all valid NIST input example models.

The overlap refinement logic has demonstrated it can detect overlaps and locally refine meshes successfully to a specified tolerance. The runtime performance of this approach as currently implemented deteriorates with increasingly large triangle counts. It is not clear yet how much of

that increase is due to the inherent difficulty of the problem and how much might be alleviated with increased implementation efficiencies and optimization.

A more advanced approach to overlap refinement and removal that avoids introducing large numbers of additional triangles has been demonstrated in principle. For the moment, only the edge-splitting-based resolution technique is exposed in BRL-CAD. More work is required to improve the robustness of the intrusion-vertex-based refinement process; however, it is not known with any certainty how much additional work would be needed to define enough refinement heuristics to ensure reliable completion. In addition to the implementation difficulty of this approach, it has known performance implications. The necessity of using an inside/outside test [which is currently O(n) in performance where n is the number of triangles in the mesh being tested, due to robustness requirements] means that for large models, even relatively sparse triangulations may be difficult to resolve without runtimes too long even for non-interactive use. If runtimes on the order of days can be achieved for large models and robustness is improved, the process will still be useful for non-interactive conversions.

5.2 Plate-Mode NURBS BRep Results

Plate-mode NURBS raytracing has been successfully demonstrated in BRL-CAD, both from a visualization/rendering perspective and for generating the "solid shot lines" necessary for V/L analysis methodologies.

Modification of plate-mode brep object settings is accomplished using MGED's "brep" command (Figure 50). Detection of plate-mode NURBS BReps (as opposed to solid or invalid BReps) is currently automatic, but thickness and COS/NOCOS modes may be set by users. Thicknesses (which defaults to zero on import) may be specified with a "brep <objname> plate_mode #' command, where "#" is the desired thickness for the object in current database units. Switching between COS and NOCOS evaluation modes is also done with the brep command, specifying either "brep <objname> plate_mode cos" or "brep <objname> plate_mode nocos" to set the desired mode. The default if no explicit setting is made is "COS" mode.

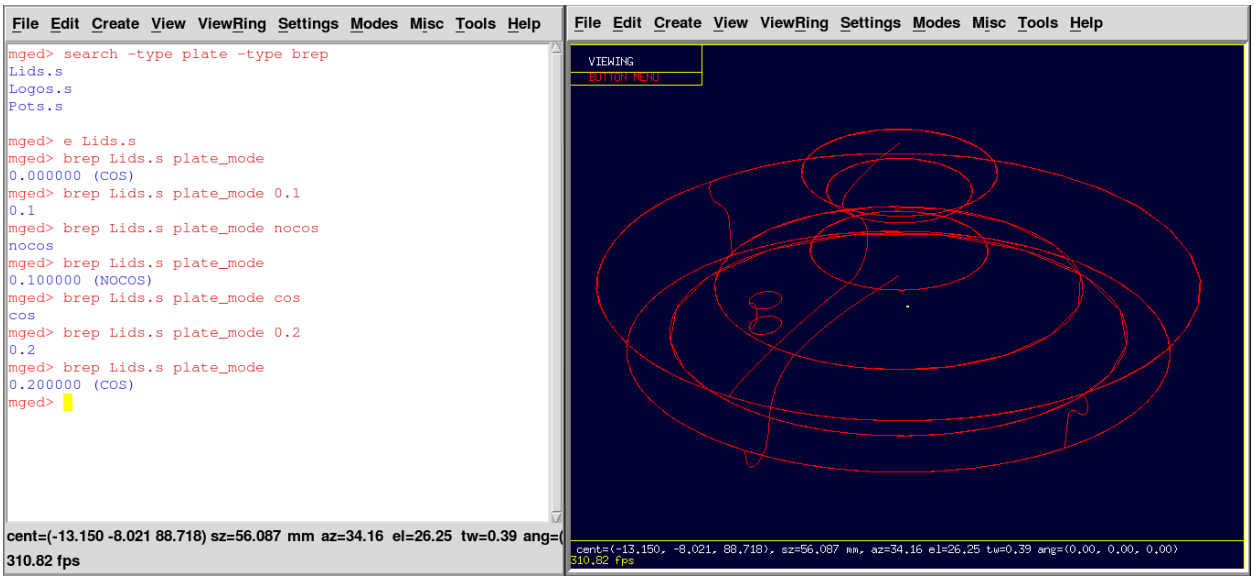

Figure 50. Using MGED's "search" and "brep" commands to find and modify a plate-mode brep object

The BRL-CAD "search" command has also been enhanced with the ability to list objects in a .g file that use plate-mode evaluation via the "-type plate" filter. To find all plate-mode NURBS BRep objects in a database, the user may run "search -type plate -type brep". Similarly, to find bot meshes that are plate mode they may run "search -type plate -type bot".

6. CONCLUSION AND RECOMMENDATIONS

Although the difficulty of implementation considerably exceeded original estimates, BRL-CAD now has a working meshing method for NURBS solids that can generate both watertight triangulations and locally refined meshing in overlapping BRep face regions. Continued refinement of the initial implementation for performance, code organization/clarity, and developer centric documentation is in order. Plate-mode NURBS methodology was also successfully integrated, with appropriate user-level tools for identification and modification of such objects.

