A real-time malleable molecular surface

Teri E. Klein, Conrad C. Huang, Eric F. Pettersen, Gregory S. Couch, Thomas E. Ferrin and Robert Langridge

Computer Graphics Laboratory, University of California, San Francisco, CA, USA

We describe a method for generating a molecular surface using a parametric patch representation. Unlike previous methods, this algorithm generates a parametric patch surface which is smooth and G^1 continuous and manipulable in real-time. Crucial to our approach is the creation of a net of approximately equilateral triangles from which we generate the control points used as the basis for describing the surface. We present in detail the method used for generating the triangular net and accompanying control points, along with examples of the resulting surfaces.

Keywords: triangular net generation, real-time interactive surface

BACKGROUND

The use of surfaces in molecular modeling and drug design is well documented.[1-3,7-10] The two most common surfaces for these applications are solvent accessible[3,5,11] and van der Waals[4,6] representations, although both of these types of surfaces require knowledge of the atomic coordinates of the structure(s) under investigation. In contrast, in drug design studies the detailed coordinates of the receptor site are often unknown. A combination of molecular modeling and quantitative structure–activity relationships (QSAR) provides a powerful tool in better understanding ligand–receptor interactions in such circumstances,[10] and does not depend upon knowing the exact three-dimensional structure of the receptor. It is still desirable, however, to have a graphical representation of the receptor site. We have developed a system, Knowledge Assisted Receptor Mapping Analysis (KARMA),[12,13] that incorporates quantitative structure–activity relationships, conformational analysis and three-dimensional interactive graphics for use in drug design studies. A major component of the KARMA system is the representation of a molecular surface that is malleable and can be deformed in real-time in response to a set of rules partially based on QSAR calculations and conformational analysis. These molecular surfaces do not represent molecular volumes the way that solvent accessible and van der Waals surfaces do, but rather represent the potential shape of a receptor site. Since KARMA uses a knowledge rule-based iterative approach in its attempt to determine the chemical and geometric characteristics of a receptor site, it is imperative that KARMA surfaces be dynamic in response to interactive usage of the system by the scientist user. Once a receptor site has been characterized, geometric fitting techniques can be used to search available databases for additional ligands potentially capable of binding with the receptor.[14,15]

We describe here the method used for generating a KARMA surface that is malleable in real-time, has the additional desirable properties of color and density visual cues, and can be efficiently manipulated in an interactive modeling environment.

Our algorithm provides for a dynamically manipulable surface based on a set of *control points* and *surface patches*. A user may interactively select a control point on one of the surface patches with a pointing device and translate the control point in the current view's reference frame. After the control point data has been modified, the graphics interface recalculates and redraws the patches of the surface model in real-time based on the new data.

SURFACE BASICS

A molecular surface model can be described as a G^1 continuous surface[16] which envelopes a set of intersecting spheres. The surface must be closed and generated about a continuous volume. Required properties include smoothness of the surface, *i.e.*, continuity of position and tangent to the surface. Continuity of position and tangent may be obtained by using either parametric or closed-form equations for the curves.[17] However, surfaces enclosing volumes will always have vertical tangent planes with respect to any coordinate system; this causes problems with closed-form representation when the derivative goes to infinity (see Figure 1). Thus, we chose to use parametric equations to represent the curves.

In general, a surface describing a union of spherical volumes is based on a set of points, P, defined as

$$P = \bigcup_i P_i - \bigcup_{i,j} I_{ij} \qquad (1)$$

where P_i is the set of points distributed over a sphere cor-

Address reprint requests to Dr. Klein at the Computer Graphics Laboratory, University of California, San Francisco, CA 94143, USA.
Received 1 June 1988; accepted 4 August 1989

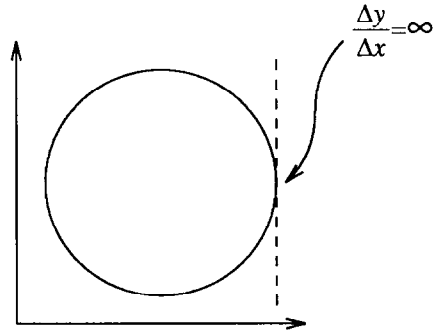

Figure 1. Vertical tangent of an enclosing surface has infinite derivative when expressed in closed form representation

responding to atom i, and I_{ij} is the set of points in P_i which fall inside atom j. The density of points on spheres is usually determined by the user. These points can serve as control vertices for parametric patches that model the surface. If the density is high, a *large* number of patches each with small area are generated; to address each patch at a high density would be time-consuming and difficult at best. If the density of points is low, the patches become too large and do not yield enough detailed information about the surface model.

