A New Method for the Repair of CAD Data with Discontinuities

Antonio E. Uva[1,2], Giuseppe Monno[1], Bernd Hamann[2]

[1] *Dipartimento di Progettazione e Produzione Industriale*
Politecnico di Bari
Bari, Italy
uva@dppigr.poliba.it, gmonno@poliba.it

[2] *Center for Image Processing and Integrated Computing (CIPIC)*
Department of Computer Science
University of California, Davis
{uva,hamann}@cs.ucdavis.edu
Davis, CA, USA

Keyworks: CAD, solid modeling, error correction, approximation, discontinuity, boundary representation.

Abstract

It is imperative for CAD, solid modeling, grid generation, and data exchange applications that a complicated, surface-based geometry definition does not contain discontinuities in the form of overlapping surfaces, gaps between surfaces or surface intersections. Most currently used approaches require a significant amount of user interaction to correct faulty CAD data. We describe a new approach for approximating large-scale CAD data sets by a relatively small number of patches that no longer contain discontinuities -- except those that are part of the design. Our approach first determines connectivity information for the given patches and removes discontinuities smaller than a certain user-specified tolerance. Next, feature lines are determined in the model by identifying patch boundary curves where large changes in the surface normal occur. Original patches with in regions bounded by feature lines are then grouped together, and a "blending-edge" post-processing step ensures C^0 continuity of the entire model. Finally, bigger approximating patches replace each group of patches. The output of our algorithm is a much smaller set of patches with C^0 continuity.

1. INTRODUCTION

Typically, a complicated real-world geometry is defined in terms of thousands of patches, usually piecewise polynomial (or even rational) B-spline patches. Complicated models often contain errors due to inconsistent model modifications or various designers working on the same model without necessarily enforcing consistency checks and enforcement. Inconsistencies / errors are characterized by surface patches that do not connect properly or those that intersect each other.

Computer-generated CAD models are becoming increasingly complex due to the increase in computing power and storage capacity.

This only aggravates the problem of surface inconsistencies. A solution to this problem is given by highly automated algorithms, requiring minimal user input, "correcting" CAD data with inconsistencies / discontinuities. The undesired "gaps," "overlaps," and "intersections" in a CAD model usually prohibit further processing of the CAD data. Such further processing might be the automatic definition of a grid for fluid flow simulation for aircraft configurations or even an automated manufacturing process. For grid generation, flow analysis and manufacturing to be done automatically and efficiently it is crucial that the supplied CAD data are "error-free." The problem at hand most commonly arises in the aircraft and automobile industries. A robust, semi-automatic method is of great importance for CAD and post-processing processes, including computer aided manufacturing (CAM). Most methods described in the literature and used in practice require a significant amount of user interaction.

Typically, the user will have to interactively "repair" CAD data by utilizing certain CAD operations and locally replacing data by some suitable approximation that corrects the inconsistencies in the model. This is time-consuming and still leaves room for introducing additional error. Our method assumes that we are given a set of CAD patches without explicitly described connectivity. The data might or might not contain discontinuities. The output of our method is a set of patches that approximate the original ones within a certain tolerance and that define a new model that is free of discontinuities and is consistent. Considering a realistic, complicated model, consisting of thousands of patches, we believe that a certain degree of user-interaction will always be needed. There is the issue of "tolerance," and it might be hard, if not impossible, for a method to automatically define the right tolerance value to be used to identify discontinuities and characterize them properly as "desired" or "undesired."

A completely automated approach for solving this problem might contradict a designer's original intention. Therefore, one must keep in mind that certain "problems" will still require a user to interactively correct a faulty part of geometry.

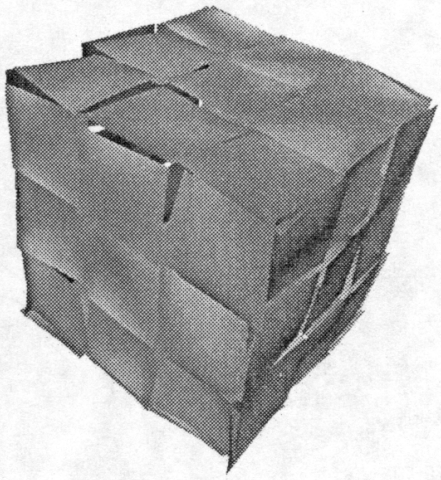


Figure 1.a. Input data.

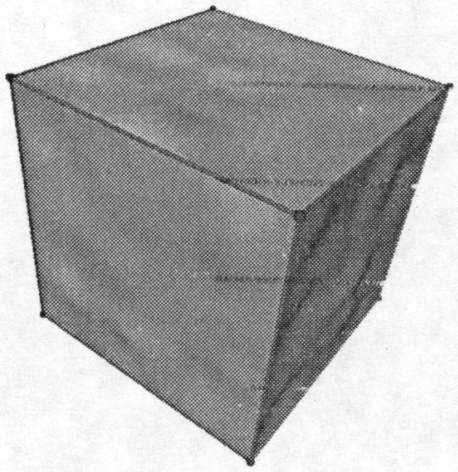


Figure 1.b. Final model.

Figure 1 illustrates the basic concept of our algorithm -- the input of our algorithm (54 disconnected patches, Figure 1.a), and the output (six approximating patches with no discontinuities, Figure 1.b).