The next step for the meshing logic is to rework the data containers and implementation of meshing algorithms with an eye toward improved efficiency and flexibility. Now that the basic workflows needed for successful completion of the meshing are clear, a number of structural improvements have been identified that will improve both performance and outcomes.

For plate-mode support, the next steps are to explore the behavior of BRL-CAD's tools with plate-mode solids and correct any limitations found. Investigation is needed to ensure the view-dependent nature of the solids does not produce unexpected behaviors. Work is also needed to identify the root cause of the raytracing artifacts observed during the first plate-mode integration attempt. That behavior should at least be understood, and if it represents some flaw in the solid raytracing implementation, it should be addressed.

7. REFERENCES AND DOCUMENTS

Aitken, E. D., Jones, S. L., & Dean, A. W. (1993). *A guide to FASTGEN target geometric modeling.* https://apps.dtic.mil/docs/citations/ADA273171

Bowman, W. K. (2014). *Implementation of NURBS triangulation using* Poly2Tri. https://sf.net/p/brlcad/code/62598/tree/brlcad/trunk/src/librt/primitives/brep/brep.cpp

Boyer, P. *Verb – a JavaScript library for creating and manipulating NURBS surfaces and curves.* Retrieved January 26, 2020 from https://github.com/pboyer/verb

Dickinson, M. *Polyhedron – a Python implementation of a robust algorithm for point-in-polyhedron testing*. Retrieved January 25, 2020 from https://github.com/mdickinson /polyhedron/

Domiter, V., & Zalik, B. (2008). *Sweep-line algorithm for constrained Delaunay triangulation*. International Journal of Geographic Information Science. 22(4):449–462.

Defense Systems Information Analysis Center. *Vulnerability toolkit.* Retrieved February 28, 2020 from https://www.dsiac.org/resources/models_and_tools/vulnerability-toolkit

Eberly, D. *Geometric tools – closest point on triangle test.* Retrieved January 25, 2020 from https://www.geometrictools.com/GTEngine/Include/Mathematics/GteDistPointTriangle.h

Elg, J. *9-cylinder radial engine.* Retrieved January 30, 2020 from https://grabcad.com/library/9-cylinder-radial-engine—12

Green, M., & poly2tri open-source software community. *Poly2Tri constrained Delaunay triangulation library 2012–2020*. Retrieved February 6, 2020 from https://github.com/jhasse /poly2tri

Guttman, A. (1984). *R-trees: a dynamic index structure for spatial searching*. Proceedings of the 1984 ACM Association for Computing Machinery's Special Interest Group on Management of Data International Conference on Management of Data (SIGMOD). p. 47.

McNeel, Inc. *OpenNURBS 6 sample models.* Retrieved January 29, 2020 from https://www.rhino3d.com/download/opennurbs/6/opennurbs6samples

Möller, T. (1997). *A fast triangle-triangle intersection test*. Journal of Graphics Tools. 2(2):25–30. 10.1080/10867651.1997.10487472

National Institute of Standards and Technology (NIST). (2014, December). *PMI test CAD model dataset 1: combined/complex test cases*. (2014, December). Retrieved January 25, 2020 from http://www.nist.gov/el/msid/infotest/mbe-pmi-validation.cfm

Robert McNeel and Associates. (2018). *The openNURBS initiative: 1993–2018.* Retrieved January 25, 2020 from http://www.rhino3d.com/opennurbs

Sederberg, T., Anderson D., & Goldman R. (1984). *Implicit representation of parametric curves and surfaces*. Computer Vision, Graphics and Image Processing. 28:72–84.

Wu, B. (2016). Google summer of code development blog. Retrieved January 27, 2020 from https://bojianwu.github.io/gsoc2016

Appendix A – List of Acronyms

2-D	two-dimensional
3-D	three-dimensional
bot	Bag of Triangles
BRep	Boundary Representation
brep	NURBS BRep object type in BRL-CAD database
BRL-CAD	Ballistic Research Laboratory–Computer-Aided Design
CAD	computer-aided design
CDT	Constrained Delaney Triangulation
CSG	constructive solid geometry
NIRT	Natalie's Interactive Ray-Tracer
NIST	National Institute of Standards and Technology
NURBS	Non-Uniform Rational B-Splines
PMI	Product and Manufacturing Information
V/L	vulnerability/lethality

Appendix B – Distribution List

ORGANIZATION

U.S. Army CCDC Data & Analysis Center
FCDD-DAD-OL/T Handlir
FCDD-DAS-LBS/C Yapp
6896 Mauchly St.
Aberdeen Proving Ground, MD 21005-5071

U.S. Army CCDC Army Research Laboratory
FCDD-RLD-CL/Tech Library
2800 Powder Mill Rd.
Adelphi, MD 20783

Defense Technical Information Center
ATTN: DTIC-O
8725 John J. Kingman Rd.
Fort Belvoir, VA 22060-6218