BUTTERWORTH HEINEMANN

An algorithm for smoothly tessellating β-sheet structures in proteins

Lawrence D. Bergman,* Jane S. Richardson,† and David C. Richardson†

**Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599, USA*

†Department of Biochemistry, Duke University, Durham, NC 27710, USA

An algorithm is presented for tiling the surface of β-sheet structures, smoothed both along the strands and perpendicular to them. The algorithm is intended for creating graphical representations of β structure within the three-dimensional context of proteins. This article presents sufficient detail to allow a programmer with some knowledge of protein structure to implement the tiling algorithm. Several examples of its use are illustrated.

Keywords: β-sheet, tessellated surfaces, molecular graphics, protein structure.

INTRODUCTION

Experience has proven the effectiveness of using simplified and smoothed schematic diagrams for showing three-dimensional protein structure. The commonest are ribbon diagrams, either hand drawn[1] or computer drawn.[2,3] These work extremely well, especially for moderate-sized proteins, and we wish to complement, not to replace, them.

For more complex structures, while helices can successfully be shown as cylinders, there is no accepted convention for simplifying β-sheets. Sheets have been drawn approximately as twisted or barrel-shaped surfaces,[4,5] but not in a way that represents their shape accurately. Also, these representations fail to make the strand pairings explicit. The present algorithm is proposed as a way of producing a surface representation for β-sheets that not only follows the local shape closely, but also gives an overall impression of smoothness and unity.

The algorithm described in this article has been implemented as a tool within the VIEW molecular visualization system.[6] Scripting the algorithm by using the VIEW system's interpreted language, with its facility for user-defined interaction, was beneficial in several ways:

- We were able to try a variety of algorithmic possibilities before settling on those described in this article.
- The resulting tool is integrated into an environment that allows ready generation of other representations, including α-helical cylinders and splined main chain (Figures 17–19 and 21).
- Interactive user guidance of the tool (described in Section 4) was easily implemented.

The VIEW script for the tessellation algorithm is available (contact L.D. Bergman via e-mail: bergman@cs.unc.edu).

ALGORITHM DESCRIPTION

The algorithm consists of two distinct phases. The first is a gridding phase, which identifies β strands along the main chain (hereafter referred to as *chain-strands*) and then creates a set of cross-connecting strands that run roughly perpendicular to the chain strands (hereafter referred to as *cross-strands*). These two types of strands create a roughly rectilinear grid. The second phase of the algorithm is tiling. This portion of the algorithm creates a piecewise bicubic surface, using the grid intersections as control points.

This algorithm has not been fully optimized. Our objective was to develop a fairly intuitive, working algorithm. We encourage future implementers of this algorithm to seek optimizations.

2.1 Gridding phase

The grid generation portion of the algorithm consists of (1) identifying peptide units, (2) computing grid points, (3) identifying those peptides that are in β-sheet, (4) accumulating chain-strands, (5) accumulating cross-strands, and (6) orienting both cross- and chain-strands.

2.1.1. Identifying peptide units. For the purposes of this algorithm, we consider the protein main chain to consist of peptides rather than residues. If the protein were treated as

Color plates for this article are on p. 58.

Address reprint requests to Dr. Bergman, IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598.

Received 11 February 1994; revised 24 June 1994; accepted 20 July 1994.

655 Avenue of the Americas, New York, NY 10010

0263-7855/95/$10.00
SSDI 0263-7855(94)00003-B

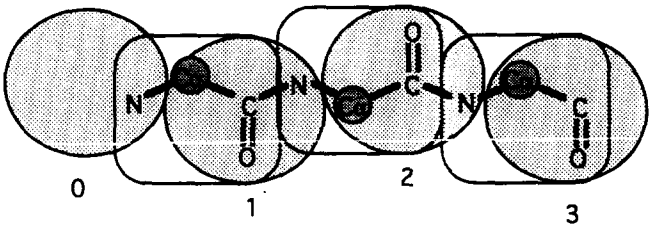

Figure 1. Reassignment of residues to peptide units.

a sequence of residues, we would need to consider three cases in looking for hydrogen-bonding patterns that characterize β-sheet: parallel, antiparallel wide pairs, and antiparallel narrow pairs.[7] With the protein treated as a sequence of peptides, only a single pattern need be considered. Furthermore, the cross-strand traversal procedure is greatly simplified.

We identify peptides as follows: the first nitrogen is assigned peptide number 0. The following α-carbon, carbonyl carbon, carbonyl oxygen, and nitrogen (from the next residue) are assigned the peptide number 1. Assignment continues in this fashion until the end of the chain is reached. The final peptide at the C-terminus will be missing a nitrogen. Figure 1 illustrates these assignments. The initial residues are shown enclosed in squares. The peptide assignments are shown in circles with accompanying peptide numbers.

As each peptide is identified, we build a data structure for it. The fields in the structure are as follows:

- *peptide-number*
- *peptide-xyz*
- *nitrogen-cross-connection-peptide*
- *oxygen-cross-connection-flag*
- *chain-strand-number*
- *chain-strand-index*
- *cross-strand-number*
- *cross-strand-index*

All fields except *peptide-number* and *peptide-xyz* (assigned as described in Section 2.1.2) are initialized to *unassigned*. Use of the fields in this structure are explained as we discuss the algorithm. The structure is stored in an array, each element corresponding to a peptide number.

2.1.2. Computing grid points. The grid intersection points are not calculated using a single peptide for each (i.e., not an atom position within a peptide, or an average associated with one peptide). Rather, midpoints between successive α-carbons along the chain are used, as diagrammed in Figure 2. Midpoints have the advantage of smoothing the pleats normally found in β-sheet, as shown in Figure 3 for both parallel and antiparallel sheet. Each midpoint is assigned an index number that associates it with the first of the two peptides used to compute it (see Figure 2) and is used to set the *peptide-xyz* field for that peptide.

The start and end of the peptide chain must be handled as special cases. The (x,y,z) position of the starting nitrogen is assigned to the first peptide (index number 0) in the chain. The position of the carbonyl carbon is assigned to the last peptide. Figure 1 shows peptide assignments for both the start and end of a main chain sequence.