1.1. Related Work

The classical literature in geometric modeling and grid generation does not provide "help" for solving the faulty-CAD-data problem, which one encounters in practical problems rather often. Most grid generation systems provide simple CAD functionality that is used to correct faulty CAD data prior to the application of automated grid generation algorithms -- that require a continuous surface description. The method we present in this paper is based on mathematical methods well understood in geometric modeling; we have combined existing methods with the eventual goal of accelerating and automating -- when possible -- the time-consuming task of correcting large CAD data sets. We use fundamental concepts of geometric modeling and computer aided geometric design (CAGD) for the algorithm we present. These fundamental concepts are described in detail in [1], [2], [3], [4], [5], [6], [7]. One interesting approach for correcting fault CAD data is highlighted in [8]; this approach is based on these steps:

(i) constructing "rough" local approximants to a given CAD data set;
(ii) projecting these initial approximants onto the original patches; and
(iii) using the projections lying exactly on the geometry as interpolation conditions for the definition of a much "closer" approximation.

A user can interactively specify the boundary curves of the "rough," initial surface approximants that are then automatically refined to the "closer" approximations. Finally, one obtains a continuous geometry representation consisting of a much smaller number of patches.

1.2. Our Approach

We present a topology-based algorithm having two primary goals in mind:

- Eliminate discontinuities between neighbor patches if the discontinuities are smaller than some tolerance; and
- reduce the overall number of CAD patches describing a complex geometry by approximating multiple original patches by a single approximant one

This is the overall structure of our algorithm:

Input: List of (bi-cubic B-spline) patches -- specified by control nets and knot vectors
Output: List of (fewer) approximating patches; patches meet in C^0-continuos fashion, except in places where an original discontinuity exceeds a certain tolerance

The individual steps of the algorithm are:

- Determine the connectivity among the given patches.
- Verify the connectivity along all the edges of each patch and visualize the errors (=discontinuities) in the model.
- Determine feature lines in the model according to the normal variation along patch boundaries.

- Merge patches into a set of *group-of-patches*; stop the merging process when a feature line is reached.
- Merge the control information (boundary discontinuities are removed for those original edges deviating less than a certain tolerance) to define a new approximating patch.
- Repeat the process with different user-defined parameters, if necessary.

These steps are discussed in detail in the following sections. Section 2 presents an overview on the topological approach, a basic concept we use in our algorithm. Section 3 discusses the algorithm to determine the connectivity among patches. Section 4 describes how to average the original patch boundary curves to guarantee C^0 continuity in the whole model. Section 5 describes the recognition of the feature lines to be preserved in the model and the merging phase used to reduce the number of patches. Section 6 describes the output model, the blending of patches, and computing the approximating surfaces. Sections 7 and 8 present results and ideas for future work.

2. TOPOLOGICAL APPROACH

Topology is the mathematical study of properties of objects, which are preserved through deformations. A circle is topologically equivalent to an ellipse, and a sphere is equivalent to an ellipsoid. Topology began with the study of curves, surfaces, and other objects in the plane and 3-space. One of the central ideas in topology is that spatial objects like circles and spheres can be treated as objects in their own right, and knowledge of objects is independent of how they are "represented" or "embedded" in space. The "objects" of topology are often formally defined as topological spaces.

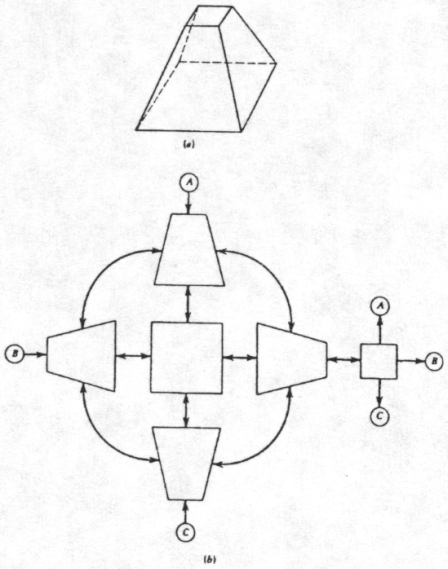
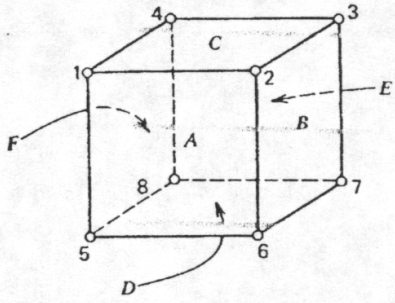


Figure 2. Topological atlas of a truncated pyramid.



	A	B	C	D	E	F
A	0	1	1	1	0	1
B	1	0	1	1	1	0
C	1	1	0	0	1	1
D	1	1	0	0	1	1
E	0	1	1	1	0	1
F	1	0	1	1	1	0

Figure 3. Connectivity matrix for faces of a cube.

Topological spaces are crucial to represent 3D models in CAD. A topological representation allows us to ignore geometry and physical location of vertices, edges, and surfaces for certain types of CAD data processing. Each model can be described as a list of geometric surfaces (patches in our case) plus the topological connectivity among them (the connectivity among patches is based on shared edges). Any model can be represented in topological space as an "atlas" (Figure 2). To work in this space we need to represent the "atlas" in an accessible format, e.g., a matrix. The connectivity matrix is a binary matrix; zero-valued elements indicate no connectivity, and one-valued elements indicate connectivity between two elements (Figure 3).

