9-2004

Trim Loop Closure for Enhanced CAD Interoperability

Armand D. Assadi
UGS PLM Solutions

James H. Oliver
Iowa State University, oliver@iastate.edu

Trim Loop Closure for Enhanced CAD Interoperability

Abstract
The transfer of design data among different CAD systems or subsequent downstream analysis applications is critically important to the acceleration of the product development cycle. Since each vendor has its own proprietary native file format, this transfer of data among differing systems is difficult at best. International standards such as IGES and STEP have evolved to address this challenge, but they are generally not sufficiently explicit. Each vendor writes its own "flavor" of the standard that other applications may not understand. This paper bridges a gap between disparate systems by developing a strategy to assess the completeness and robustness of models represented in IGES or STEP format, and a technique to either repair the representation or add missing information so that a downstream application can properly interpret it. The method ensures that the receiving system gets a full and accurate NURBS-based representation: the original surfaces, the corresponding full complement of model space trim curves, and the corresponding full complement of parameter space trim curves. With all the information present, the downstream system is more likely to receive the information it requires to interpret the model.

Keywords
CAD Repair, interoperability, IGES, STEP, Virtual Reality Applications Center

Disciplines
Computer-Aided Engineering and Design | Graphics and Human Computer Interfaces

Comments
This is a conference proceeding from *ASME 2004 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 1 (2004): 1, doi:10.1115/DETC2004-57533. Posted with permission.

Proceedings of DETC'04:
ASME 2004 Design Engineering Technical Conferences and Computer and Information in Engineering Conference
September 28-October 2, 2004, Salt Lake City, Utah USA

DETC2004-57533

TRIM LOOP CLOSURE FOR ENHANCED CAD INTEROPERABILITY

Armand D. Assadi
UGS PLM Solutions
2321 North Loop Drive
Ames, Iowa 50010
USA
515-296-9106
armand.assadi@ugsplm.com

James H. Oliver
Virtual Reality Applications Center
2274 Howe Hall, Room 1620
Iowa State University
Ames, Iowa 50011-2274
USA
515-294-2649
oliver@iastate.edu

ABSTRACT

The transfer of design data among different CAD systems or subsequent downstream analysis applications is critically important to the acceleration of the product development cycle. Since each vendor has its own proprietary native file format, this transfer of data among differing systems is difficult at best. International standards such as IGES and STEP have evolved to address this challenge, but they are generally not sufficiently explicit. Each vendor writes its own "flavor" of the standard that other applications may not understand. This paper bridges a gap between disparate systems by developing a strategy to assess the completeness and robustness of models represented in IGES or STEP format, and a technique to either repair the representation or add missing information so that a downstream application can properly interpret it. The method ensures that the receiving system gets a full and accurate NURBS-based representation: the original surfaces, the corresponding full complement of model space trim curves, and the corresponding full complement of parameter space trim curves. With all the information present, the downstream system is more likely to receive the information it requires to interpret the model.

Keywords: CAD Repair, interoperability, IGES, STEP

INTRODUCTION

In today's Computer-Aided Design (CAD), Computer-Aided Manufacturing (CAM), and Computer-Aided Engineering (CAE) world, the transfer of geometric data among disparate CAD systems and subsequent downstream engineering applications, analysis tools, and visualization software is critically important to the acceleration of the development cycle. However, the success of this process is predicated on the creation and transfer of accurate and complete geometric models [1,2]. Frequently, data files do not transfer between systems effectively [2]. This leads to increased costs in the product development process.

A common solution for the data transfer problem is the use of neutral standard file formats, such as the Initial Graphics Exchange Specification (IGES) and, the Standard for the Exchange of Product model data (STEP). However, these standards are often subject to individual interpretation leading to ambiguities and inconsistencies. Also, time-to-market pressures led many vendors to lobby for broad definitions so they could meet the general intent of the specification with minimal effort and investment. As a result, translators created by different vendors vary greatly in terms of how they implement the standards. Thus, the files still do not transfer as effectively as would be desired [2]. There are other sources of problems occurring with IGES or STEP including different specification and interpretation of geometric tolerances, modeling practice variations, degeneration of information, corruption of files, differences in modeling standards used within companies, manual data entry, and the generally inverse relationship between model complexity and translator reliability.

An entire software market has evolved to address these deficiencies. Commercial solutions such as ITI's CADfix [3], XOX Corporation's GDX [4], Avatech Solutions' DesignQA and GeometryQA [5], TransMagic, Inc.'s translation software products [6], Translation Technologies Inc.'s Acc-u-Trans and Mirror Model Comparator [7], Theorem's CADhealer [8] and UGS PLM Solution's Parasolid Bodyshop [9] vary in scope and approach. However, since they are proprietary, they are generally not described in the open literature.

By far, the most common source of errors encountered in transferring contemporary surface and solid models are surface representation errors caused by incomplete and or misinterpreted trim loops. Despite their prevalence and importance, neither IGES nor STEP imposes a rigorous specification for trimmed surfaces. For example, trim curves may be defined in

the coordinate system of the parent surface (model space) or within the 2D parametric domain (parameter space) of the parent surface. Successive curves forming a trim loop need not be topologically ordered, and need not be closed. In some CAD systems, the boundaries of the parametric domain are implicitly assumed to be trim curves if not otherwise defined; while in others, loops must be explicitly defined. Some systems require trim curves to be defined only in model space, while others require both model space and parameter space trim curves.

Researchers have addressed these challenges of modifying the topological definition of the model in various ways. Some have proposed interfacing directly with the native CAD package itself [10,11] while others deal with the neutral file formats [12-15]. Of these, some are semi-automatic [12,13] requiring user interaction for unsolvable instances, while others are automatic [14,15]. However, none of these automatic methods are completely accurate due to the difficult nature of the problem [16].

This work introduces an automatic methodology towards making the neutral file formats IGES and STEP work better for geometric data transfer problems. Analyzing the available geometric information within these files and repairing or adding missing information allows for a much improved rate of successful transfers between different systems. The contribution of this work is a loop closure algorithm that assesses the quality of an IGES or STEP trimmed surface definition and repairs and augments the data so that a receiving system is more likely to be successful in properly interpreting it.

LOOP CLOSURE ALGORITHM

To assist with understanding the loop closure algorithm, the following terminology and assumptions are used:

- A trim curve is an individual parameter space or model space curve.
- A composite trim curve is a collection of trim curves linked together.
- A trim loop is a trim curve or composite trim curve that closes on itself.
- A parent surface is the base surface on which trim curves and loops are defined.

The primary assumption made by the loop closure algorithm is that the parent surface NURBS definition is complete and accurate. If there is no underlying surface to deal with, the algorithm cannot work. Also, its parameter and model space curves are not assumed to be ordered or oriented. They are initially treated by the loop closure algorithm as a general collection of curves.

One final point of interest, although the loop closure algorithm accepts parameter and model space curves, is that all analysis is done within parameter space. This is done for the simplicity of working in a bounded 2D workspace rather than an unbounded 3D workspace. For this reason, if the input is limited to only model space curves, they must be inversely mapped to parameter space.

The motivation of this work, as stated earlier, is to get the standard data exchange formats IGES and STEP to transfer the needed geometric information to another system correctly. To that end, there needs to be an assumption that the IGES and STEP files are valid according to their specification since this work is not a "how-to" for exporting valid geometric data into a valid file structure. What this work focuses on is the current standard for the representation of surface and solid models; trimmed NURBS surfaces. Specifically, it focuses on correcting trim curve errors caused by number precision accuracy or by the different geometric representation types of CAD systems.

To solve any trim curve errors, there needs to be a validity and consistency check of the geometric information. The validity check ensures that the trim curves are properly defined and that a trim loop is created. The consistency check ensures that a full trimmed NURBS surface definition is provided, meaning there exists a surface, a full complement of model space curves, and a full complement of parameter space curves.