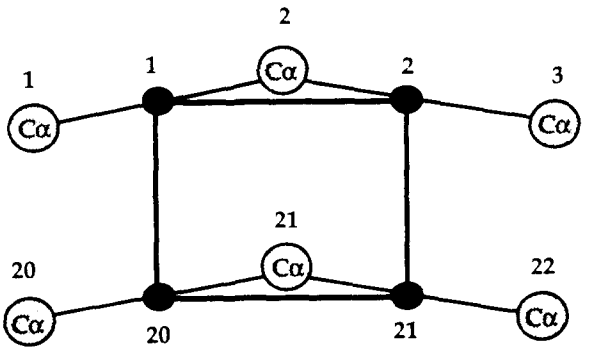

Figure 2. Computation of grid points using α-carbon positions.

The algorithm as described (and implemented) does not properly handle breaks in the peptide chain (i.e., multiple subunits). Currently, midpoints are computed even when there is a chain break; connections to these cross-subunit points can be edited out (as described in Section 4.1). A simple extension would allow us to handle breaks properly. When assigning peptide units, a distance check (perhaps between adjacent α-carbons) would be performed. When a

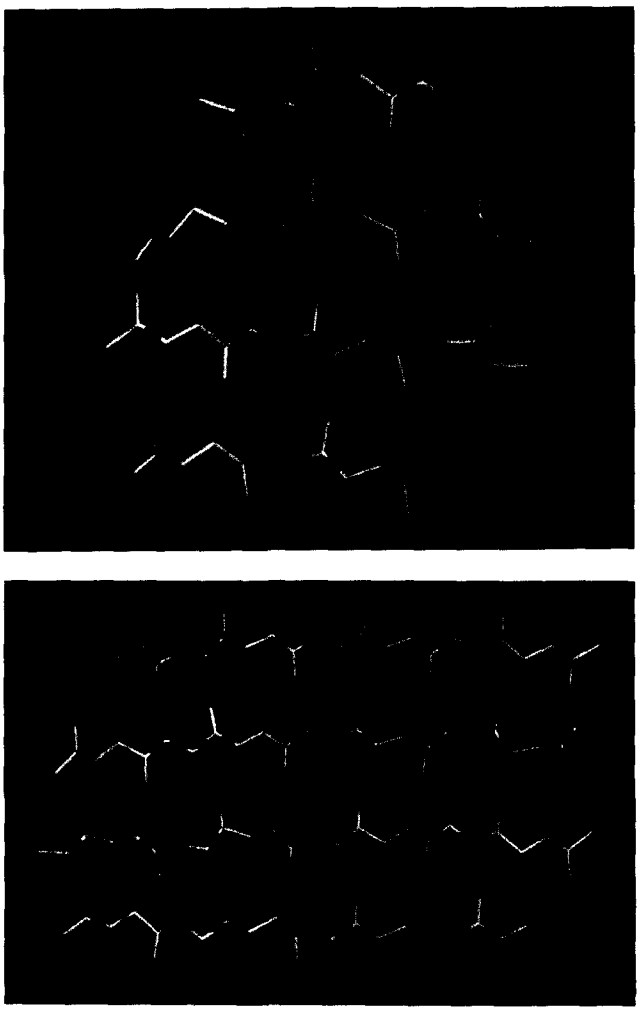

Figure 3. Grids for tessellation of (a) parallel and (b) antiparallel β-sheet. Chain strands are horizontal, cross-strands vertical.

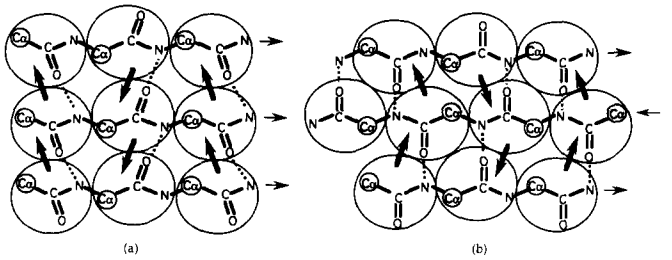

Figure 4. Hydrogen-bonding pattern and cross-strand following for (a) a parallel and (b) an antiparallel sheet.

break is detected, a peptide at the end of the segment would be assigned with no nitrogen; the nitrogen would be assigned its own peptide, and would begin a new chain segment. The end-of-segment peptide would be flagged. When chain-strands are accumulated (as described in Section 2.1.4), the end-of-segment flag would be used to ensure that strands do not cross segment breaks. The carbonyl carbon position would be used to assign each end-of-segment grid point, and the nitrogen position used to assign each start-of-segment grid point.

2.1.3. Identifying peptides in β sheet. Peptides are assigned to β structure on the basis of local hydrogen-bonding pattern. Each peptide is checked for a β-type H-bond pattern to a single adjacent strand. β Structure is characterized by hydrogen bonds between adjacent strands, with the bond alternating from nitrogen to oxygen along a strand (Figure 4). For each peptide (n), we check if the nitrogen has hydrogen bonds (we used an H-bond algorithm similar to the one in Ref. 7). If it does, we check each peptide to which it is bonded (m) in turn. We perform the following checks:

- peptide n + 1's O hydrogen-bonded to peptide m + 1's N? (parallel)
- peptide n + 1's O hydrogen-bonded to peptide m + 1's N? (antiparallel)
- peptide n − 1's O hydrogen-bonded to peptide m − 1's N? (parallel)
- peptide n − 1's O hydrogen-bonded to peptide m + 1's N? (antiparallel)

Figure 5a shows the bonding pattern for an antiparallel sheet and Figure 5b shows the pattern for a parallel sheet. Checks in each direction are necessary because we may be at the edge of the sheet at either the start or end of a chain-strand.

If a peptide is identified as β by success in terms of any of these checks, we set the following fields in the peptide data structure:

- peptide n's *nitrogen-cross-connection-peptide* is set to m.