3. DETERMINING PATCH CONNECTIVITY

Typically, a CAD definition of a complex geometry consists of individual patches without topological information describing the connectivity among patches. Furthermore, most CAD systems do not necessarily enforce continuity conditions when a designer is operating on a pair of patches that should be continuous along a common boundary curve. We therefore have to worry about determining the connectivity information for a given set of patches. We assume that:

(i) All patches are supposed to be sharing entire boundary curves with neighboring patches (except on the boundary of a non-closed geometry).
(ii) Triangular patches are degenerate four-sided patches, where one edge is collapsed to a corner.
(iii) No more than four (topologically four-sided) patches may share a common vertex.

The evaluation of the connectivity is based on a user-defined parameter-threshold ($\mathbf{u}_1$) for maximum distance between vertices. For each patch and for each vertex **V** of the patch a search for the at most 3 closest vertices is conducted and these must be in the sphere centered at **V** with radius $\mathbf{u}_1$. The same check is applied to the vertices belonging to the edges connected to the 3 closest vertices; if they are within the same tolerance $\mathbf{u}_1$, then an edge connectivity (and a patch connectivity) is established (Figure 4). This is a high-level description of the connectivity algorithm:

Algorithm 1: ***Creating patch connectivity***
Input: *patches, corner vertices*
Output: *connectivity*

```
for each patch P do
    for each vertex V of P do
    {
        • find the 3 closest vertices V1 ,V2 ,V3 ;
        if distance(V ,Vi )<u1
            if the vertex connected (by edge) with Vi is close to the patch P
                • add connection between P and the patch associated to Vi ;
    }
```

T-connectivity (Figure 5.a) must be considered. Not only a search for the three closest vertices is performed, but also a search for the closest edge. If the minimum distance between this

edge and $\mathbf{V_i}$ is less than $\mathbf{u_1}$, then the edge is automatically split, and a new connectivity vertex is created (Figures 5.b, 5.c). At the end of this phase, we have determined the connectivity among patches and edges based only on geometric distance among the four corner vertices of each patch.

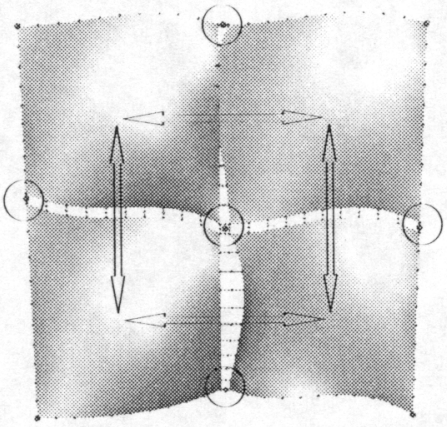


Figure 4. Patch connectivity.

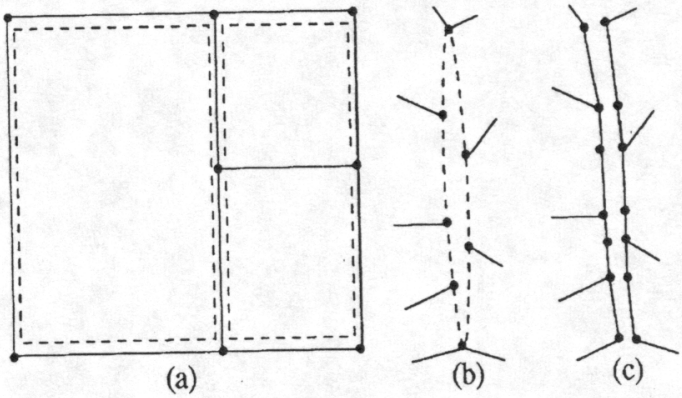


Figure 5. T-connectivity and creation of "connectivity vertices."

3.1. Verification of Patch Connectivity

In most cases it is sufficient to know about connectivity between two patches, but in some cases a gap exists in the middle of an edge even if the corner vertices are well connected (Figures 6.a, 6.b). To ensure that edge connectivity is not established in the presence of such "interior gaps," an additional test is required to verify each edge connectivity.

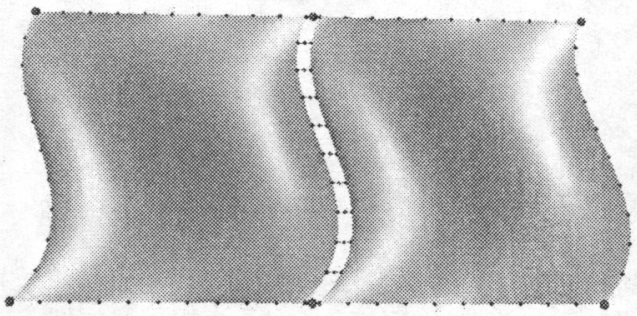


Figure 6.a. Edge connectivity established.

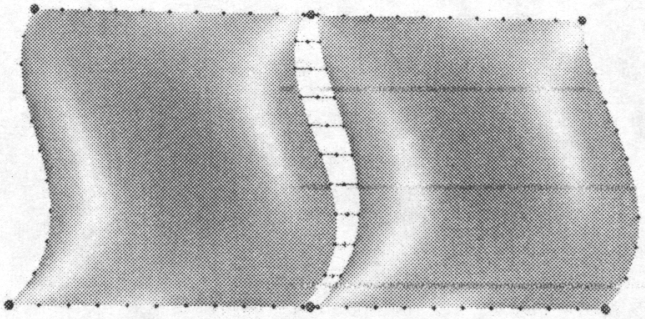


Figure 6.b. No edge connectivity established.