Validity ensures that the trimmed NURBS definitions provided by the IGES or STEP file are valid and complete. The validity checks are that an underlying NURBS surface is provided and that any provided models space and parameter space curves are valid. After checking each individual definition, the trim curves are checked to see if they form a valid trim loop. If not, closed trim loops need to be formed.

For consistency, there are four possible combinations of a trimmed NURBS surface definition that an authoring system may create: 1) definition of the surface only, 2) definitions for the surface and its corresponding model space curves, 3) definitions for the surface and its corresponding parameter space curves, or 4) definitions for the surface and its corresponding model space and parameter space curves. A target system will also require that the imported data be in one of these four combinations. However, if there is an information type mismatch, the target system will not be able to recreate the geometric information. For example, an authoring system provides surface and model space curve definitions, but a target system requires surface and parameter space curve definitions. Therefore, the consistency check, given a valid underlying NURBS surface, maps any of the missing model space or parameter space curves.

To provide for a robust representation of models, the loop closure algorithm is comprised of several specific functions to achieve this goal. These functions include creating, deleting, and splitting curves; normalizing curve and surface definitions; truncating or extending curves to form an endpoint intersection; mapping or inverse mapping of parameter and model space curves; forming composite trim curves from trim curves; joining disjoint trim curves and composite trim curves; and orienting trim loops and their corresponding trim curves in the proper trimming direction.

The loop closure algorithm is described in three parts: the pre-processor, the processor, and the post-processor. The pre-processor takes care of any initial inverse mapping that is needed and normalizes all NURBS definitions. The processor is the heart of the loop closure algorithm where robust trimmed surface definitions are created. The post-processor orients the trim loops properly before the repaired surface is passed on.

Surface representation errors are usually caused by incorrect trim curves. Trim curve errors commonly occur with surfaces that contain poles, seams, or degenerate edges. For trim loops to be correct they must meet the following criteria: be continuous, yield a bounded active surface, lie completely within the parametric space, and not intersect themselves.

Surface relationship errors may be self-intersecting surfaces, small gaps between surfaces, or overlaps between surfaces. Catastrophic self-intersecting surfaces are rarely seen.

Small gaps or overlaps between surfaces may be corrected through correction of the surface's trim curves. If not, the surfaces need to be mended [17-22].

LOOP CLOSURE METHODOLOGY

This section provides the details of the three main parts of the loop closure algorithm: the pre-processor, the processor, and the post-processor. Figure 1 depicts the general process flow of the loop closure algorithm.

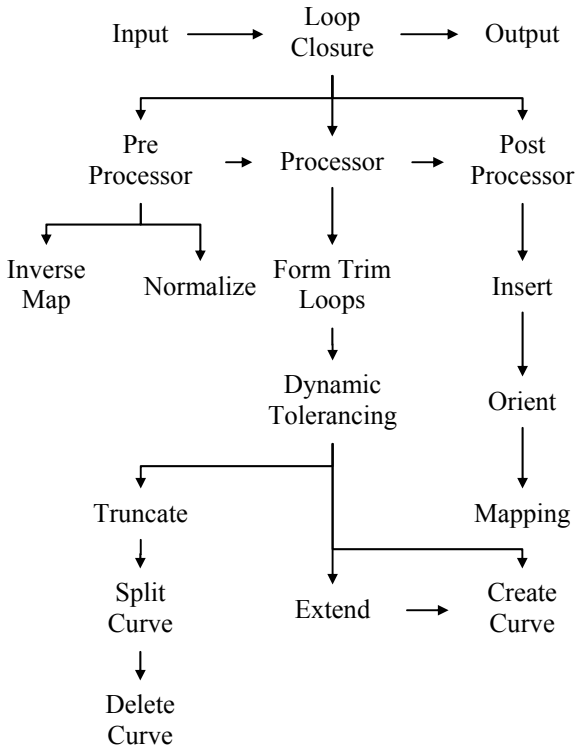


Figure 1. The loop closure structure.

Pre-Processor

The pre-processor is the first part of the loop closure algorithm. Its two steps involve getting the geometric data definitions into a form that the loop closure algorithm uses by making sure all trim curves are defined in parameter space and normalizing all NURBS definitions. Making sure everything is defined in parameter space is the first step. If there are only model space curves defined, the corresponding parameter space curves are created using the Inverse Mapping function. The next step is to make sure that the surface and all curves are normalized to a parametric range of [0,1] using the Normalize function. After these two steps, the NURBS definitions of the surface and curves are in their default state. With the proper form for the definitions, the data is ready to proceed to the main processor.

Inverse Mapping

Given a NURBS model space curve $\mathbf{C}_{\mathrm{m}}(t) = \{x(t), y(t), z(t)\}$ that lies in the vicinity of a NURBS surface $\mathbf{S}(u,v)$, inverse mapping refers to finding its image $\mathbf{C}_{\mathrm{p}}(t) = \{u(t), v(t)\}$ in parameter space [23,24]. Iteratively marching along $\mathbf{C}_{\mathrm{m}}(t)$ by $\Delta t = 1/(n-1)$ computes surface data points $\mathbf{S}_i(u(t_i), v(t_i))$, where $i = 1 \ldots n$ and n is the number if data points that produces an accurate representation of the model space curve. The simplest method for solving the inverse mapping problem is to solve the point projection problem using Newton iteration to minimize the distance between the model space curve point $\mathbf{S}_i(u(t_i), v(t_i))$ and the surface point $\mathbf{S}(u,v)$ for some (u,v). Therefore the offset vector function

$$\mathbf{r}(u,v) = \mathbf{S}(u,v) - \mathbf{S}_i(u(t_i), v(t_i))$$

and two angular difference scalar equations

$$g(u,v) = \mathbf{r}(u,v) \cdot \mathbf{S}_v(u,v) = 0$$

$$f(u,v) = \mathbf{r}(u,v) \cdot \mathbf{S}_u(u,v) = 0$$

are solved seeking point coincidence and zero cosine convergence. Figure 2 illustrates the mapping from model space to parameter space.

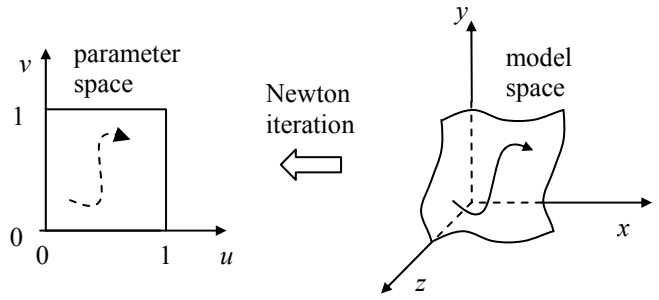


Figure 2. Inverse mapping from model space to parameter space

Normalize

To ensure that all curve and surface definitions are defined on a consistent knot span, the normalize function normalizes the parametric range of curves and surfaces to [0,1]. Given model or parameter space curves, this function normalizes the knot vector, and for parameter space curves and surfaces, normalizes their associated control points accordingly.

Processor

The processor, the heart of the loop closure algorithm, computes, modifies, and completes trim loops. The functions involved with the processor are Form Trim Loops, Truncate, Expand, Create Curve, Delete Curve, Split Curve, Mapping, and Nearest Neighbor.

The Form Trim Loops function automatically assembles successive trim curves together to form composite trim curves and, if possible (and preferable), trim loops.