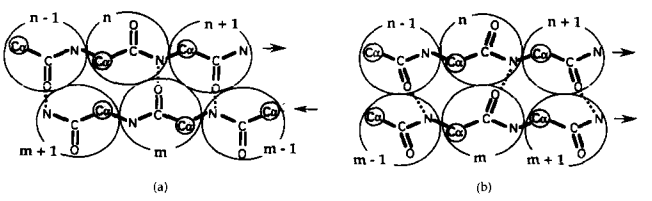

Figure 5. Cross-connection search algorithm for (a) a parallel and (b) an antiparallel sheet.

- peptide m's *oxygen-cross-connection-flag* is set to TRUE.

Note that a given peptide may have multiple H-bonds; we use only the first H-bond identified by this procedure.

2.1.4. Identifying chain-strands. Chain-strand collection is an extremely simple procedure. All peptides in the protein are checked sequentially for β cross-connections. A β cross-connection is identified by either the *nitrogen-cross-connection-peptide* field having been set (not equal to *unassigned)* or by the *oxygen-cross-connection-flag* being TRUE. Each set of sequential peptides that have cross-connections form a chain-strand. Each chain-strand is stored as an array of peptide numbers that, in turn, is stored in a strand array. The strand number (index in the strand array) is stored in the *chain-strand-number* field for the peptide, and the element number within the strand is stored in the *chain-strand-index* field.

2.1.5. Identifying cross-strands. The algorithm for collecting cross-strands processes each peptide of the protein in turn, considering only those that are identified as having a cross-connection. Each candidate peptide is checked for whether or not it lies at the start of a cross-strand. For the purpose of this algorithm, we consider cross-strands to begin at a nitrogen, and to terminate at a carbonyl carbon (in a peptide with a non-cross-connected nitrogen). A peptide that starts a cross-strand (i.e., is at the edge of a sheet) is distinguished by having a cross-connection for its nitrogen (its *nitrogen-cross-connection-peptide field* is set) and no cross-connection for its oxygen (its *oxygen-cross-connection-flag* is FALSE). When an edge residue is encountered, all other residues in that cross-strand are identified. The *nitrogen-cross-connection-peptide* field is used to retrieve the next peptide in the cross-strand, and we continue following cross-connections from one peptide to the next until a peptide with a non-cross-connected nitrogen is encountered. Each identified peptide number is added to a cross-strand array, and the *cross-strand-index* field for that peptide set to the element number in that array. Additionally the index number of the strand, from an array that holds all cross-strands, is stored in the *cross-strand-number* field.

Prior to adding a peptide to the cross-strand, the peptide is checked to see if it already belongs to a cross-strand (by seeing if the *cross-strand-number* field has been set for that peptide). If so, the peptide is not reassigned, and the cross-strand collection terminates for that strand. This case occurs where cross-strands join. Such joining of strands can occur when the sheet has bifurcated bonds; our procedure ensures that strands do not join, but leaves gaps in the grid instead. We discuss handling of these gaps in Section 4.1.

Figure 4 shows operation of the cross-strand collection procedure for both parallel and antiparallel sheet. The thick arrows indicate the direction of traversal of each cross-strand.

2.1.6. Orienting strands. The final step in creating the grid is orienting first the cross-strands and then the chain-strands. By orienting the strands, tiling the grid is greatly simplified. Tiling involves identifying four corner points for each tile (see Figure 2). Identifying these four points in complicated by the fact that either cross-strands or chain-

strands may be misoriented. Figure 6 shows an example. If we follow the cross- and chain-strands from the corner, we get the desired points in 6a, but an erroneous set in 6b.

By orienting the strands prior to tiling, we also ensure that all tiles within a connected portion of the sheet will have the same orientation. This uniform tile orientation is highly desirable, because we wish to use tiles with different colors on each side. If the tiles were not oriented, some tiles would have a front-facing color, others a back-facing color when viewed from the same side; consistently oriented tiles will give us surfaces with no color alternation.

The orientation algorithm for chain-strands consists of a simple region-growing procedure. The algorithm starts with the first chain-strand in sequence (the seed strand). This strand is processed one residue at a time. For each residue, chain-strands on either side (one residue away along its cross-strand) are checked for proper orientation (as described below) and reoriented if needed. As each adjacent chain-strand is checked, it is marked, so that the orientation check need be performed only once per strand, and it is also added to a set of yet-to-be-processed strands. When the original strand is completed, the yet-to-be-processed strand set is traversed, one strand at a time, in the same fashion. This strand checking is repeated until the yet-to-be-processed set is empty. At this point, we have found all contiguous chain-strands for a region (a *connected component* in computer science jargon). Because the protein may have more than one connected region, we next scan all chain-strands, looking for any that have not been checked. If one is found, it is used as a new seed and the algorithm restarted. When all strands have been checked, we are done.

The orientation checks at each residue are performed by constructing a vector that roughly aligns with each of the two strands to be compared. The strand following shown in Figure 6 is used to retrieve the four residues A, B, C, and D. The chain-strand is obtained for the initial residue (A), and the next residue on this strand retrieved (B). Similarly, the connected residue (C) on the A cross-strand is retrieved. Finally, the next residue (D) on the C chain-strand is retrieved. The vectors AB and CD are formed, using the α-carbon midpoints associated with each residue. The dot product between these two vectors is positive when the strands are in the same orientation, and negative when the strands are in opposite orientations. If the strands are opposite, the strand containing C and D is flipped. If A is at the end of the chain-strand, the residue that precedes it (E, not shown in the diagram) is used to form a vector (EA) in the same general direction as AB would have been. The other strand is processed similarly.

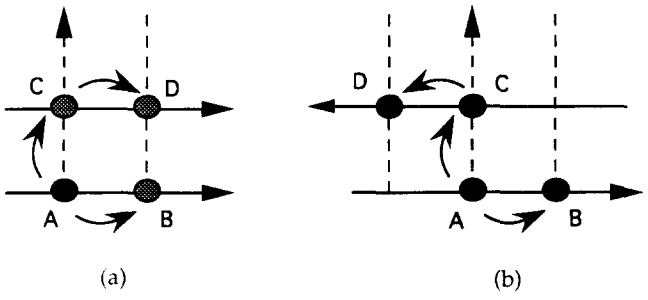

Figure 6. Grid point access for tile generation. (a) Proper chain-strand orientation; (b) improper chain-strand orientation showing erroneous set of points retrieved. Chain-strands are shown as solid arrows; cross-strands as dotted arrows.