In order to validate the connectivity among the "candidate edge neighbors," we discretize each edge by a certain number of points and check the connectivity using a distance metric applied to this discrete boundary curve representation. Intuitively, what we do is this: If two boundary curves of two patches are meant to be shared by the two patches as a common boundary curve, closest point pairs, consisting of one point of each of the two curves, will be having fairly small distances in physical space. If the maximal distance of two boundary curves of two different patches is smaller than some user-specified tolerance $\mathbf{u_2}$, we call the two edges (patches) neighbors (the maximal distance is the maximum of all distances of closest point pairs.)

If the maximal distance exceeds $\mathbf{u_2}$, then the edges are marked as disconnected, the error is visualized in the model, and the user is asked to manually correct the error or globally/locally change the threshold used to determine connectivity information.

Obviously, certain boundary curves might not be identified as shared boundary curves of two patches even if they are visually rather close. The fact that our approach might not identify a pair of curves as a common boundary curve has two explanations:

(i) The used tolerance is too small; or
(ii) there is one isolated point on one of the two curves whose closest point on the other curve is further away than the allowed tolerance.

We believe that it is impossible to completely automate the process determining connectivity; too many parameters and too many design objectives, unknown to the computer, exist to determine connectivity in accordance with a designer's goals. We therefore favor a semi-automatic approach, where the system performs "most of the work," and the user manipulates the results of an initial connectivity guess by changing tolerance values or manually establishing connectivity for certain pairs of patches.

4. AVERAGING THE ORIGINAL PATCH BOUNDARY CURVES

Once the connectivity information is known, it is possible to "fill in" the potentially existing gaps between pairs of boundary curves that are meant to be geometrically identical. We construct new curves by performing a "discrete averaging step," which works as follows:

(i) we discretize each boundary curve by a certain number of points;
(ii) for each point on one original boundary curve we determine the closest point on the "opposing" boundary curve;
(iii) we compute the midpoints of the resulting closest point pairs;
(iv) if there are n patches supposed to be sharing a particular corner vertex (end points of the edges), we average the n individual, distinct corner points of the involved patches; and
(v) we interpolate the midpoints by an interpolating cubic B-spline.

The result of this averaging procedure is a set of boundary-blended cubic B-splines without any "gaps," where boundary curves are meant to be shared and to be geometrically the same. The problem of evaluating the B-spline curve approximant (of order four) can be outlined algorithmically as follows:

Input: List of data points x_i (derived directly from the blended midpoints)

Output: (Cubic) B-spline curve $\mathbf{S}$, determined by knot vector and control vertices d_i satisfying the approximation conditions $S(d_i) \approx x_i$

We utilize the elegant solution given by Farin[1], but one could use several other methods as described in [9], [10], and [11].

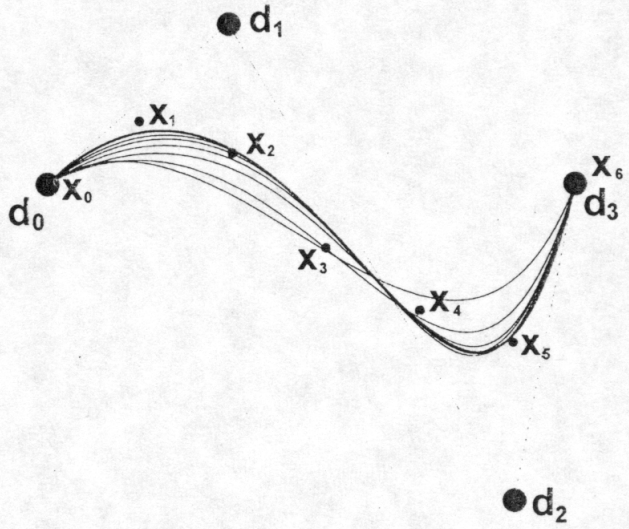


Figure 7. Cubic B-spline -- iterative approximation of seven points

Figure 7 shows an approximation of seven points using a cubic B-spline with four control points and quadruple end knots. The quality of the common boundary curves obtained by averaging point pairs is largely determined by the parametrization of the individual patches along their boundaries and the point distribution on the boundary curves resulting from the discretization step. We deal with this issue by distributing points along boundary curves uniformly with respect to arc length.

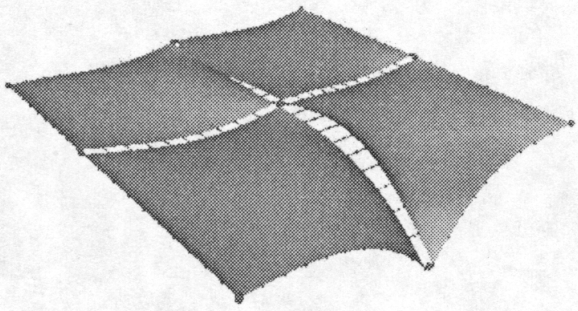


Figure 8.a. Edge connectivity.

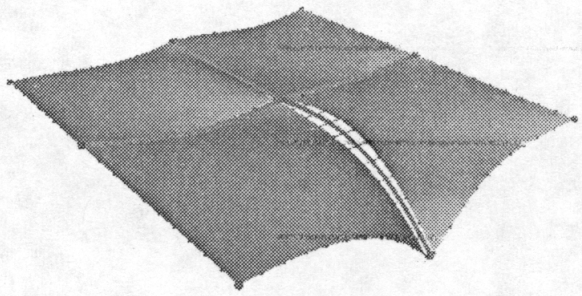


Figure 8.b. Creation of new edges by averaging.