Form Trim Loops

The loop closure automatically assembles successive trim curves together to form composite trim curves until a trim loop has been formed. As many composite trim curves as possible are initially constructed whether they form trim loops or not. From the collection of trim curves for each face, one is selected to analyze. There are three possibilities for this trim curve: it forms a trim loop by self-enclosing; it can be joined to a composite trim curve (and possibly completing a trim loop); or it does neither, and a new composite trim curve must be started. For the trim curve to form a closed loop, the leading and trailing endpoints of the curve must have the same coordinate value. For a trim curve to be appended to a composite curve, either the leading or trailing endpoint of the trim curve must be the same coordinate value as the leading or trailing edge of the composite trim curve. When the trim curve does not form a trim loop or adjoin to a composite trim curve, it is a stand-alone curve at this point forming its own composite trim curve. After this trim curve has been analyzed, it may be removed from the inputted list of trim curves. Linked lists are used as the mechanism to hold the trim curve information. Pseudo code for this operation is as follows:

Form Trim Loops (curves)
if *the number of* **curves** = *0* ***then***
    return *NULL*
while *the number of* **curves** ≠ *0* ***do***
    select a trim curve $\mathbf{C_T}$ *from the* **curves**
    if $\mathbf{C_T}$ *is a closed curve* ***then***
        create a new trim loop $\mathbf{C_L}$
    else if $\mathbf{C_T}$ *is appendable to a composite trim curve* $\mathbf{C_{CT}}$ ***then***
        append $\mathbf{C_T}$ *to* $\mathbf{C_{CT}}$
    else
        create a new composite trim curve $\mathbf{C_{CT}}$
    remove $\mathbf{C_T}$ *from* **curves**
for *each composite trim curve* $\mathbf{C_{CT1}}$ ***do***
    for *each composite trim curve* $\mathbf{C_{CT2}} \neq \mathbf{C_{CT1}}$ ***do***
        if $\mathbf{C_{CT2}}$ *is appendable to* $\mathbf{C_{CT1}}$ ***then***
            append $\mathbf{C_{CT2}}$ *to* $\mathbf{C_{CT1}}$
            remove $\mathbf{C_{CT2}}$
return *the composite trim curves and trim loops*

After the composite trim curves and trim loops have been constructed, the Nearest Neighbor function is used to determine which disjoint trim curves and/or composite trim curves need to be connected to form potential trim loops. The trim loops do not need to be evaluated since they already form a closed loop.

Nearest Neighbor

To create trim loops, trim curves (and composite trim curves) must be linked together to form closed loops. The approach of connecting endpoints that are closer than a predetermined constant tolerance ε does not always lead to accurate results. A tolerance that is too small may not link some trim curves while a tolerance that is too large may link the wrong trim curves.

Figure 3 shows a situation where a tolerance that is too small will not join all the appropriate trim curves. When curves *a* and *b* are compared, the distance d_1 between their closest endpoints is greater than the tolerance value ε_1 and are thus appropriately not connected. When curves *b* and *c* are compared, the distance between their closest endpoints d_3 is less than the tolerance value and they are appropriately joined. However, when curves *a* and *c* are compared, the distance d_2 between their closest endpoints is greater than the tolerance value and they are incorrectly not joined.

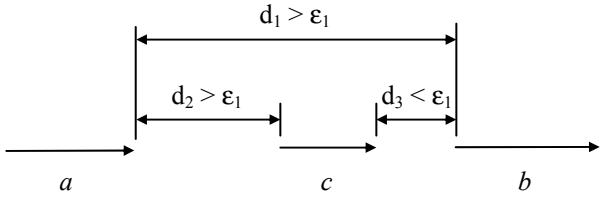


Figure 3. Joining of curves with too small a tolerance value.

Figure 4 shows a situation when the tolerance value is too large and joins the wrong trim curves together. When curves *a* and *b* are compared, the distance d_4 between their closest endpoints is less than the tolerance value ε_2 and are incorrectly joined together omitting the position where curve *c* should be placed. Therefore, different tolerances are needed for different areas of parameter space.

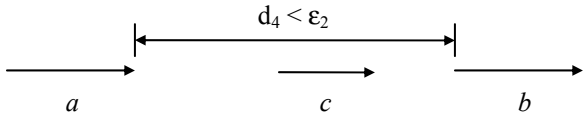
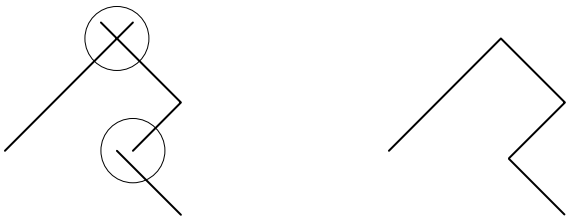


Figure 4. Joining of curves with too large a tolerance value.

Many tolerance-based approaches update the local tolerance associated with each face, edge and vertex according to previous operations [25,26]. The nearest neighbor approach used here is based on the entity linking and endpoints clustering method developed by Shpitalni and Lipson [27,28]. This approach computes the minimum distance from every trim curve endpoint to every other trim curve endpoint, excluding itself. If two endpoints are the same shortest distance from each other, they are joined together. With this method, the endpoint wants an answer to the question, "You are my closest point. Am I your closest point?" If the answer is yes, it is likely that these two endpoints need to be joined. Formally, this is stated as:

for *each endpoint* P_i ***do***
    for *each endpoint* $P_j \neq P_i$ ***do***
        compute the distance d_{ij} *between* P_i *and* P_j
        get shortest distance(s) d_{ijmin} *and corresponding endpoint(s)* P_{jmin}
for *each endpoint* $P_k \neq P_{jmin}$ ***do***
    compute the distance d_{jk} *between* P_{jmin} *and* P_k
    get shortest distance(s) d_{jkmin} *and corresponding endpoint(s)* P_{kmin}
if $d_{ijmin} = d_{jkmin}$, $d_{ijmin} <$ **tol**, *and* $P_i \neq P_{kmin}$ ***then***
    join P_i *and* P_{kmin}

This is repeated until no more trim curves are joined together. It is important to keep in mind that the nearest neighbor function is used only to solve for slight gaps between trim curves. It is not intended to solve the joining of two trim curves with a vast distance between them. Therefore, the distance is limited by some small upper bound. Figure 5 shows trim curves joined together derived using this method.



(a) Disjoint trim curves (b) Joined trim curves

Figure 5. Grouping endpoints by nearest neighbor.

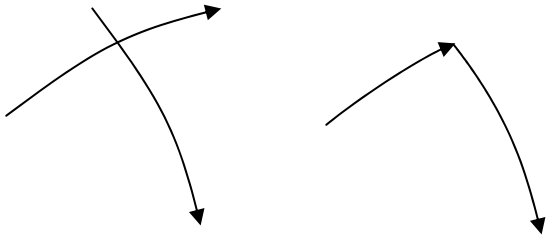
The next step is to automatically detect the reason why a trim curve or composite trim curve is invalid and correct the problem. There are three possible reasons why trim curves are not closed: a small gap exists between endpoints of trim curves, an open section exists where a trim curve needs to be created, or a catastrophic unsolvable error exists. Of these three errors, the first two are correctable by the loop closure algorithm. The unsolvable errors are caused by the authoring CAD system and can only be corrected from within.

The gap problem has two solutions, either finding an intersection of the trim curves or joining the trim curves with a linear segment if no intersection exists. If the trim curves already have an intersection point that is not their endpoint, the truncate function is used to remove the overhang.

Truncate

The truncate function finds either the (generally unlikely) self-intersection of one trim curve, or the intersection of two different trim curves, and deletes the overhanging portion of the curve. In general, given two trim curves $\mathbf{C}_1(s)$ and $\mathbf{C}_2(t)$, minimizing the function $f(s,t) = \mathbf{C}_1(s) - \mathbf{C}_2(t) = 0$ by Newton iteration solves the intersection point. Each curve exhibiting an intersection is then split at that intersection point using the split curve function. The curve that is significantly shorter by some predefined criteria than the other curve is deleted, as shown in Fig. 6. This function takes as input either one or two trim curves, and returns either one or two truncated curves. Pseudo code for this function is as follows:

Truncate (curve(s))
if there is no curve(s) or point ***then***
return *NULL*
find the intersection point using Newton iteration
split the curve at the intersection point by calling
Split Curve (curve, point)
delete the shorter curve by calling
Delete Curve (shorter curve)
repeat for the other intersection
return *curve(s)*



(a) (b)

Figure 6. Truncating trim curves.