PREVIOUS APPROACHES

In the past, we attempted to use parametric bicubic patches as the mathematical basis for the molecular surface models.[12,13,18] Advantages of parametric bicubic surfaces include continuity of position, slope, and curvature at the points where two patches meet. All points on a bicubic surface are defined by cubic equations of two parameters s and t, where s and t vary from 0 to 1. The equation for $x(s, t)$ is

$$x(s,t) = a_{11}s^3t^3 + a_{12}s^3t^2 + a_{13}s^3t + a_{14}s^3 + a_{21}s^2t^3 + a_{22}s^2t^2 + a_{23}s^2t + a_{24}s^2 + a_{31}st^3 + a_{32}st^2 + a_{33}st + a_{34}s + a_{41}t^3 + a_{42}t^2 + a_{43}t + a_{44} \quad (2)$$

Equations for y and z are similar.[19] Overlapping sets of control points allow for the joining of patches. Sixteen points define a quadrilateral bicubic patch. To determine which points define which patches, a polygonal triangular net is initially calculated. The internal edge of two triangles is dropped to form a quadrilateral. Nine quadrilaterals define a single patch. Cardinal patches[20] can then be used for surface generation because these patches are interpolated through the derived control points.

Unfortunately, the generation of a bicubic patch surface based on Cardinal patches is not continuous between patches. Graphically, this results in rosette-like patterns at the boundaries of adjoining patches. This problem is attributed to adjacent patches not sharing the same control points along the shared edge; hence, the common spline edges between adjacent patches are not identical. Since the control points cannot, in general, be made to fit a rectangular grid, a control point may belong to three or more patches. Regardless of the basis matrix used (*e.g.*, Cardinal or Bézier), this surface representation yields discontinuity between the patches.

A NEW APPROACH

The Cardinal patch representation described in the previous section is severely limited by its conditional need for rectangular regularity of its control points in order to generate a continuous fit of surface patches. Our present algorithm provides for the generation of control points in such a fashion that we are able to generate a net of approximately equilateral triangles about a given surface.† These equilateral triangles then become the basis for triangular parametric patches. Chiyokura has described the mathematics for using nonpolynomial Gregory patches for solids modeling with free form surfaces.[21,22] Séquin and his students have recently described a method for the interpolation of Gregory patches between cubic boundaries using Chiyokura's equations.[23–26] In their program UNICUBIX,[26] an object is described by vertices positioned in space and by edges and surface patches stretched between these points. We have applied this technique to the field of molecular modeling.

Our method for smooth and continuous surface generation uses cubic Bézier curves for the boundaries between patches. Quartic (4th order) Gregory patches are joined with tangent-plane continuity to provide a smooth interpolated surface. Gregory patches are used as opposed to Bézier patches because Gregory patches provide twice as many interior control points[20] and thereby guarantee tangent-plane continuity between patches. To determine which points define which patches, a triangular net is calculated from the union of atomic coordinates and radii of the ligand(s) which are found to bind with the receptor site under study; the coordinate data may come from either X-ray crystallographic data or conformational analysis. The triangular net is then converted to a parametric representation in three steps as described by Shirman.[25] The first step calculates the vertex normals based on the vertices of the triangles and the angles associated with these vertices. The second step involves the determination of Bézier control points for the curved edges. Lastly, the internal control points used for the Gregory patches are determined.

TRIANGULAR NET GENERATION

The first step in computing the surface for a set of atoms is to approximate a surface with a set of adjacent triangles that enclose the volume of the atoms. Unlike previous algorithms based on Cardinal splines,[18] where the control points are distributed in approximate uniformity over a sphere and then merged, the control points for the algorithm presented here are distributed in approximate uniformity over the entire surface. This allows the triangles of the polygonal net to better approximate equilateral triangles, yielding a surface with greater regularity.

†Appendix 1 contains a procedural description of the algorithm

Our algorithm consists of the following steps: points are generated on the surface based on a desired distance between points. Edges of the desired length are created and polygons are formed from these edges. These polygons consist of triangles and higher order polygons. The nontriangular polygons are then subdivided into triangles by adding new points and new edges, yielding a triangulated surface with approximately equilateral triangles.