Figure 8 illustrates the construction of shared boundary curves by averaging the original, disconnected edges -- based on the connectivity shown in the left image.

5. REDUCTION OF THE NUMBER OF PATCHES

One of our goals is to approximate groups consisting of large numbers of given patches, by single, much larger patches. To achieve this goal our algorithm uses "region growing" steps that produce a new patch replacing two old ones. This algorithm is defined by a set of rules guiding the "growing" of the patches. The rules are defined for the topological representation of the model. Each "growing" region is defined as a subspace of the topological space bounded by certain constrains. A constraint in our case is a ***feature line*** in the model that we want to preserve. We identify feature lines inherent to the given geometry of the model. In other words, it is our goal to "replace" a large number of given patches by a larger patch with the constraint that no feature lines "lie inside" the larger patch -- a large, approximating patch must stop at a feature line.

5.1. Identifying Features in the Model

Automated feature recognition has a wide range of applications in mechanical engineering. Many feature recognition methods are greatly limited in scope. Our assumption is that the feature in a model are "sharp" edges between two patches. The definition of "sharp" is related to the maximum variation in normal vector (or tangent plane) along shared boundary edges. Furthermore, we assume that certain boundary curves (or parts of boundary curves) of certain patches constitute feature lines. We check, for each connected pair of edges, the maximum angle between the normals for a fixed number of sample points (Figure 9).

We are primarily interested in feature lines that coincide with parts of patch boundary curves -- nevertheless, there might be large normal vector variations even in the interior of a patch. We compute the (unit) normal vectors along a patch boundary curve by computing the tangent vector of the boundary curve, by computing the cross-derivative vector along the boundary curve, and computing the cross product of these two vectors for each point on the polygonal representation of each boundary curve.

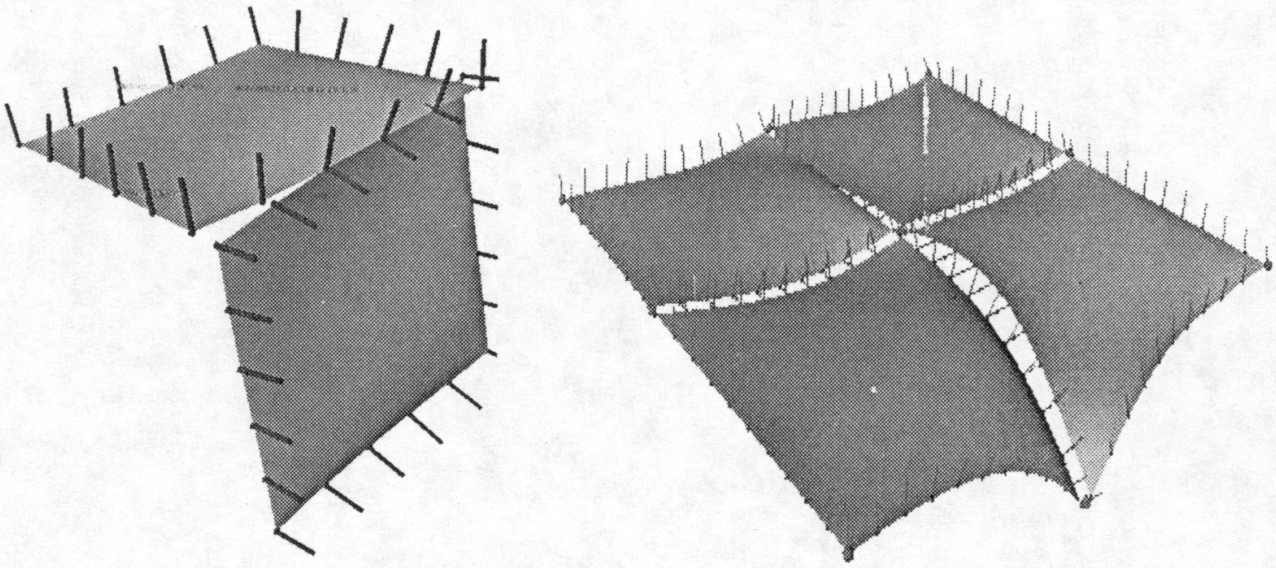


Figure 9. Feature recognition by analyzing normal variation.

Again, identifying feature lines is a numerically highly "sensitive" matter. We use the following criterion to make a first guess: If the angle between two (normal) vectors of a pair of closest points along a shared boundary curve of two patches is larger than some tolerance, then this indicates the existence of a feature line (or "feature edge"). The automatic identification of feature lines is even more sensitive than the identification of patch connectivity, and it is thus imperative to allow a user to manually add or delete parts of feature lines.

5.2. Creating a set of *Group-of-Patches*

It is one of our major goals to reduce the number of original patches and replace "groups" of original patches by large approximating patches. A necessary pre-processing step to be performed prior to the actual approximation is the identification of subsets of the original patches to be replaced by larger patch approximants. We call this pre-processing step "creating *group-of-patches*," and it is based on the following idea: Construct a *group-of-patches* by merging original patches until the construction of a *group-of-patches* hits a feature line in the model; feature lines are to become parts of the boundaries of a *group-of-patches*.