If the trim curves do not already have an intersection, the Extend function is used to see if a valid one exists.

Split Curve

The split curve function splits a NURBS curve by inserting knots at the desired parametric split location and creating the new control points so that the split curves coincide with the original un-split curve. This function takes as input the parameter space curve definition that is to be split and the parametric value of where the split should occur and returns the two new NURBS curves. Pseudo code for this function is as follows:

Split Curve (curve, value)
if *there is no curve or value* ***then***
return *NULL*
split the given curve into two curves
return *the two curves*

Extend

The extend function seeks an intersection point of parameter space curve(s) based on two endpoints and the tangents of those endpoints. In general, given two trim curves $\mathbf{C}_1(s)$ and $\mathbf{C}_2(t)$, the intersection of $\mathbf{C}_1(1) + a\mathbf{C}_1'(1)$ and $\mathbf{C}_2(0) + b\mathbf{C}_2'(0)$, where $\mathbf{C}_1'(s)$ and $\mathbf{C}_2'(t)$ are the tangents to the curves, finds the point of extension.

If the intersection occurs within parameter space, new non-degenerate linear NURBS curves are created between the endpoints and the intersection point, as shown in Fig. 7. New segments are created instead of modifying the existing curve(s) so that their original curve(s) definition is not altered in any way. If no intersection is found or if the intersection occurs outside of surface parameter space, the two endpoints are directly connected by a new linear NURBS segment. This function takes as input the two endpoints and their respective tangents and returns new curve(s) connecting those endpoints. Pseudo code for this function is as follows:

Extend (endpoint1, endpoint2, tangent1, tangent2)
extend line from **endpoint1** *along* **tangent1**
extend line from **endpoint2** *along* **tangent2**
compute the **intersection point** *of the two lines*
if *intersection exists and is within parameter space* ***then***
create a new curve by calling
Create Curve (endpoint1, intersection point)
create a new curve by calling
Create Curve (endpoint2, intersection point)

else
create a new curve by calling
Create Curve (endpoint1, endpoint2)
return *curve(s)*

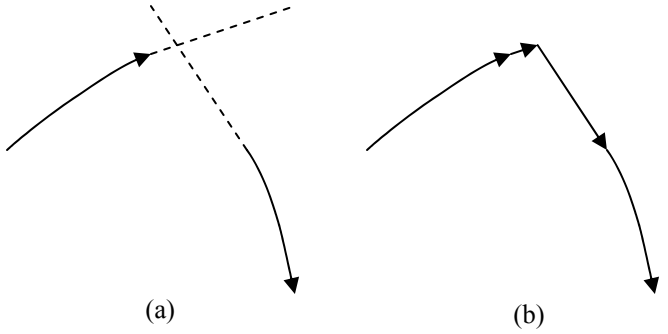


Figure 7. Extending trim curves.

If a valid intersection cannot be computed, then the Create Curve function is used to create a line segment between the two trim curve endpoints.

Create Curve

The Create Curve function creates a linear parameter space NURBS curve between two given points. These points are the endpoints of the curve as well as its control points. Since a parameter space curve is being created, it must lie completely within the bounded region of parameter space. This is done with a simple check on the two endpoints. This function takes as input the two points and returns a linear curve. Pseudo code for this function is as follows:

Create Curve (point1, point2)
if *point1 or point2 is not bounded parametrically* ***then***
return *NULL*
create a curve between point1 and point2
return *the curve*

Four cases exist that encompass the problem areas associated with trim curve error resolution: no trim curves, one (composite) trim curve, two (composite) trim curves, and more than two (composite) trim curves.

No Trim Curves

The first trivial case is when there are no parameter space trim curves available. This also means there are no model space trim curves either in which to produce the parameter space curves. With this case, it is assumed that the entire surface is needed. Therefore, a trim loop is defined that encloses all of parameter space. Four individual trim curves are defined on each of the parameter space boundaries (Fig. 8) using the Create Curve function.

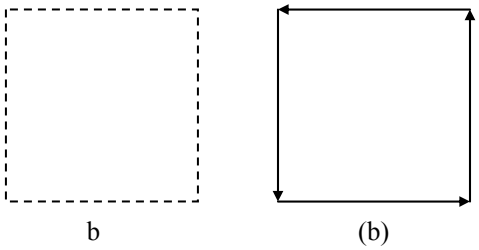


Figure 8. (a) No trim loops and (b) complete boundary trim loop.

One Trim Curve

The second case of trim curve error resolution is when there is only one trim curve. There are three positions in parameter space that this curve can occupy: the endpoints are in the interior (Fig. 9 (a)), one endpoint is on the boundary and the other in the interior (Fig. 9 (b)), or both endpoints are on the boundary (Fig. 9 (c)).

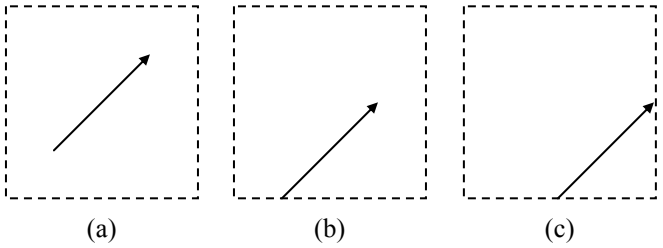


Figure 9. Spatial positions of one trim curve in parameter space.

For the first two conditions, there are two situations to address, that the trim curve may or may not be a linear segment as shown in Fig. 10 (a)-(d). If the trim curve is linear, there is not enough information to create a trim loop causing an unsolvable error. If the trim curve is not linear, then the Truncate, Extend, or Create Curve functions are used to connect the endpoints as, long as an intersection, other than at an endpoint, is not created.

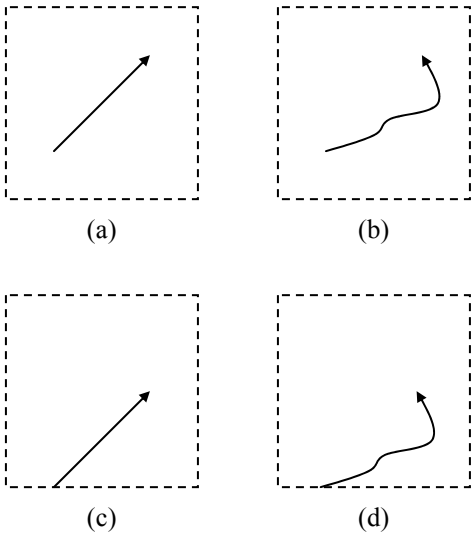


Figure 10. Linear and non-linear interior and semi-interior trim curves.

When both endpoints are on the boundary, traversing around the boundary in a counter-clockwise direction from the head to the tail of the trim curve defines the trim loop (Fig. 11). An area of concern is that the boundary curves that are created cannot intersect the trim curve at more than point contact intersections. If this condition is detected, the loop cannot be resolved and a fatal error is reported.

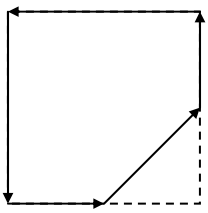


Figure 11. Completed boundary trim loop.

Two Trim Curves

The third case is when there are two trim curves. The Dynamic Tolerancing function determines if these trim curves should be joined. If so, the Truncate, Extend, or Create Curve function is used to connect them. If not, each one is evaluated independently using the one trim curve case.

More Than Two Trim Curves

The fourth case is when there are more than two trim curves. Again the Dynamic Tolerancing function determines which trim curves should be joined. For each pair of joinable trim curves, the two trim curve case is recursively used until the one trim curve case can be used for completion or until failure.