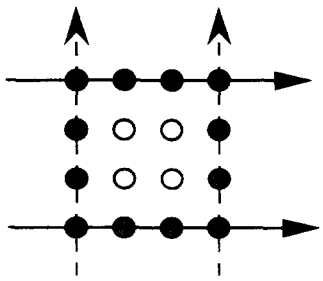

Figure 7. Sixteen control points required for a single Bezier patch—original grid points (black), intermediate points calculated along strands (gray), and interior points (white).

This procedure may not succeed in all cases. The dot product between vectors may be near zero, indicating that the strands are near 90 degrees with respect to each other. In this case, the proper orientation is not evident. We have not yet encountered a case with a dot product near zero, and do not provide for it. If such cases were to be encountered, we suggest user input to resolve the ambiguity.

Reorienting or flipping a strand is simple. The strand array is recreated with residue numbers in reverse order from the original strand. The new array is stored in the same location (in the array of chain-strands) as the original. The strand element number is updated for each residue in the strand, reflecting the new ordering of residues.

The algorithm for cross-strands is identical to that described for chain-strands.

2.2 Tiling phase

The second phase of the algorithm fits a tiled surface to the grid points. We chose to tile the sheet using triangles rather than a higher-order surface (such as nonuniform rational B-spline [nurbs]) in order to make the algorithm accessible for a wide range of implementation platforms. There are several criteria that we wish to satisfy in tiling the β-sheet.

First, we wish to smooth minor local irregularities in the β-sheet. Second, we wish to determine the tiling piecewise, restricting the information used in computing each piece to four local control points and the strands that contain them. The tiling is determined piecewise to minimize computation and to keep the implementation simple. Third, we wish to ensure that the surface tiling has continuous first derivatives. There should be no sudden changes in the slope of the surface, as we wish to avoid kinks.

For these reasons, we chose a piecewise bicubic surface with C^1 continuity (continuous derivatives; see Ref. 8). We selected Bezier patches for their ease of implementation. A Bezier surface with C^1 continuity imposes several restrictions on the control grid. The restrictions, and the means by which we enforce them, are as follows:

1. The grid must consist of adjoining, nonoverlapping 4 × 4 regions, each of which will produce a single patch.

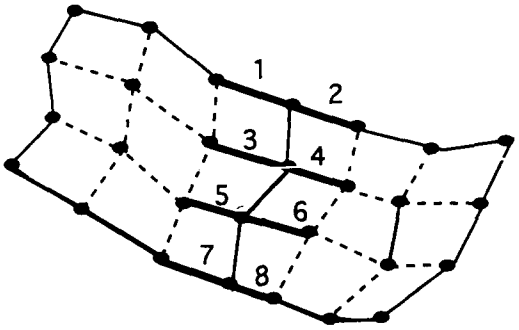

Figure 8. Colinearity requirements along edge of two adjoining Bezier patches.

This stringent requirement makes it infeasible to use the grid points generated in the first phase of the algorithm as the Bezier control points. The grid produced by a real β-sheet is far too irregular to be decomposed in the required manner. Instead we use each set of 4 grid points as the outer corner for a patch and then calculate 12 additional edge and interior points (Figure 7).

2. Points on either side of a patch boundary must be colinear. In Figure 8, this requirement means that segment 1 must be colinear with 2, 3 with 4, 5 with 6, and 7 with 8. This colinearity condition is most restrictive where four patches meet at a point. The four segments, A, B, C, and D in Figure 9 must each be linear. Thus, the surface enclosed by A, B, C, and D is a bilinear surface. The simplest surface that meets this criterion is a plane. A plane allows us to assign the vectors p1-p3 and p2-p4 using an approximation of the two roughly perpendicular strand derivatives, which is an easy and reasonable thing to do (p1, p2, p3, and p4 are the midpoints of the segments A, B, C, and D, respectively).
3. The ratio of segments lengths on one side of a patch boundary to that on the other must be constant. In Figure 8, this requirement means than len(s1)/len(s2) = len(s3)/len(s4) = len(s5)/len(s6) = len(s7)/len(s8). Where len(s*i*) represents the length of segment *i*, $1 \leq i \leq 8$. This length restriction is simply dealt with by forcing the lengths of edge segments to be equal throughout the control surface. This guarantees that ratios are 1:1. This has the undesirable effect of making the distances between interior control points unpredictable and uneven. This will be a minor effect, however, owing to the regularity of the α-carbon to α-carbon distances in β-sheet (always 3.8 Å along a strand, and only slightly greater between adjacent strands). By choosing the edge segment length carefully, we can produce a reasonable grid.

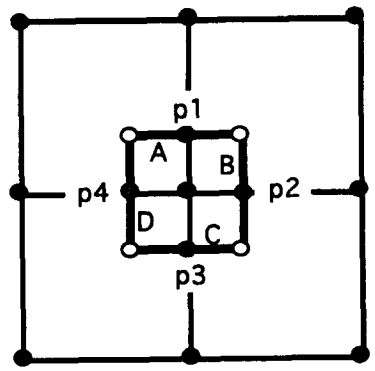

Figure 9. Colinearity requirements along edges of four adjoining Bezier patches.

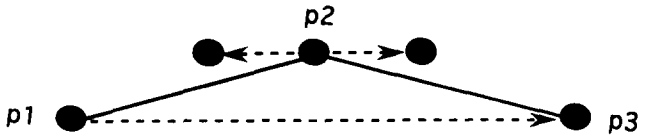

Figure 10. Use of finite-difference approximation to strand derivative for calculating intermediate points along strand.

We wanted to limit the use of nonlocal information when processing a tile to that along cross-strands or chain-strands. This turns out to be practical, and allows us to optimally produce all information required for the control grid in a single traversal of the strands, with each strand processed individually.

We produce intermediate control points along a strand by using a finite difference approximation to the derivative at each point to produce offset vectors. This is diagrammed in Figure 10. The vector from p1 to p3 approximates the derivative at p2. A vector in this direction of length 1.25 Å (approximately one-third of 3.8 Å) is added and subtracted from p2 to give two colinear points that will be used with p1 to span a patch edge. At the end of a strand, the vector from the first (or last) to the next (or preceding) strand point is used to approximate the derivative.