The generation of surface points is based on a desired distance between the surface points, which we refer to as the *target distance, t*, in our discussion below. Actual distance between surface points is within a tolerance Δt. Initially, points are generated on a sphere that are approximately one-tenth the target distance apart; we refer to this distance as the *minimum distance, l*. The sphere is then divided into layers. To compute the number of layers, it is necessary to first compute $\Delta\phi$ which is approximated by

$$\Delta\phi \approx \left(\frac{\textit{minimum distance}}{\textit{radius}}\right) \tag{3}$$

The number of layers can then be obtained by

$$\textit{number of layers} = \left\lfloor \frac{\pi}{\Delta\phi} \right\rfloor \tag{4}$$

The specific angles and distances used in this algorithm are diagramed in Figure 2. After the number of layers has been calculated, the number of points for each layer is determined. The layers are obtained by stepping through the colatitude angle with a $\Delta\phi$ step size. The number of points at each ϕ is found by first computing the circumference of the circle at that colatitude and then dividing that circumference by the minimum distance. Thus

$$\textit{number of points for layer } i = \left\lfloor \frac{\textit{circumference}}{\textit{minimum distance}} \right\rfloor \tag{5}$$

To obtain all the points on that layer it is necessary to calculate $\Delta\theta$ which is found by

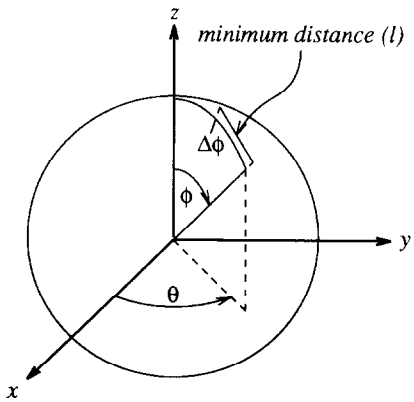

Figure 2. Potential control points are positioned on the surface of the sphere as determined by ϕ and θ. ϕ determines the current layer and is incremented by $\Delta\phi$. $\Delta\phi$ is chosen so that the distance between layers is approximately l

$$\Delta\theta = \frac{2\pi}{\textit{number of points for layer } i} \tag{6}$$

All points on the layer are then obtained by stepping θ from 0 through 2π. These points are *not* the control points, but rather potential control points. Most of the points from this calculation are discarded as described below.

Having obtained a sphere described by a set of points, Equation (1) is used. Additionally, when constructing the set of potential control points, there are certain points which need to be eliminated even though they do not fall within other spheres. An example of such a point is *O* in Figure 3. Point *O* causes a problem when trying to connect the potential control points into a triangular surface network. If we have already formed the valid surface *WXYZ* which connects the two spheres, it is no longer legitimately possible to connect point *O* to the surface. If not connected, then *O* is isolated and therefore not a viable surface control point. One can visualize this problem in three dimensions as having a connecting surface between the two spheres where there are isolated control points caught "below" the surface. This problem can be alleviated by extending the effective clipping radius of one sphere against the other so that there are no control points deep enough in the crevice between the spheres to get buried beneath a connecting surface. Potential control points are generated using the actual radius but clipped against the effective radius.

Figure 4 illustrates the computation of the effective clipping radius by repeatedly applying the law of cosines. In Figure 4, the radii (r_1, r_2), target distance (t), and the distance between sphere centers (d) are known. The following equations can then be used to calculate the effective radius, r_2'

$$s = \sqrt{r_1^2 - \frac{t^2}{4}} + \frac{\sqrt{3}}{2}t \tag{7}$$

$$t^2 = r_1^2 + s^2 - 2r_1 s\cos\beta \tag{8}$$

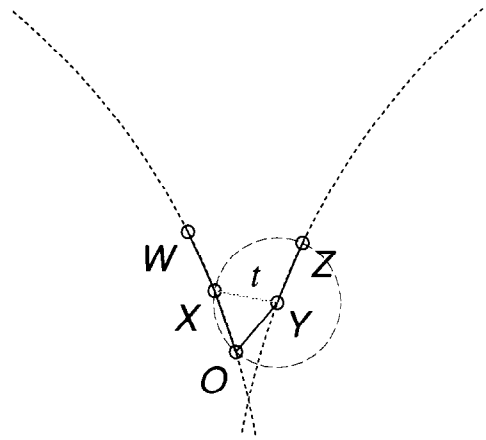

Figure 3. When forming a triangular network with edges approximately of length t, WXOYZ and WXYZ are both possible connectivities. WXYZ has the problem of "isolating" point O. Worse, the algorithm may try to form ZYXO or WXYO, leaving an end that cannot be connected. To avoid this, the clipping radii of the spheres is increased so that points deep in the crevice (such as O) are eliminated