We now explain the construction of a *group-of-patches* in more detail. We assume, as mentioned before, that the given set of patches is (at least locally) bi-directional in topological space. This means that it must be possible to associate a topological i- and j-direction to each original patch, i.e., it is possible to view (at least locally) the patches as patch rows and patch columns. The bi-directionality in topological space implies that

(i) each patch has at most one neighbor patch along a boundary curve (no T-connectivity allowed); and
(ii) at most four patches may have a corner vertex in common.

These assumptions allow us to define the four "growing directions" called for similarity: *north, south, east, and west.*

The "growing" process starts with merging patch as in a random direction. Before merging two patches (or two *group-of-patches*) a verify is necessary to determine if the two edges are compatible, i.e., if they have the same "topological length" (= defined by the same number of control points). We define the aspect ratio ρ of *group-of-patches* as:

$$\rho = \frac{length(north\_edge) + length(south\_edge)}{length(east\_edge) + length(west\_edge)} \quad . \tag{1}$$

The next "growing direction" is determined by the objective to optimize ρ (optimal is $\rho = 1$).

We construct the largest possible union of patches, according to the previous set of rules, until a feature line is reached as a edge (or a part of edge). Then, we try to grow in another direction sacrificing the aspect ratio ρ. We terminate the growing process when a feature line is part of each of the four *group-of-patches* boundary edges. On a high level, our algorithm looks like this:

Algorithm 2: ***Patch growing***
Input: *list of patches*
Output: *list of group-of-patches*

while** growing is possible **do

{

- *select a growing direction (according to the optimal ρ);*
- *check the "topological length" of the edges to be merged;*
- *check the presence of a **feature line**;*
- *merge the two group-of-patches;*

}

The creation of a set of *group-of-patches* leads to the definition of areas having feature lines as parts of their boundaries but no feature lines in their interior. The notation *"group-of-patches[i,j]"* means that *i* rows of patches are grouped in the north/south direction and *j* columns of patches are grouped in the east/west direction. The total number of patches grouped in *group-of-patches[i,j]* is *i* times *j*. We will approximate the set of original patches defining each one of these groups by larger approximating patches. The number of *group-of-patches* created by this procedure does depend on the particular initial patches being selected to start the grouping process and the order in which grouping is performed in the four directions. At this point, we have no conclusive answers regarding an optimal strategy leading to a minimal number of *group-of-patches*. This will be an issue for future research.

6. MERGING THE CONTROL INFORMATION OF GROUP-OF-PATCHES

In order to represent a set of B-spline patches as a larger, "conglomerate" patch, we have to merge the underlying, individual control information for each patch, i.e., we have to merge the knot vectors and the control points of the individual pieces. It is only essential to describe the process of merging the control information of two B-spline curves to be merged into a single B-spline curve:

(i) We merge the two knot vectors by performing a "union" operation for the two knot vectors (using a quadruple knot at the break point of the two curves to preserve continuity); and

(ii) we merge the control points by performing a "union" operation for the d_i's (using a double control point where the two curves meet).

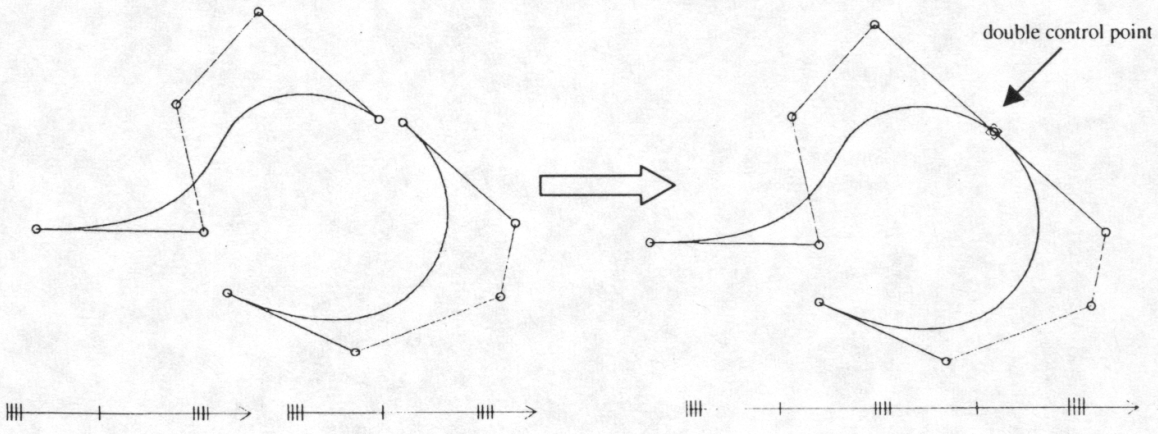


Figure 10. B-spline "union" operation for control points and knot vectors.

Figure 10 shows the B-spline merging process. The final model is composed of bi-cubic B-spline patches (C^2-continuous inside each patch). The continuity is only $\mathbf{C^0}$ where feature lines are present (boundaries of the B-spline patches). Each *group-of-patches* is eventually characterized by a matrix $\boldsymbol{M_{B\text{-}spline}}$ of control points, obtained by the method described in section 5.2, where

(i) all the original control net sub-matrices are ordered according to the topological row/column order;
(ii) the rows (columns) related to the boundary of each original patch are blended; and
(iii) the first and the last row (column) of the matrix are directly obtained from the blended boundary B-spline curves.