For each strand, the set of points produced by this procedure (including the original grid points) is stored in order along the strand in a new array. This new strand array is stored within an array of strands at exactly the same index as the original strand (referenced by *cross-strand-number* or *chain-strand-number*). Three times the index numbers (*cross-strand-index* or *chain-strand-index*) used to reference the original strand array serves to index the first of the four points in the new strand array required for a single patch.

With creation of intermediate strand points for both cross- and chain-strands, we have 12 of the 16 control points required for each patch. Because each interior point must be coplanar with the three edge points in its corner of the grid, we can produce this point using vector addition.

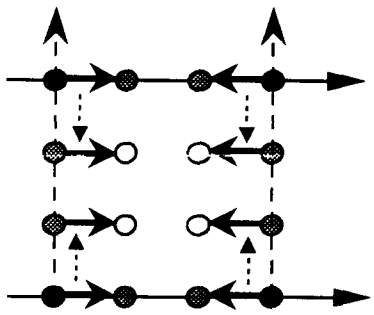

Figure 11. Creation of interior grid points using vector addition.

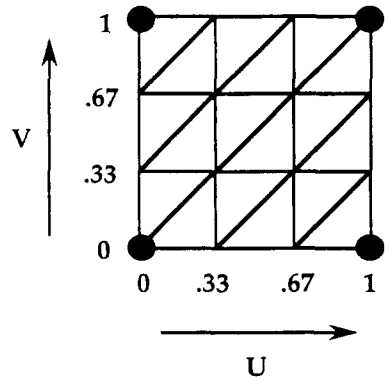

Figure 12. Parameter values for Bezier surface point generation, and surface triangle connection pattern.

Figure 11 diagrams how we can obtain the interior points by adding the vector (possibly with a sign change) on the chain-strand computed in the previous step to the appropriate intermediate cross-strand point.

With all 16 control points available, we are now ready to tile the patch. The Bezier surface routine (described in Ref. 8) is passed the 16 control points and creates a triangular tiling for a single patch. The routine creates triangle vertices based on a set of (u, v) parameter values. Each (u, v) value is used in conjunction with the control points to compute a single vertex (an $[x, y, z]$ value). The u and v values must each lie between 0 and 1. We elected to create a regular (u, v) mesh with 4 points along each of the u and v axes, for a total of 16 parameter values (the decision to use 16 values is arbitrary, and unrelated to the use of 16 control points, which is essential). Figure 12 shows the (u, v) values used and diagrams the connections between the (x, y, z) points produced to create 18 triangles.

The Bezier formulation can be used to compute surface normals as well as surface positions. We chose, instead, to use an existing routine that creates normals for a tiled surface by averaging the plane normals of all triangles coincident at each vertex. This makes it easy for the user to see the triangulated surface and to smooth it later.

3. ALGORITHM SUMMARY

Figure 13 diagrams the algorithm for a portion of β-sheet from the protein concanavalin A,[9] file 2cna from the Brookhaven Data Bank.[10] Figure 13a shows the main chain and hydrogen bonds for the protein. Figure 13b shows the grid points, and the cross-strands and chain-strands that connect them. Figure 13c shows the set of intermediate points produced along each strand. A single set of four interior points is shown at the corners of the white plane. Figure 13d shows the Bezier surface with the corner points (and grid) for each patch. Figure 13e shows the surface after triangle normals have been averaged. Narrow white cylinders have been added that follow the patch edges to indicate the chain-strand positions and directions. In general, strand-following splines that are computed using the α-carbon midpoints will closely match the tessellated surface (see Figures 17–19 and 21 for examples).

4. PROBLEMS WITH THE ALGORITHM

There are two main problems with the algorithm: (1) bifurcated hydrogen bonds, often found at β bulges, may cause irregularities in the grid, and (2) missing hydrogen bonds produce gaps in the sheet. These problems are dealt with in the same fashion—by providing for manual editing of the control grid.

4.1 Editing the control grid

Figure 14 shows the main chain with hydrogen bonds for a β-sheet region in carboxypeptidase A[11] (3cpa); a bifurcated bond causes an irregularity in the control grid. In back is the sheet produced by this grid; note a tear in the sheet in the region of the bifurcated bond (also note the presence of gaps at the edge of the sheet due to missing hydrogen bonds in Figure 13).

Although a set of heuristics for dealing with bifurcations could be added to the tiling algorithm, such a set is likely to be complicated and still not handle all cases properly. In addition, there are choices to be made (whether or not to fill in gaps; which bond of a bifurcation to use; whether to extend the sheet edges where nitrogen–oxygen distances were a bit too far to qualify for hydrogen-bonds). Researchers are likely to differ in opinion as to how to resolve these decisions. For this reason, we decided to leave the algorithm fairly simple, and allow the user to edit the control grid prior to tiling. However, after trying out alternative versions, a given user might settle on a consistent style for handling bulges and add such a rule to the algorithm for their own use.

Two edited versions of the control grid from Figure 14 are shown in Figure 15, along with the tilings that they produce. Note that the tiling has been regularized in the region of the bifurcation so that the gap has been filled. Note that the first editing choice accentuates the β bulge, whereas the second minimizes it.

Editing is performed using the interactive event mechanism within the VIEW system.[6] Three events are provided:

- "delete a connection"
- "add a chain-connection"
- "add a cross-connection"

Each event is initiated by pressing a separate keyboard key, which then prompts the user for selection of on-screen geometry. The deletion event requires selecting one of the strand connections. The two add-connection events require that the user select the two points to be connected. Selection is among a set of icons (in our implementation, octahedrons), one at each potential grid location(α-carbon midpoints). Each event processes the user input, updates the internal data structures appropriately, and updates the on-screen representation of the grid.