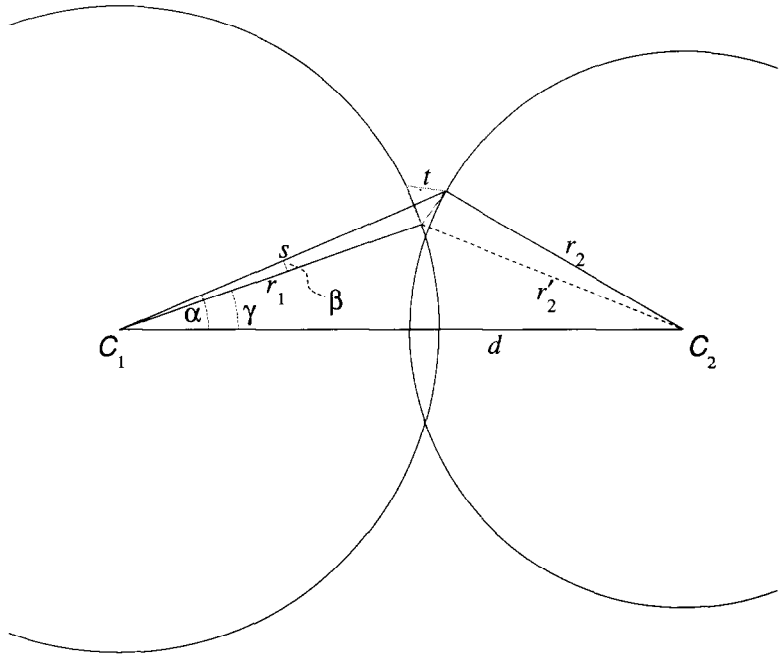

Figure 4. The effective clipping radius r_2' is computed so that the gray shaded equilateral triangle of side t has its lowest vertex clipped away. This prevents a surface that connects the spheres from being able to extend into the crevice after crossing the gap between the spheres (see Figure 3)

$$r_2^2 = s^2 + d^2 - 2sd\cos\alpha \tag{9}$$

$$\gamma = \alpha - \beta \tag{10}$$

$$r_2'^2 = r_1^2 + d^2 - 2r_1 d\cos\gamma \tag{11}$$

After having computed all effective radii, we can then apply Equation (1). To compute the set of potential control points, a band of acceptance is used to classify surface points as seen in Figure 5.

All points are marked with a label of *live, dead, potential,* or *saved* and each point has a counter associated with it. Initially, all points are marked as *live*. One of the points is chosen arbitrarily (point *J* in Figure 5), and all points which are closer than the target distance less the tolerance ($t - \Delta t$) are marked as *dead*. Any point that is not marked as *dead* and is less than $t + \Delta t$ distance is marked as *potential*. Each time a point is marked *potential,* the counter for that point is incremented. Additionally, associated with each such point is how close that point is to the actual target distance t. After all potential points have been examined, the next point saved is that point that has the largest value in its counter. If more than one point exists with the same counter value, the point that is nearest t in value is chosen and saved. Points can only go from *live* → *potential, live* → *dead, potential* → *dead,* or *potential* → *saved*. This procedure is repeated until there are no more points marked as *live* or *potential*. The list of all points marked *saved* then forms an approximately uniformly distributed set of points for the entire surface. These points are a subset of the control points for the surface model.