Post-Processor

The final step in processing a face is to ensure that each trim loop is properly oriented, counter-clockwise for regions or clockwise for holes. This is the final step to ensure that a robust definition of a trimmed surface exists.

The first step is to ensure that the individual trim curves within a trim loop are consistently oriented, as shown in Fig. 12. This should have already been done by passing through the forming trim loops phase, which connects the curves head to tail, but one last check is done for robustness.

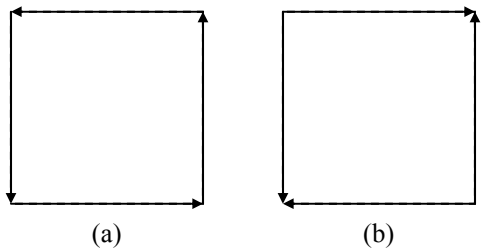


Figure 12. (a) Proper vs. (b) improper orientation of trim curves within a trim loop.

Orient

The orient function makes sure that all the trim loops are pointing in the appropriate direction. This is simple when using the hierarchy tree of trim loops. The trim loop represented by the root node of the hierarchy tree is always in counter-clockwise direction. Each subsequent level down the hierarchy tree is a trim loop going in the opposite direction. Therefore the trim loops alternate going from counter-clockwise to clockwise in successive levels of the tree. All trim curves on the same level within the tree are oriented in the same direction. The input to this function is the trim loops. Pseudo code for this function is as follows:

```
Orient (trim loops C_TL)
    create an trim loop list C_TLL
    for each trim loop C_TLi in C_TL do
        Insert (C_TLi, C_TLL)
    for each level in the hierarchy tree C_TLL do
        if level is odd then
            orient trim loop counter-clockwise
        else
            orient trim loop clockwise
```

After all the trim loops are consistently oriented, the second step is to build the containment-based hierarchy using the list of trim loops and the Insert function. After the hierarchy is created, orienting the trim curves becomes trivial. The root of the hierarchy tree is the outer most trim loop and must, by definition, be oriented in a counter-clockwise direction. Each subsequent progression down the tree orients the trim loop in the opposite direction, alternating between clockwise and counter-clockwise directions.

Insert

The Insert function, borrowed from [29], builds a containment-based hierarchy of trim loops, as shown in Fig. 13. Each node in the hierarchy is a list of trim loops and each trim loop may refer to another list of trim loops that are contained with it.

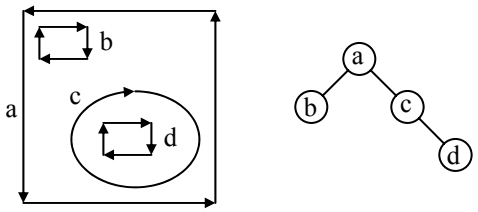


Figure 13. Trim curve loops and the resulting hierarchy.

Since the trim curve loops are not allowed to intersect, there are only three possible relationships between any two trim curve loops $\mathbf{C_1}$ and $\mathbf{C_2}$: $\mathbf{C_1}$ contains $\mathbf{C_2}$, $\mathbf{C_2}$ contains $\mathbf{C_1}$, or neither $\mathbf{C_1}$ nor $\mathbf{C_2}$ is contained within the other, as shown in Fig. 14 (a)-(c).

For $\mathbf{C_1}$ to contain $\mathbf{C_2}$, a ray from any point on $\mathbf{C_2}$ must intersect $\mathbf{C_1}$ an odd number of times (Fig. 14 (a)) as long as the intersection point(s) is not a tangent. For $\mathbf{C_2}$ to contain $\mathbf{C_1}$, a ray from any point on $\mathbf{C_1}$ must intersect $\mathbf{C_2}$ an odd number of

times (Fig. 14 (b)) as long as the intersection point(s) is not a tangent. For neither $\mathbf{C_1}$ nor $\mathbf{C_2}$ to contain each other, a ray from either curve must intersect the other curve an even number of times (Fig. 14 (c)) as long as the intersection point(s) is not a tangent.

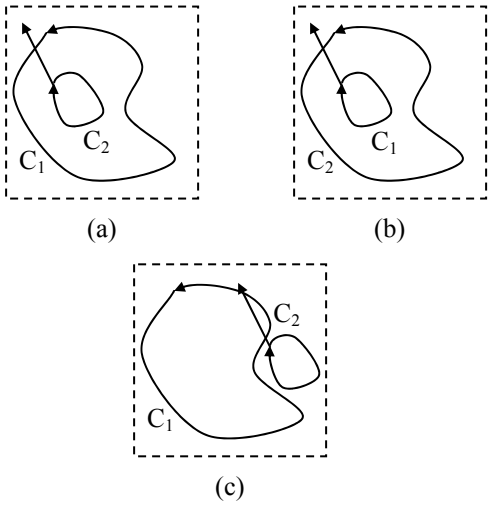


Figure 14. (a) C_1 contains C_2, (b) C_2 contains C_1, and (c) C_1 and C_2 are disjoint.

If tangents are found, a new ray is needed. This function takes as input a trim loop to be placed within the hierarchy and the hierarchy built so far. Pseudo code for this function is as follows:

Insert (trim loop C_{TL}, trim loop list C_{TLL})
for *each trim loop* C_{TLi} *in* C_{TLL} ***do***
if C_{TLi} *contains* C_{TL} ***then***
Insert (C_{TL}**,** C_{TLi} *trim loop list***)**
return
else if C_{TL} *contains* C_{TLi} ***then***
Insert (C_{TLi}**,** C_{TL} *trim loop list***)**
remove C_{TLi} *from the* C_{TLL} *trim loop list*
add C_{TL} *to the* C_{TLL} *trim loop list*

Once full definition of the parameter space trim curves is attained, they are mapped to model space, if necessary (i.e., if corresponding model space trim curves do not exist).

Mapping

Mapping from parameter space to model space solves the dependent (x, y, z) values given the independent (u, v) values for a curve lying on a parametric surface. Given a NURBS parameter space curve $\mathbf{C}_{\mathrm{p}}(t) = \{u(t), v(t)\}$ and its parent NURBS surface $\mathbf{S}(u, v)$, the equivalent NURBS model space curve $\mathbf{C}_{\mathrm{m}}(t) = \{x(t), y(t), z(t)\}$ is computed by interpolating surface data points $\mathbf{S}_i(u(t_i), v(t_i))$, where $i = 1 \ldots n$ and n is the number of points such that the model space curve meets a predetermined criteria of closeness to the surface. The $(u(t_i), v(t_i))$ are solved by incrementally moving along the parameter space curve from beginning to end by $\Delta t = 1/(n-1)$, assuming a parametric range of [0,1] for t. The surface data points $\mathbf{S}_i(u(t_i), v(t_i))$ are computed using these parameter values with the surface definition. Figure 15 illustrates the mapping from parameter to model space.

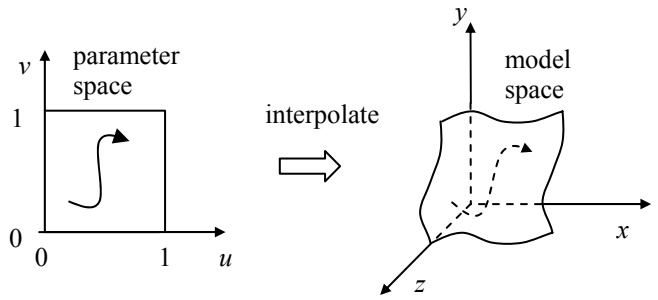


Figure 15. Mapping parameter space to model space.

EXAMPLES

The loop closure methodology was implemented using International TechneGroup Incorporated's (ITI's) PDElib [30] to parse the IGES and STEP files and filter the geometric data. The algorithm was written in C++ and operates on various platforms.