Although implementation of the three editing operators is conceptually simple, the implementation is complicated by the fact that several cases must be considered for each. The delete operator must allow for either deletion of segments at the ends of strands, or for deletion in the middle of the strand. In the latter case, two strands have been created where there was previously one. The strand array must be appropriately updated, and all residues in the strands must

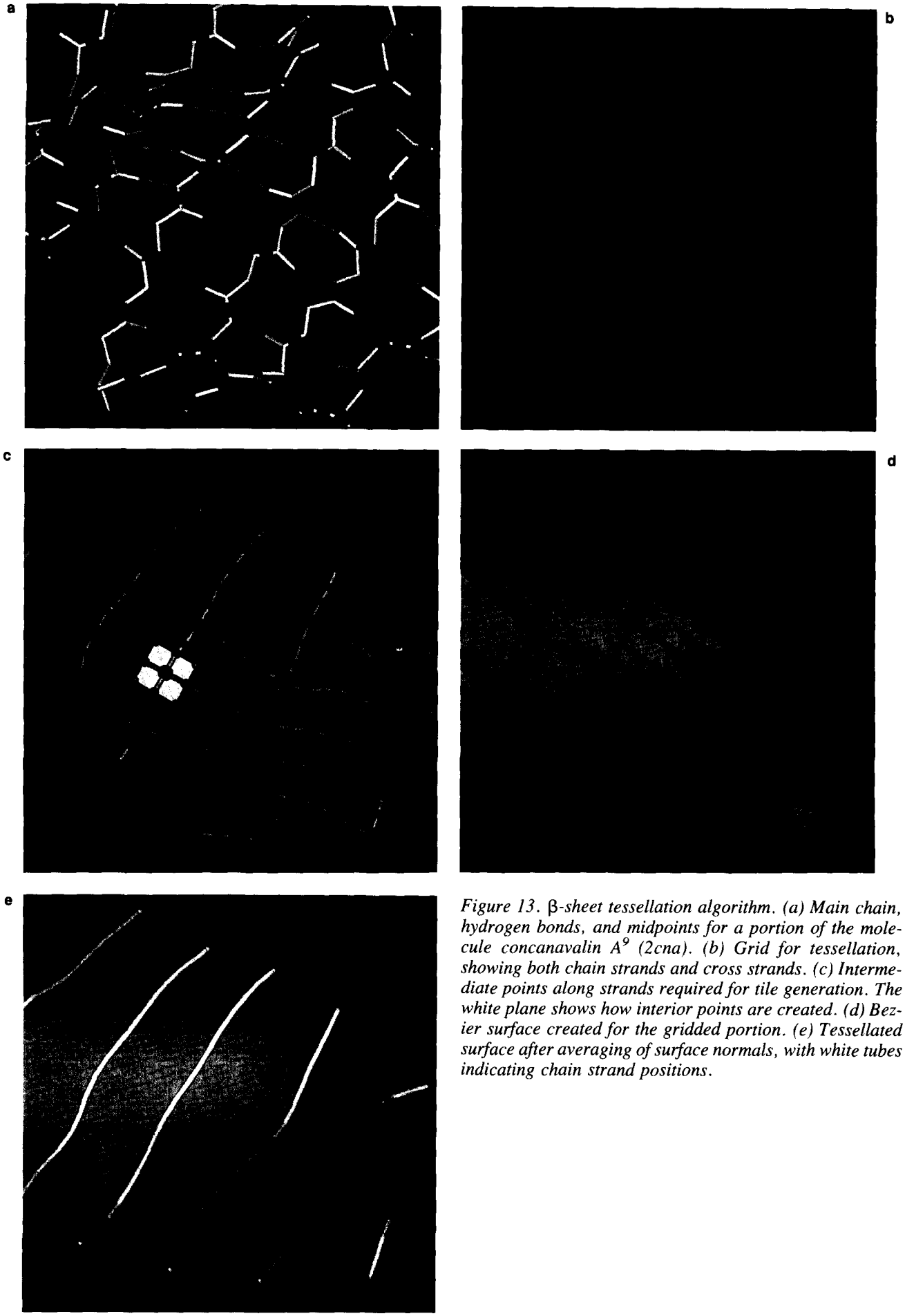

Figure 13. β-*sheet tessellation algorithm. (a) Main chain, hydrogen bonds, and midpoints for a portion of the molecule concanavalin A*[9] *(2cna). (b) Grid for tessellation, showing both chain strands and cross strands. (c) Intermediate points along strands required for tile generation. The white plane shows how interior points are created. (d) Bezier surface created for the gridded portion. (e) Tessellated surface after averaging of surface normals, with white tubes indicating chain strand positions.*

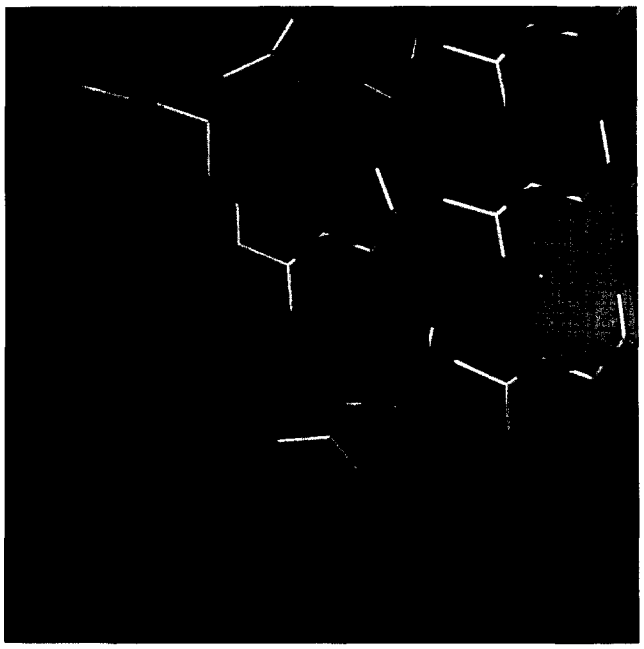

Figure 14. Grid irregularity created by a bifurcated bond. Irregularity in the grid, with one five-sided patch, resulting in a break in the tessellated surface. This figure is a composite, with the surface shown as if it were entirely behind the atoms, so that it does not hide any of them.

have their strand numbers and element numbers updated. Similarly, adding a segment may be in the form of extending an existing segment, joining two existing segments, or starting a new segment. Changes to the data structures differ for each of these cases.