After obtaining the set of points that describes the outline of the surface model, the next step in generating the triangular net is to form the *obvious edges* if the edge length,

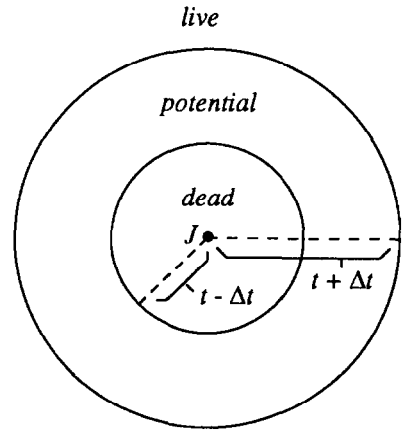

Figure 5. The selection of J as a control point eliminates all potential control points within a radius of $t - \Delta t$ from further consideration as control points. Points whose distance from J is between $t - \Delta t$ and $t + \Delta t$ are more likely to be selected as control points. Points beyond $t + \Delta t$ are unaffected

d, falls in the interval formed by the target distance and the minimum distance

$$(t - l) < d < (t + l)$$

Once the obvious edges have been found, the *obvious triangles,* sets of three obvious edges that have exactly three distinctive vertices among them, are formed. For every edge, all points adjacent to one endpoint (E_1) are checked. If an adjacent point (A) is also adjacent to the other endpoint (E_2),

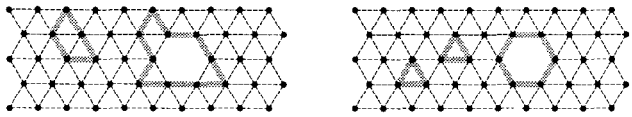

Figure 6. In the left hand diagram, the gray shaded polygons are noncanonical *because they have edges crossing their interiors. The right hand diagram demonstrates canonical polygons with no interior edges*

then a triangle is formed with vertices A, E_1, and E_2. Those points which are neighbors to both endpoints are used to form the obvious triangles. The edges have an associated counter which is incremented every time an edge is used. Edges that are used twice are no longer available to be used in another polygon.

After the obvious triangles are found, higher order canonical polygons are determined. Canonical polygons are those polygons which cannot be decomposed into smaller polygons (see Figure 6).

To find the canonical polygons, a depth first search is performed along the live edges (those edges that are not part of two polygons). An initial live edge is chosen and a path from one of its endpoints (forming one vertex of the polygon) to the other endpoint is found. The direct connection between the two endpoints is discarded. Paths are only accepted if they describe canonical polygons.

A multistep procedure is used to determine whether a polygon is canonical. Initially the points and edges for the given polygon are labeled. In the next step we chose a point not on the polygon; this point is also labeled and a depth first search is done from this starting point. All neighboring points and edges which are not already labeled are visited and labeled. This step is continued until all reachable points have been visited. If unlabeled edges remain, then the polygon is not canonical.

An aspirative search technique[27,28] is used to search for potential polygons. An aspirative search sets a limit to the search depth N from the outset and will not search beyond N. If a solution is found, the search is complete. If a solution is not found, the search depth is increased. For example, if an initial depth value of 7 is chosen and if a polygon cannot be found within that maximum depth, the depth value is incremented by that same amount ($7 + 7 = 14$) and the search is repeated. A depth value of 7 is chosen because most polygons have 7 or fewer sides due to our method of polygon generation. Termination of these searches occurs when there are no more live edges or the number of live edges left is less than the current aspirative search depth value. The advantage of using the aspirative search technique is that it quickly finds simple polygons and eliminates those edges from consideration thus limiting search space and speeding up future searches.

Lastly, after the canonical polygons have been found, it is necessary to add new edges to subdivide them into triangles. The initial step is to make all polygons convex. This is accomplished by looking at each polygon and its points, and eliminating concavities by forming new edges. A point X is concave if there exists a point Y on the polygon such that the distance between Y and X is less than the distance from the neighbors of X to Y as illustrated in Figure 7.

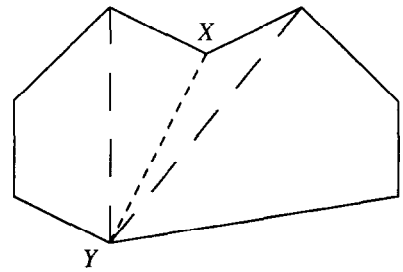

Figure 7. Point X is concave relative to point Y because the distance XY is less than the distances from Y to X's neighbors

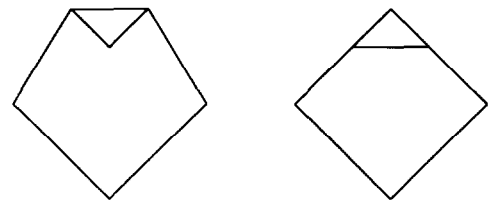

Figure 8. These two-dimensional polygons have the same connectivity, yet the larger subpolygon in the left hand figure is concave, while in the right hand figure it is convex