For example, the matrix $\boldsymbol{M_{B\text{-}spline}}$ for a 2x2 *group-of-patches* consisting each of a single bi-cubic patch, defined by (2), can schematically be illustrated as shown in Figure11.

$$\begin{bmatrix} Bn_1 = Bw_1 & Bn_2 & Bn_3 & Bn_4 & Bn_5 & Bn_6 & Bn_7 = Be_1 \\ Bw_2 & d^1_{1,1} & d^1_{2,1} & \left(d^1_{3,1}+d^2_{0,1}\right)/2 & d^1_{1,1} & d^1_{2,1} & Be_2 \\ Bw_3 & d^1_{1,2} & d^1_{2,2} & \left(d^1_{3,2}+d^2_{0,2}\right)/2 & d^1_{1,2} & d^1_{2,2} & Be_3 \\ Bw_4 & \left(d^1_{1,3}+d^3_{1,0}\right)/2 & \left(d^1_{2,3}+d^3_{2,0}\right)/2 & \left(d^1_{3,3}+d^2_{0,3}+d^3_{3,0}+d^4_{0,0}\right)/4 & \left(d^2_{1,3}+d^4_{1,0}\right)/2 & \left(d^2_{2,3}+d^4_{2,0}\right)/2 & Be_4 \\ Bw_5 & d^3_{1,1} & d^3_{2,1} & \left(d^3_{3,1}+d^4_{0,1}\right)/2 & d^4_{1,1} & d^4_{2,1} & Be_5 \\ Bw_6 & d^3_{1,2} & d^3_{2,2} & \left(d^3_{3,2}+d^4_{0,2}\right)/2 & d^4_{1,2} & d^4_{2,2} & Be_6 \\ Bw_7 = Bs_1 & Bs_2 & Bs_3 & Bs_4 & Bs_5 & Bs_6 & Bs_7 = Be_7 \end{bmatrix} \quad (2)$$

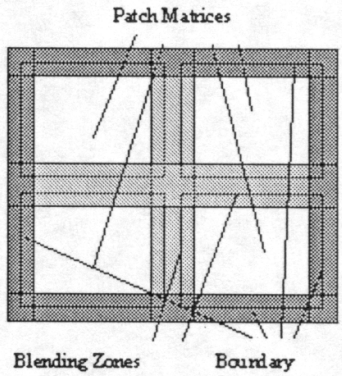


Figure 11. "Blending patch" matrix for four B-spline surfaces, each consisting of one bi-cubic patch only.

The coordinate vectors of points $\mathbf{B_n}$, $\mathbf{B_s}$, $\mathbf{B_w}$, $\mathbf{B_e}$ refer to the control points associated with the blended B-spline curves bounding the B-spline patches in the north, south, west, and east topological directions.

Once we have completed the control net for each B-spline surface, we can alter the smoothness of surfaces by modifying the knot vectors or control points. The upper-left image in

Figure 12 represents the original B-spline patches with their control nets. The other three images in Figure 12 show approximating B-spline surfaces with the same control nets but with different knot vectors. C^0 continuity along the boundary of each B-spline patch is guaranteed since the external control points are the same for each pair of contiguous B-spline patches.