Grid editing does not require use of an interactive graphics system such as VIEW. The algorithm could be implemented with a set of editing functions that would be specified with text commands. The user would run the algorithm once to create a representation of the grid with all strands and/or residues labeled. The user would determine which connections are to be deleted and/or added and then rerun the algorithm including a set of textual directives to make these changes to the grid.

4.2. Controlling the surface orientation

Because a single protein may contain several sheet regions, the orientation of the regions with respect to each other may not come out as desired (e.g., the user may wish the interior-facing side of all sheets to show a consistent color). We have devised a tool that allows the user to specify particular regions that are to have their orientation reversed. The user of this tool selects a single triangle from a region. This triangle is flipped (the ordering of the three vertices reversed, and the sign changed on each of the vertex normals), and then the tool ensures that all other triangles in the connected region have the same orientation.

The algorithm is a region-growing procedure using a breadth-first tree search (see Ref. 12) starting with the single selected triangle. Triangles with vertices common to those already checked are used to grow the region. As the

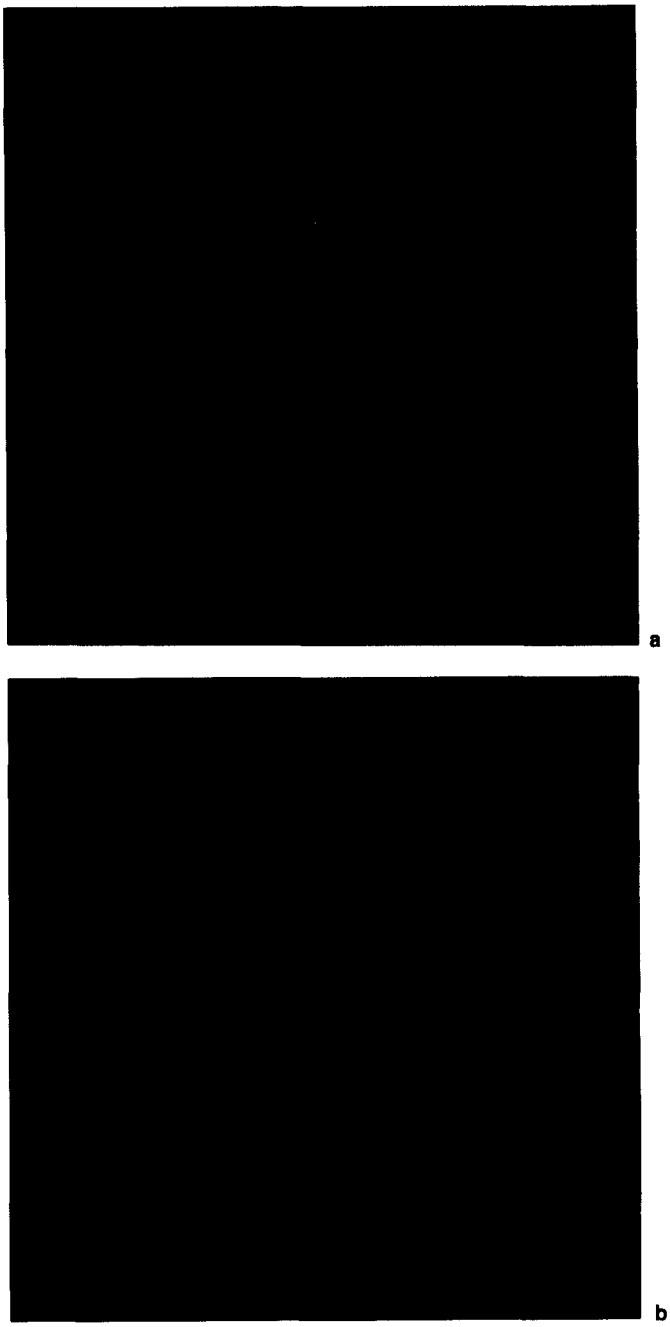

Figure 15. Effects of different user choices on editing of control grid. (a) First choice for edited grid, with the resulting surface; (b) second choice for edited grid, with a different resulting surface.

region grows, each new triangle is checked against an already oriented one. Plane normals are compared by taking a dot product. It the dot product is positive, the triangles are already aligned. If the dot product is negative, the triangles are antialigned, and the new triangle is flipped by reversing its vertex order. If the dot product is near zero, the triangles are oriented near 90 degrees, and the outcome is indeterminate. Owing to the local smoothness of the Bezier surface, we do not anticipate such a case ever arising, and have not provided for it. Color Plate 1A shows the β-sheets for the protein transthyretin[13] as produced by the tessellation

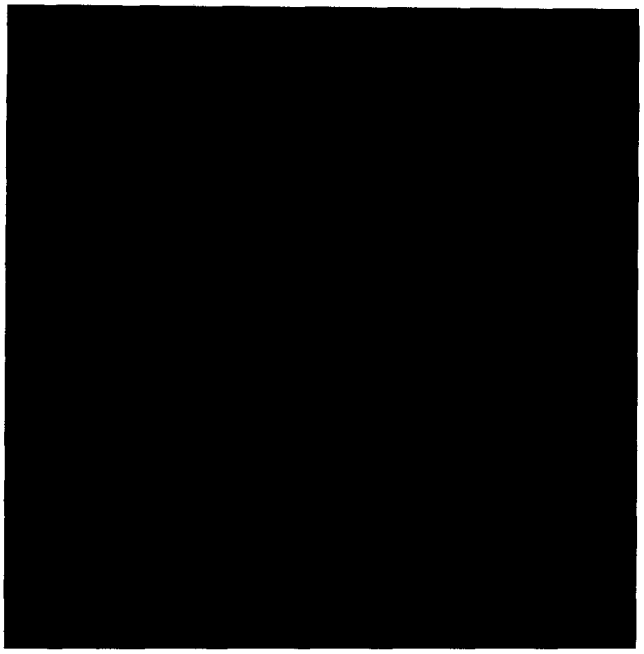

Figure 16. β bulge in a sheet region of satellite tobacco necrosis virus[16] (2stv); the resulting irregularity in the tessellated surface is clearly shown by shading and highlights from most views.

algorithm. Several of the sheet segments are improperly oriented. Color Plate 1B shows the same structure with two of the sheets reoriented using the region-flipping tool. The protein subunits now show the correct twofold and pseudo-twofold symmetries.