Though not true in general, this criterion is sufficient in our case because the edges of the polygon are approximately equal in length. Concavity cannot be determined from connectivity alone and must be determined from numerical values as seen in Figure 8. To eliminate the concave point, a new edge is formed between point X and the point closest to point X which satisfies the concavity criterion. This process yields two polygons which then must be recursively tested for convexity, as breaking a concave polygon does not guarantee two convex polygons.

After all the polygons have been determined to be convex, new interior edges are added to form triangles. The target distance constraint is relaxed such that any distance from two-thirds to four-thirds the target distance is acceptable. Remaining polygons which are not triangles are split into triangles by adding a new point which forms a triangle with each edge. The new point is added by finding a point previously marked as *dead* which is nearest to the center of mass of the vertices of the polygon, and then using this point to form new triangles. The addition of this point yields edges which are typically shorter than the target distance.

These triangles, and their vertices which serve as control points, are used for the Gregory quartic patches following the method of Shirman.[25]

PERFORMANCE

Table 1 shows the CPU time required to generate the Gregory patch control points and then graphically generate and display the patches themselves. All timings are from a Silicon Graphics (SGI) IRIS model 4D/70GT workstation, which

Table 1. CPU time required to generate and display molecular surfaces

Number of atoms	Target distance (Å)	Number of control points	Number of calc. surface patches	Generation time (sec)	Display rate (frames/sec)
1	1.0	46	88	1.8	21.5
	1.3	26	48	1.3	27.5
6	1.0	114	224	8.1	12.4
	1.3	68	132	6.2	18.5
10	1.0	184	364	18.4	8.4
	1.3	103	202	12.9	13.8
14	1.0	246	488	30.2	7.2
	1.3	143	282	22.2	10.6
19	1.0	309	614	46.6	5.0
	1.3	178	352	32.9	8.6
26	1.0	305	606	47.7	5.0
	1.3	178	352	35.3	8.5

utilizes a 12.5MHz Mips Computer Systems R2000 Reduced Instruction Set Computer (RISC) chipset. Control point generation times were obtained in a straight forward manner with the UNIX "time" command, while the display update rate was obtained by measuring the CPU time (user + system) required during a typical interactive session and then dividing the number of frames generated during this interval by the time. We expect to gain a significant speedup in all timings when we move to an IRIS 4D/80GT workstation (16.7MHz clock) in the near future. All algorithms are implemented in the C programming language. The Gregory patches were rendered using a single light source in the lighting model, with a single color, and using the alpha blending feature of the IRIS hardware. To make the displayed surface appear smoother, we actually display nine patches for each surface patch we calculate.

The 5.0 fps worse case display rate shown in Table 1 represents the marginal limit of interactive performance (using our current hardware). Display rates of 15–20 fps result in good interactive responsiveness.

RESULTS

A complete Gregory patch surface for the drug methotrexate is seen rendered as a net in Color Plate 1 and illustrates the use of multiple colors on a patch surface. These colors may be used to represent different chemical properties. In this instance, the electrostatic potential of methotrexate is illustrated. Currently, five styles of solid rendering are available: solid, thatched, half-tone, cross-hatched and translucent. For all solid rendering, a light source may be arbitrarily positioned to define the direction from which light rays appear. Translucent rendering is seen in Color Plate 2, with the surface of methotrexate again color-coded based on electrostatic potential.

Color Plate 3 shows the control points (displayed as tetrahedron) which allow the KARMA rule system or a scientist user to interactively choose an area of the surface to manipulate in real-time. To illustrate the dynamic nature of this surface, Color Plate 4 shows a control point which has been moved but not yet saved. The user can easily distinguish which part of the surface is moving since that area is displayed by a dynamic triangular net relative to the original surface.

CONCLUSION

The generation of a real-time malleable molecular surface representing a potential receptor site and therefore useful in drug design studies has been described. The design criteria for the surface includes: (1) the surface is closed and generated about a continuous volume; (2) the surface is smooth and continuous; and (3) the surface is manipulable in real-time. Initial attempts using a Cardinal patch representation did not result in a surface which was everywhere continuous. Generation of an equilateral triangular net and accompanying control points about the surface have allowed for the creation of a smooth and continuous parametric patch surface based on Gregory patches. Gregory patch surfaces successfully and efficiently meet the above design criteria.