This chapter presents examples of IGES and STEP files created using a variety of commercial CAD systems. These files contain a variety of errors that are representative of the current state of the art. In most cases, the neutral file can be read successfully by the system that created it, since the same assumptions used in writing the data are generally employed in reading and interpreting it. However, most of the examples exhibited fundamental problems when read by a system other than the authoring system. Typical errors encountered include: improperly addressing poles, seams, and degenerate edges and gaps between trim curves.

In each example, the model is first shown using a visualization tool that requires complete closed loops (in both parameter space and model space) in order to robustly tessellate and display the models. After the loop closure algorithm is applied to the models, they are shown with the same visualization system.

While these types of errors are demonstrated with a visualization application, they are representative of the types of problems encountered by many downstream analysis systems.

Pole Errors

Figure 16 shows five primitives: a torus, a sphere, a cone, a cylinder, and a cube. The IGES data file was created using I-DEAS Master Series 5 and saved using its IGES exporter. This collection of primitives contains 19 faces, of which four need correction using the loop closure algorithm. The torus, cylinder, and cube provide complete surface, model space trim curve, and parameter space trim curve information, but the sphere and cone do not. Both of these primitives have the same problem; they have poles that correspond to trim curves and need to be defined in parameter space and then mapped to their corresponding singular point in model space. Since the sphere, cone, and cylinder were defined as two halves and the torus as

four sections, potential seam problems were avoided. The cube, defined as six planar faces, has neither a pole nor seam problem.

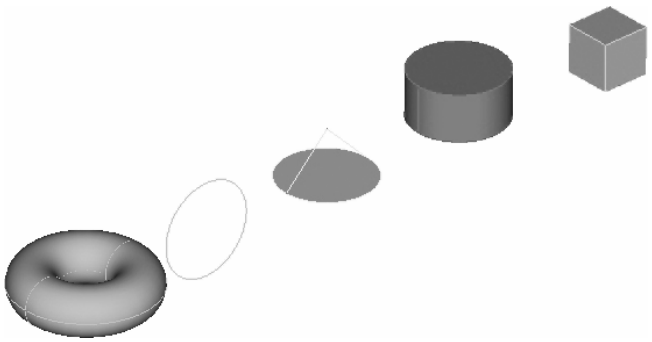


Figure 16. Primitives with pole errors.

Figure 17 (a) and (b) show the incomplete parameter space definitions for the pole problem of the cone and Fig. 17 (c) and (d) show it for the sphere. The open segments in parameter space indicate the location of the poles. The cone composite trim curve consists of three connected parameter space curves (*a*, *b*, and *c*). The sphere trim curves consist of two unconnected parameter space curves (*a* and *b*). In model space, the curves appear connected (Fig. 18 (a)-(d)).

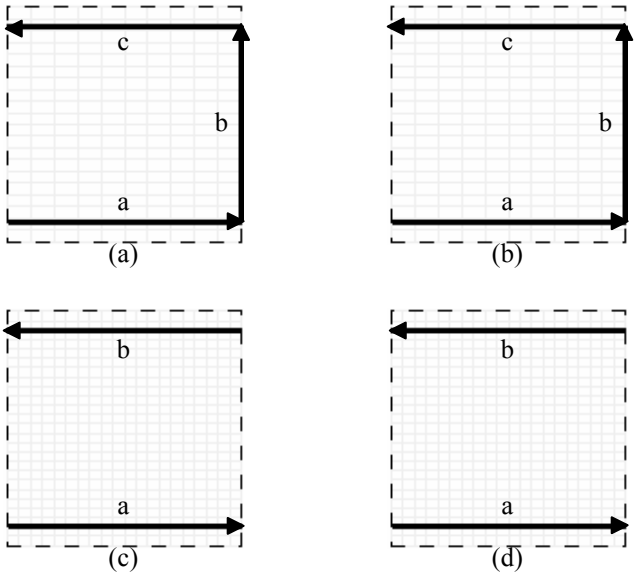


Figure 17. Parameter space trim curves exhibiting a pole error.

Figure 19 (a)-(d) and Fig. 20 (a)-(d) show the complete parameter space and model space definitions for the conical and spherical surfaces after having been completed by the loop closure algorithm. Each trim loop now consists of four connected parameter space curves (*a*, *b*, *c*, and *d*) that trim the underlying parent surfaces to provide the necessary half surfaces of the cone and sphere that are needed.

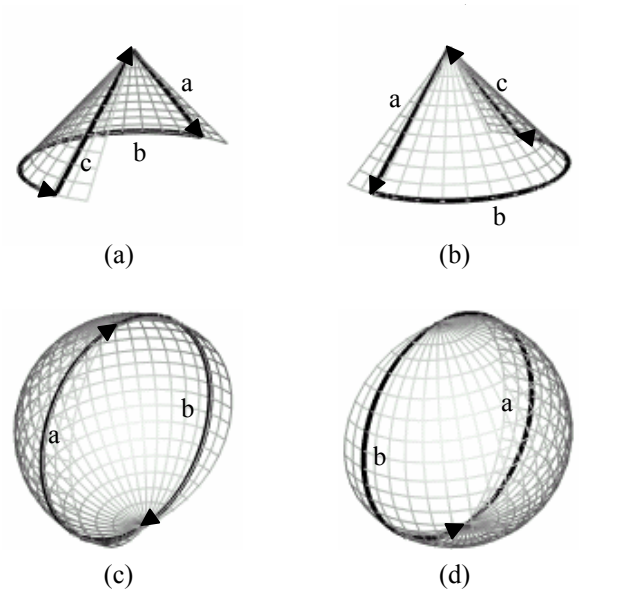


Figure 18. Model space trim curves exhibiting a pole error.

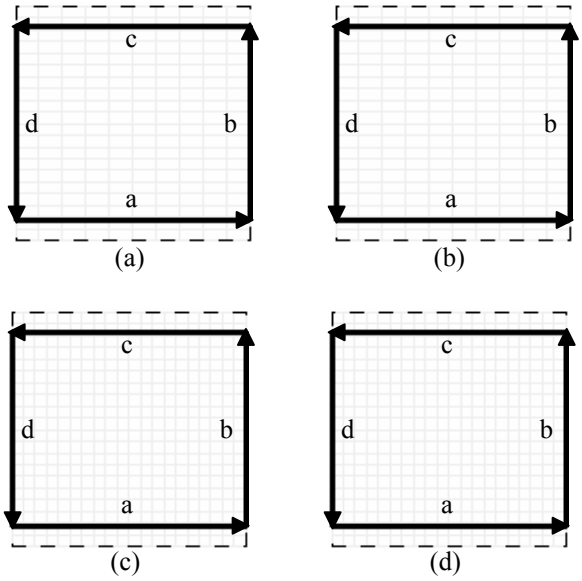


Figure 19. Parameter space trim curves without a pole error.

Figure 21 shows the same five primitives after having passed through the loop closure algorithm. The torus, cylinder, and cube needed no correction and were bypassed by the correction routine, but the cone and sphere were captured and corrected.

Gap Errors

Figure 22 shows a simple motor housing unit created using CATIA. This STEP file contains 23 faces with three needing repair. Two of these faces contain gaps between parameter space curves, most likely caused by a machine precision problem, or they may have been slightly ill defined.

Figure 23 (a) shows the parameter space curves for a face containing a gap. The blow-up is of the gap area (Fig. 23 (b)). Figure 24 (a)-(b) shows the corresponding model space curves. The composite trim curve for this surface consists of seven curves (*a*, *b*, *c*, *d*, *e*, *f*, and *g*).