DISCUSSION

We have discovered this surface tiling algorithm to be useful for examining and presenting both global and local structures within proteins. In this section we show examples of each.

5.1. Global structure

These sheet tessellations allow clear presentation of β-sheet organization even in large subunits or in multisubunit proteins, in closed β-barrels as well as sheets, and they work well in combination with the cylinder representation for α-helices. Color Plate 2 shows the 394-residue subunit of α1-antitrypsin[14] (7api), with one side of the β-sheets in blue and the other side in green. The length and shape of the lower sheet and the strong curl of the upper sheet are evident, and there is no ambiguity about which strands are hydrogen-bonded, unlike a ribbon diagram.

Color Plates 3 and 4 show tessellated β-sheets in multisubunit proteins with different symmetries and organization: the dimer of dimers in transthyretin[13] with magenta and orange β-sheets, and the fivefold screw, almost an exact fivefold, of verotoxin[15] (1bov), with gold β-sheets and an inner ring of α-helices. In both cases, the β structure is continuous between subunits.

A feature that might add to this representation would be semitransparent renderings. Because VIEW does not support this, we have not created such images.

5.2. Local structure

Figure 16 shows one of the sheet regions in a subunit of satellite tobacco necrosis virus[16] (2stv). Note the dimple that appears at a β bulge (connections were added to the grid to tile the bulge). This tiling algorithm allows one to show the smooth overall shape of a β-sheet and also to preserve information about local irregularities by using surface cues such as shading or silhouettes.

In other cases, it may be the strand pairings rather than the strands that are of interest in a β-sheet, as in the Greek key structure of γ-crystallin[17] (1gcr) in Color Plate 5 (made for Ref. 18), or for distinguishing antiparallel vs. parallel strand pairs in a mixed sheet.

6. CONCLUSION

We have presented an algorithm for creating smooth tiling of β-sheet surfaces. When combined with a manual editing capability for extending, deleting, and redefining portions of the surface, this is a powerful new way of representing secondary structure in proteins.

The algorithm has been applied to a number of proteins, and has been found useful for exploring local structural detail of β-sheet, as well as for showing global organization of protein secondary, tertiary, and quaternary structure.

7. ACKNOWLEDGMENTS

Amitabh Varshney provided a great deal of assistance in devising the patch tiling procedure as well as reviewing the manuscript. Jens Hemmingsen helped devise the algorithm for identifying β-sheet. Kim Gernert provided a number of helpful comments, in particular suggesting that we do the tiling within a grid square. Fred Brooks provided invaluable support during development of the VIEW system. This work was funded by the Biotechnology Research Program, National Center for Research Resources, NIH, Grant RR02170 (UNC) and NIH Grant GM15000 (Duke).

REFERENCES

1 Richardson, J.S. Schematic drawings of protein structures. *Methods Enzymol.* 1985, **155b,** 359–380
2 Carson, M. and Bugg, C.E. Algorithm for ribbon models of proteins. *J. Mol. Graphics* 1986, **4,** 121–122
3 Kraulis, P. MOLSCRIPT: A program to produce both detailed and schematic plots of protein structures. *J. Appl. Crystallogr.* 1991, **24,** 946–950
4 Dickerson, R.E. and Geis, I. *Structure and Action of Proteins.* Harper, New York, 1969
5 Salemme, F.R. Structural properties of protein β-sheets. *Prog. Biophys. Mol. Biol.* 1983, **42,** 95–133
6 Bergman, L.D., Richardson, J.S., Richardson, D.C., and Brooks, F.P. VIEW—an exploratory molecular visualization system with user-definable interaction sequences. *Comput. Graphics* 1993, **27(4),** 117–126. 1993 SIGGRAPH Proceedings

7 Kabsch, W. and Sander, C. Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers* 1983, **22,** 2577–2637
8 Farin, G. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, San Diego, CA, 1990
9 Reeke, G.N., Jr., Becker, J.W., and Edelman, G.M. The covalent and three-dimensional structure of concanavalin A. IV. Atomic coordinates, hydrogen bonding, and quaternary structure. *J. Biol. Chem.* 1975, **250,** 1525–1547
10 Bernstein, F., Koetzle, T., Williams, G., Meyer, E., Jr., Brice, M., Rodgers, J., Kennard, O., Shimanouchi, T., and Tasumi, M. The Protein Data Bank: A computer-based archival file for macromolecular structures. *J. Mol. Biol.* 1977, **112,** 535–542
11 Rees, D., Lewis, M., and Lipscomb, W. Refined structure of carboxypeptidase A at 1.54Å resolution. *J. Mol. Biol.* 1983, **168,** 367–387
12 Aho, A.V., Hopcroft, J.E., and Ullman, J.D. *The Design and Analysis of Computer Algorithms*, Addison Wesley, Reading, MA, 1974
13 Vivian Cody, personal communication
14 Engh, R., Loebermann, H., Schneider, M., Weigand, G., Huber, R., and Laurell, C.-B. The S variant of human α1-antitrypsin, structure and implications for function and metabolism. *Protein Eng.* 1989, **2,** 407–415
15 Stein, P.E., Boodhoo, A., Tyrrell, G.J., Brunton, J.L., and Read, R.J. Crystal structure of the cell-binding B oligomer of verotoxin-1 from *E. coli*. *Nature* (*London*) 1992, **355,** 748–750
16 Jones, T.A. and Liljas, L.L. Structure of satellite tobacco necrosis virus after refinement at 2.5 Å resolution. *J. Mol. Biol.* 1984, **177,** 735–767
17 Blundell, T., Lindley, P., Miller, L., Moss, D., Slingsby, C., Tickle, I., Turnell, B., and Wistow, G. The molecular structure and stability of the eye lens: X-ray analysis of γ-crystallin II. *Nature* (*London*) 1981, **289,** 771–777
18 Hemmingsen, J.M., Gernert, K.M., Richardson, J.S., and Richardson, D.C. The tyrosine corner: A feature of most Greek key β-barrel structures. *Protein Sci.* (submitted).