ACKNOWLEDGEMENTS

This work was supported by National Institutes of Health Division of Research Resources (RR-1081).

REFERENCES

1 Porter, T. K. Spherical shading. *Computer Graphics* (Siggraph '78 Conf. Proc.) 1978, **12**, 282
2 Porter, T. K. The shaded surface display of large molecules. *Computer Graphics* (Siggraph '79 Conf. Proc.) 1979, **13**, 234
3 Langridge, R., Ferrin, T. E., Kuntz, I. D. and Connolly, M. L. Real-time color graphics in studies of molecular interactions. *Science* 1981, **211**, 611
4 Bash, P. A., Pattabiraman, N., Huang, C., Ferrin, T. E. and Langridge, R. Van der Waals surfaced in molecular modeling: Implementation with real-time graphics. *Science* 1983, **222**, 1325
5 Connolly, M. L. Solvent-accessible surfaces of proteins and nucleic acids. *Science* 1983, **221**, 709

6 Pearl, L. H. and Honegger, A. Generation of molecular surfaces for graphic display. *J. Mol. Graphics* 1983, **1**, 9 and 1979, **13**, 234
7 Humblet, C. and Marshall, G. E. Three-dimensional computer modeling as an aid to drug design. *Drug Des. Res.* 1981, **1**, 409
8 Blaney, J. M., Jorgensen, E. C., Connolly, M. L., Ferrin, T. E., Langridge, R., Oatley, S. J., Burridge, J. M. and Blake, C. C. F. Computer grahics in drug design: Molecular modeling of thyroid hormone–prealbumin interactions. *J. Med. Chem.* 1982, **25**, 785
9 Max, N. L. Computer representations of molecular surfaces. *IEEE Computer Graphics and Applications* 1983, **3**, 21
10 Hansch, C. and Klein, T. E. Molecular graphics and QSAR in the study of enzyme–ligand interactions. On the definition of bioreceptors. *Acc. Chem. Res.* 1986, **19**, 392
11 Richards, F. M. Areas, volumes, packing and protein structure. *Ann. Rev. Biophys. Bioengng.*, 1977, **6**, 151
12 Klein, T. E., Huang, C. C., Ferrin, T. E., Langridge, R. and Hansch, C. Computer-assisted drug receptor mapping analysis. In *Artificial Intelligence in Chemistry*. (T. H. Pierce and B. A. Hohne, Eds.) ACS Symposium Series 306, 147–158, 1986
13 Klein, T. E., Kneller, D., Huang, C., Ferrin, T. E. and Langridge, R. Computer graphics and artificial intelligence in drug design and protein engineering. *J. Mol. Graphics* 1985, **3**, 111–113
14 Kuntz, I. D., Blaney J. M., Oatley, S., Langridge, R. and Ferrin, T. E. A geometric approach to macromolecule–ligand interactions. *J. Mol. Biol.* 1982, **161**, 269–288
15 DesJarlais, R. L., Sheridan, R. P., Seibel, G. L., Dixon, J. S., Kuntz, I. D. and Venkataraghavan, R., Using complementary shape as an initial screen for the design of inhibitors at an active site of known three-dimensional structure. *J. Med. Chem.* 1988, **31**, 722–729
16 Bartels, R. H., Beatty, J. C. and Barsky, B. A. *An Introduction to Splines for use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann Publishers, Inc., USA, 1987
17 Mortenson, M. E. *Geometric Modeling*. John Wiley & Sons, USA, 1985
18 Klein, T. E. KARMA: A knowledge-based system for receptor mapping. Ph.D. Thesis, Department of Medical Information Sciences, University of California, San Francisco, CA, 1987
19 Foley, J. D. and Van Dam, A. *Fundamentals of Interactive Computer Graphics*. Addison–Wesley Publishing Company, USA, 1982
20 Clark, J. Parametric curves, surfaces, and volumes in computer graphics and computer aided geometric design. Technical Report No. 221, Computer Systems Laboratory, Stanford University, 1981
21 Chiyokura, H. and Kimura, F. Design of solids with free-form surfaces. *Computer Graphics* (Siggraph '83 Conf. Proc.) 1983, **17**, 289
22 Chiyokura, H. Localized surface interpolation method for irregular meshes. *Proc. Computer Graphics Conf.*, Tokyo, 1986
23 Gregory, J. A. *Computer Aided Geometric Design*. (Barnhill, R. E. and Riesenfeld, R. F., Eds.) Academic Press, New York, 1974, p. 71
24 Longhi, L. Interpolating patches between cubic boundaries. Master's Project Report, Computer Science Division, University of California, Berkeley, CA, 1985
25 Shirman, L. Symmetric interpolation of triangular and quadrilateral patches between cubic boundaries. Master's Project Report, Computer Science Division, University of California, Berkeley, CA, 1986
26 Séquin, C. Procedural spline interpolation UNICUBIX. *Proc. Third USENIX Computer Graphics Workshop*, 1986, 63
27 Brudno, A. L. *Problems of Cybernetics 10*. Pergamon Press, UK, 1963, p. 141
28 Marsland, T. A. A review of game-tree pruning. *Int. Comp. Chess Assoc. Journal* 1986, **9**, 3