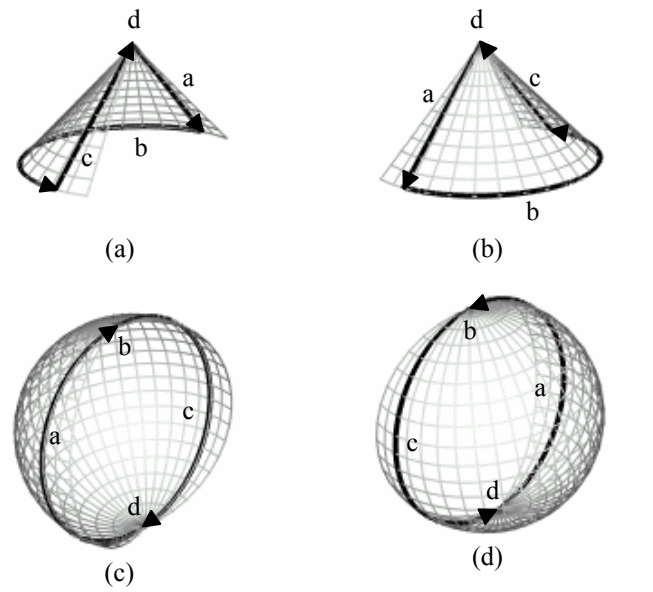


Figure 20. Model space trim curves without a pole error.

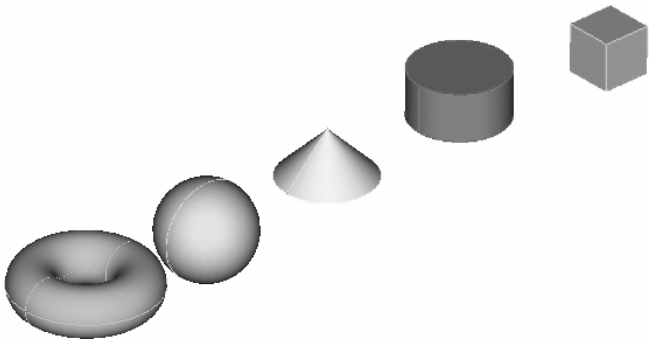


Figure 21. Primitives without pole errors after loop closure.

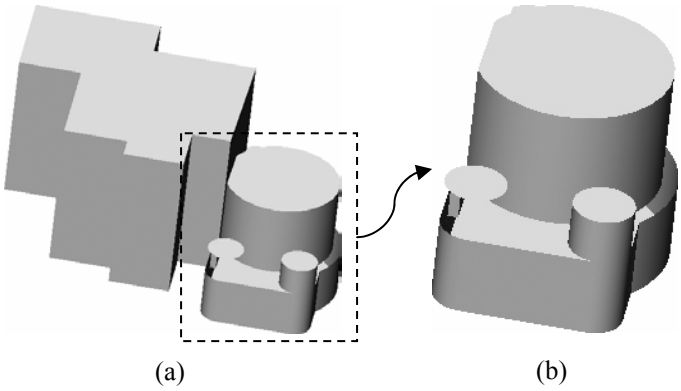


Figure 22. Motor housing unit with gap errors.

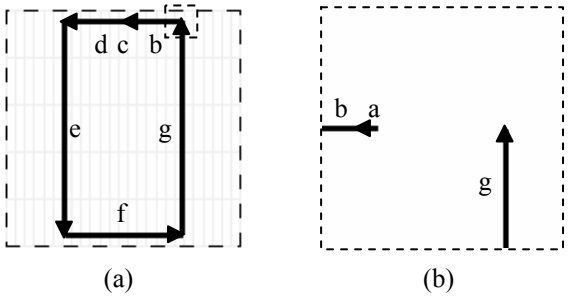


Figure 23. Parameter space trim curves exhibiting a gap error.

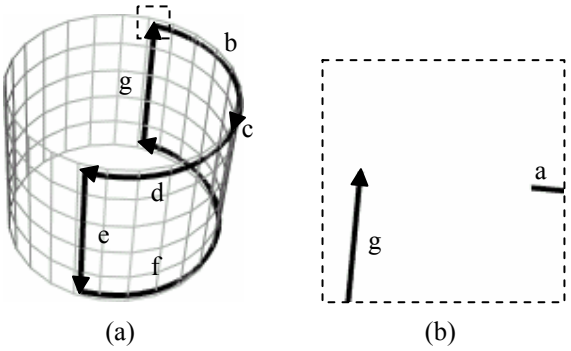


Figure 24. Model space trim curves exhibiting a gap error.

Figure 25 (a)-(b) and Fig. 26 (a)-(b) show the complete parameter space and model space definition after having passed through the loop closure algorithm. Computing the common intersection point of parameter space curves *a* and *g* closes the gap (Fig. 25 (b)). Figure 27 shows the motor housing unit after having passed through the loop closure algorithm.

Detailed descriptions of other typical errors, including seam errors and degenerate edge errors are presented in [31].

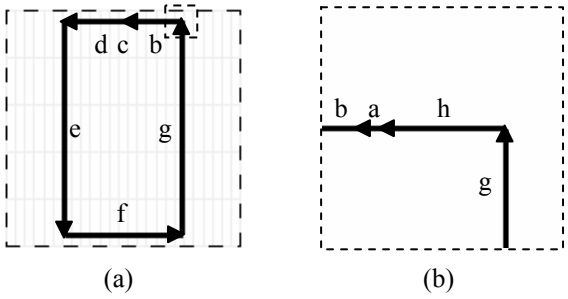


Figure 25. Parameter space trim curves without a gap error.

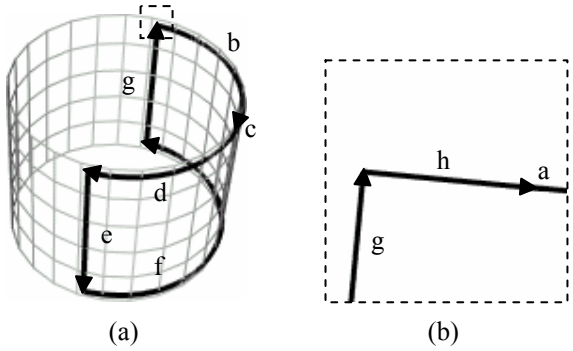


Figure 26. Model space trim curves without a gap error.

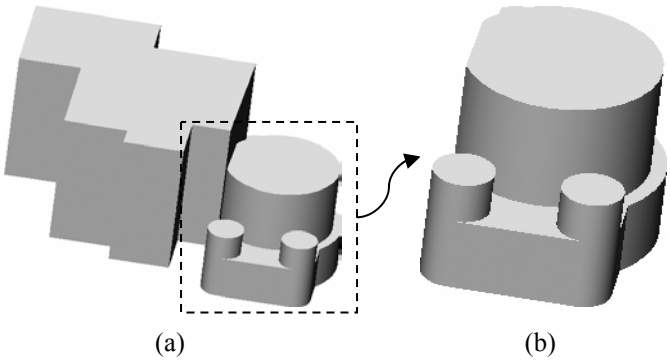


Figure 27. Motor housing unit without gap errors after loop closure.

RESULTS

This paper introduces the loop closure methodology for providing a robust definition of parameter space and model space trim curves for a given surface. The goal is to improve the reliability of data transfer between different CAD/CAM/CAE systems as well as other engineering applications, analysis tools, and visualization software using the neutral file formats IGES and STEP.

This method demonstrates a marked improvement of data transfer over the default translators and exporters provided by vendors. Of the 24 files of various complexities tested that needed correction, 22 of them were corrected completely, for a 91.7% success rate. There were a total of 10,229 faces within the 24 files, of which, 330, or 3.22%, that needed repair or completion. Of these 330 faces, 326 of them were properly corrected, for a success rate of 98.8%, leaving only 0.0391% of the total number of faces still with errors.

Table 1 lists the test files used along with their file type and the CAD package that was used to create them. These IGES and STEP files had varying degrees of missing topological information and were submitted to Engineering Animation, Inc. for analysis and repair. From this list, it is evident that many different CAD packages suffer from similar poor definitions of their models. This table shows the statistics that were gathered testing these files. However, this only represents the files that had trim curve problems, many files were tested that did not contain any errors.