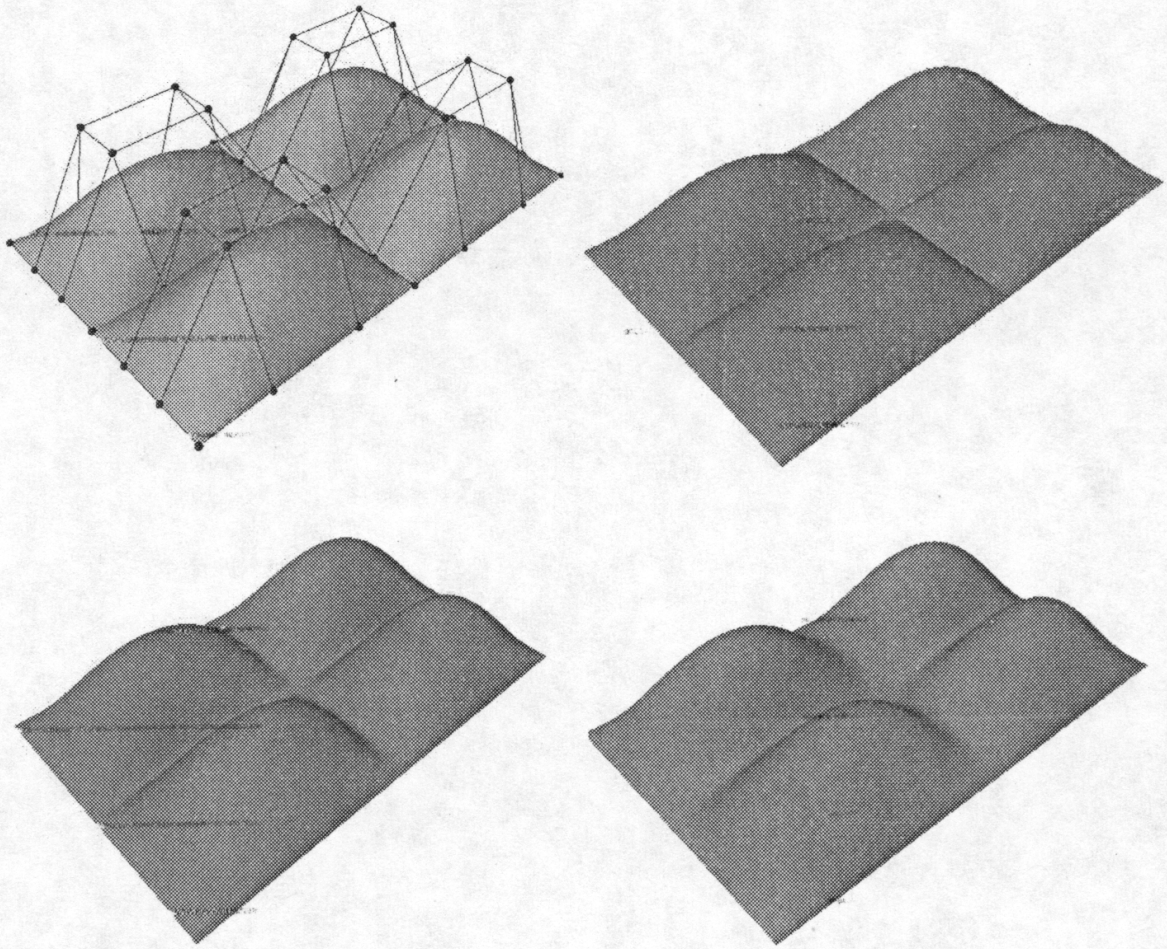


Figure 12. Original patches and different degrees of smoothness of approximant; 2x2 *group-of-patches* configuration and modification of knot vectors.

7. EXPERIMENTAL RESULTS

We have implemented the described algorithms on a Silicon Graphics Onix^2 workstation. The visualization component is based on OpenInventor. We have tested our algorithm on several data sets. Two examples are shown in Plates 1 and 2.

Plate 1 illustrates the overall approximation process. The boundary edges of the original disconnected model (Plate 1.a) are replaced by new edges obtained by edge averaging (Plate 1.b).

Plates 1.c and 1.d show the effect of using different threshold values for identifying feature lines by considering normal vector variation.

Plate 2 shows a discontinuous data set representing the geometry of a cube (Plate 2.a). The twelve edges of the cube are correctly identified as feature lines (Plate 2.c) and are preserved as boundary curves in the final approximation (Plate 2.d).

8. CONCLUSIONS AND FUTURE WORK

We have presented a new technique for the correction and approximation of large CAD data sets. Our algorithm is currently limited a certain type of patch topology; given data sets must consists of patches arranged, topologically, in a row/column-type fashion. In is necessary to handle more general patch topologies where any number of patches can share a common corner vertex. We plan to do this in the near future. Our method preserves feature lines of a given model by forcing the algorithm to use feature lines as parts of the boundaries of the approximating patches. Feature line definition and extraction are numerically rather sensitive operations, and we therefore provide mechanisms for a user to delete or add feature lines from/to the set of feature lines that the algorithm identifies automatically. We plan to conduct more research regarding a more robust way for feature line extraction. Our algorithm serves two primary purposes:

(i) elimination of undesired discontinuities; and
(ii) reducing the number of patches needed to represent a complicated geometry.

All discontinuities within a user-specified tolerance are eliminated automatically, and a user can always force the system to correct more, larger discontinuities by manual intervention. For certain applications, given CAD data sets might contain more detail than required, e.g., for a "crude" fluid flow analysis around a car body. Our algorithm provides a means for reducing the number of original CAD patches significantly by replacing them with fewer B-spline patches approximating large regions within a specified tolerance.

9. ACKNOWLEDGEMENTS

This work was supported by the National Science Foundation under contract ACI 9624034 (CAREER Award), the Office of Naval Research under contract N00014-97-1-0222, the Army Research Office under contract ARO 36598-MA-RIP, the NASA Ames Research Center under NRA2-36832 (TLL) program and the Lawrence Livermore National Laboratory under contract W-7405-ENG-48 (B335358), awarded to the University of California, Davis. We would like to thank the members of the Visualization Thrust at the Center for mage Processing and Integrated Computing (CIPIC) at the University of California, Davis.

REFERENCES

[1] Farin G. (1997), *Curves and Surfaces for Computer Aided Geometric Design -- A Practical Guide*, 4th ed., Academic Press, Boston, Massachusetts.

[2] Bartels, R. H., Beatty, J. C. and Barsky, B. A. (1987), *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann Publishers, Inc., Los Altos, California.

[3] Boehm, W. and Prautzsch, H. (1994), *Geometric Concepts for Geometric Design*, A K Peters, Wellesley, Massachusetts.

[4] de Boor, C. (1978), *A Practical Guide to Splines, Applied Mathematical Sciences*, Vol. 27, Springer-Verlag, New York, New York.

[5] Hamann, B. (1994), *Construction of B-spline approximations for use in numerical grid generation*, Applied Mathematics and Computation 65(1--2), pp. 295--314.

[6] Hoschek, J. and Lasser, D. (1993), *Fundamentals of Computer Aided Geometric Design*, A K Peters, Ltd., Wellesley, Massachusetts.

[7] Piegl, L. A. and Tiller, W. (1996), *The NURBS Book*, 2nd edition, Springer-Verlag, New York., New York.

[8] Farin, G. and Hamann, B. (1997), *Current trends in geometric modeling and selected computational applications*, Journal of Computational Physics 138(1), Academic Press, pp. 1--15.

[9] Yamaguchi, F. (1998), *Curves and surfaces in Computer Aided Geometric Design*, Springer-Verlag, New York, New York.

[10] de Boor, C. (1977), *Package for Calculating with B-Splines*, SIAM J. Numer. Anal., p. 441.

[11] Hanson, R. J. (1978), *Constrained Least Squares Curve Fitting to Discrete Data Using B-Splines*, Sandia National Laboratories Technical Report, SAND-78-1291.