ACKNOWLEDGMENTS

The authors wish to thank the Virtual Reality Applications Center at Iowa State University and Engineering Animation, Inc.

Table 1. Test file statistics.

File #	CAD Package	File Type	# of Parts	# of Faces	# of Bad Faces	% Bad Faces	# of Repaired Faces	% of Repaired Faces
1	I-DEAS Master Series 5	IGES	1	32	1	3.1	1	100.0
2	PDGS Version 26.03	IGES	1	140	29	20.7	26	89.7
3	Pro/E	IGES	1	17	2	11.8	2	100.0
4	Pro/E	IGES	1	108	2	1.9	2	100.0
5	Pro/E	IGES	1	9	4	44.4	4	100.0
6	I-DEAS Master Series 7	IGES	1	50	1	2.0	1	100.0
7	I-DEAS Master Series 5	IGES	1	19	4	21.1	4	100.0
8	CoCreate SolidDesigner	IGES	1	139	2	1.4	2	100.0
9	Unigraphics 14.0	STEP	1	341	78	22.9	78	100.0
10	Unigraphics 14.0	STEP	1	341	78	22.9	78	100.0
11	Pro/E	STEP	1	163	1	0.6	1	100.0
12	Pro/E	STEP	1	371	1	0.3	1	100.0
13	Pro/E	STEP	1	108	3	2.8	3	100.0
14	CoCreate SolidDesigner	STEP	1	123	2	1.6	2	100.0
15	CoCreate SolidDesigner	STEP	12	5642	64	1.1	64	100.0
16	I-DEAS Master Series 5	STEP	1	330	18	5.5	18	100.0
17	I-DEAS Master Series 5	STEP	1	70	4	5.7	4	100.0
18	I-DEAS Master Series 5	STEP	1	43	4	9.3	4	100.0
19	Pro/E	STEP	1	79	4	5.1	4	100.0
20	CATIA	STEP	89	1075	4	0.4	4	100.0
21	Unigraphics 15.0	STEP	1	107	15	14.0	15	100.0
22	EAI STEP Exporter 1.0	STEP	27	661	4	0.6	4	100.0
23	Pro/E	STEP	5	238	2	0.8	2	100.0
24	CATIA	STEP	3	23	3	13.0	2	66.7

REFERENCES

[1] Bohn, J. H., 1993, "Automatic Cad-Model Repair," Ph.D. Thesis, Rensselaer Polytechnic Institute, Troy, New York.

[2] Farouki, R. T., 1999, "Closing the Gap Between CAD Model and Downstream Application." *SIAM News*, **32**.

[3] *CADfix*, International TechneGroup Incorporated, http://www.iti-oh.com/.

[4] *GDX*, XOX Corporation, http://www.xox.com/.

[5] *DesignQA* and *GeometryQA*, Avatech Solutions, Inc., http://www.avatechsolutions.com/.

[6] *TransMagic*, TransMagic, Inc., http://www.transmagic.-com/.

[7] *Mirror Model Comparator*, Translation Technoliges, Inc., http://www.translationtech.com/.

[8] *CADhealer*, Theorem Solutions, Inc., http://www.theorem.co.uk/.

[9] *Bodyshop*, UGS PLM Solutions, http://www.ugs.com/.

[10] Tautges, T. J., 2000, "The Common Geometry Module (CGM): A Generic, Extensible Geometry Interface," *Proceedings of the 9th International Meshing Roundtable*, New Orleans, Louisiana, pp. 337-348.

[11] Merazzi, S., Gerteisen, E. A., and Mezentsev, A., 2000, "A Generic CAD-Mesh Interface," *Proceedings of the 9th International Meshing Roundtable*, New Orleans, Louisiana, pp. 361-370.

[12] Krause, F.-L., Stiel, C., and Lüddemann, J., 1997, "Processing of CAD-Data – Conversion, Verification and Repair," *Proceedings of the 4th ACM Symposium on Solid Modeling and Applications*, Atlanta, Georgia, pp. 248-254.

[13] Petersson, N. A., and Chand, K. K, 2001, "Detecting Translation Errors in CAD Surfaces and Preparing Geometries for Mesh Generation," *Proceedings of the 10th International Meshing Roundtable*, Newport Beach, California, pp. 363-371.

[14] Mezentsev, A. A., and Woehler, T., 1999, "Methods and Algorithms of Automated CAD Repair for Incremental Surface Meshing," *Proceedings of the 8th International Meshing Roundtable*, South Lake Tahoe, California, pp. 299-309.

[15] Butlin, G., and Stops, C., 1996, "CAD Data Repair," *Proceedings of the 5th International Meshing Roundtable*, Pittsburgh, Pennsylvania, pp. 7-12.

[16] Shen, G., Sakkalis, T., and Patrikalakis, N. M., 2001, "Analysis of Boundary Representation Model Rectification," *Proceedings of the Sixth ACM Symposium on Solid Modeling and Applications*, Ann Arbor, Michigan, pp. 149-158.

[17] Steinbrenner, J. P, Wyman, N. J., and Chawner J. R., 2000, "Fast Surface Meshing on Imperfect CAD Models," *Proceedings of the 9th International Meshing Roundtable*, New Orleans, Louisiana, pp. 33-41.

[18] Uva, A. E., Monno, G., and Hamann, B., 1998, "A New Method for the Repair of CAD Data with Discontinuities," *Proceedings of Atti del II Seminario Italo-Espanol - Progettazione e Fattibilita dei Prodotti Industriali (Diseno y Fabricabilidad de los Productos Industriales),* F. Caputo et al., eds, pp. 59-75.

[19] Sheffer, A., Blacker, T., and Bercovier, M., 1998, "CAD Data Repair Based on Virtual Topology," *Proceedings of DETC'98, 1998 ASME Design Engineering Technical Conferences*, Atlanta, Georgia, pp. 1-10.

[20] Sheffer, A., Blacker, T., Clements, J., and Bercovier, M., 1997, "Virtual Topology Operators for Meshing," *Proceedings of the 6th International Meshing Roundtable*, Park City, Utah, pp. 49-66.

[21] Barequet, G., and Kumar, S,, 1997, "Repairing CAD Models," *Proceedings IEEE Visualization '97*, Phoenix, Arizona, pp. 363-370.

[22] Barequet, G., and Sharir, M., 1995, "Filling Gaps in the Boundary of a Polyhedron," Computer Aided Geometric Design, **12**, pp. 207-229.

[23] Mortenson, M. E., 1985, *Geometric Modeling*, John Wiley & Sons, New York.

[24] Hoffmann, C. M., 1989, *Geometric and Solid Modeling*, Morgan Kaufmann Publishers, San Mateo, California.

[25] Segal, M., 1990, "Using Tolerances to Guarantee Valid Polyhedral Modeling Results," *Computer Graphics*, **24**, pp. 105-114.

[26] Jackson, D. J., 1995, "Boundary Representation Modelling with Local Tolerances," *Proceedings of the 3rd ACM Symposium on Solid modeling and Applications*, Salt Lake City, Utah, pp. 247-254.

[27] Lipson, H., 1998, "Computer Aided 3D Sketching for Conceptual Design," Ph.D. Thesis, Technion Isreal Institute of Technology, Haifa, Israel.

[28] Shpitalni, M., and Lipson, H., 1997, "Classification of Sketch Strokes and Corner Detection using Conic Sections and Adaptive Clustering," Transactions of the ASME Journal of Mechanical Design, **119**, pp. 131-135.

[29] Martin, W., Cohen, E., Fish, R., and Shirley, P., 2000, "Practical Ray Tracing of Trimmed NURBS Surfaces," Journal of Graphics Tools, **5**, pp. 27-52.

[30] 2002, *PDElib*, International TechneGroup Incorporated.

[31] Assadi, A. D., 2003, "CAD Model Robustness Assessment and Repair," Ph.D. Thesis, Iowa State University, Ames, Iowa.