APPENDIX 1

Overall strategy:

Given: A set of sphere centers and radii
Want: A set of connected triangles covering *global surface*

Generate prototype sets of points on sphere surfaces for all different radii
Generate *dense set* of points on *global surface* and eliminate intersections
Select initial set of triangle vertices from *dense set*
Make triangles from initial set of vertices and possibly other points from *dense set*

Generate prototype points:

Given: Sphere center (**x**) and radius (r)
Minimum distance between points (d)
Want: Set of points densely covering sphere surface

Divide sphere into layers at regular intervals
Assuming direct distance between points is approximately equal to the arc length between points,
angular interval = d/r
total layers = π/angular interval
Generate points on each layer based on circumference of layer
ϕ = layer# $\times$ π/(total layers $-$ 1)
radius = $r \sin\phi$
circumference = 2π radius
#points = circumference/d
The points are uniformly distributed over the circumference
Merge points from layers

Generate dense set of points:

Given: A set of sphere centers and radii
A set of prototype spheres
Want: Dense set of points on global surface

For each sphere center (s)
Select prototype sphere with same radius
Translate prototype sphere center to match real sphere center

For each point on sphere
For each real sphere (other than *s*)
Compute distance from point to sphere center
Compute the effective clipping radius
If distance is less than the effective clipping radius, eliminate point
Merge remaining points into *dense set*

Select initial set of triangle vertices:

Given: A dense set of points on global surface
Desired distance between vertices
Maximum deviation from desired distance
Want: A sparse set of points on global surface suitable as triangle vertices

Mark all points as **live**
Select any point and mark it as **candidate** and set its *neighbor count* to 1
While there are **candidate** points
Select the **candidate** point with the greatest *neighbor count*
Mark this point as **saved**
Mark all **live** and **candidate** points which are closer than the minimal acceptable distance to this point as **dead**
For all **candidate** points which are closer than the maximal acceptable distance, increment the *neighbor count*
Mark all **live** points which are closer than the maximal acceptable distance as **candidate** and set the *neighbor count* to 1
Collect all **saved** points into *sparse set*

Make triangles:

Given: An initial set of triangle vertices
Want: A set of connected triangles covering *global surface*

Form obvious edges by connecting points which are the desired distance apart, plus or minus the tolerance
Form triangles that are defined by the obvious edges
Find canonical polygons formed by edges which are not part of two triangles
Break concave polygons into convex polygons
Split small polygons into triangles by adding edges
Split large polygons into triangles by adding a new vertex
Collect all triangles

Find canonical polygons:

Given: List of edges
Want: List of polygons which cannot be decomposed into smaller polygons

Set maximum search depth to low integral value
Repeat
Mark all edges as untested
While there are untested edges
Select any untested edge
Find canonical path from one vertex to the other with search depth being maximum search depth and existing path being the untested edge
If path is found, add path and edge as a polygon and delete edges which are part of two polygons
Else mark edge as tested
Increase the search depth by low integral value
Until search depth is greater than number of unused edges

Find canonical path:

Given: Source vertex
Target vertex
Search depth
Existing path
List of edges
Want: Canonical path from one vertex to other which is shorter than given search depth.

If maximum search depth is zero, then return failure
If source vertex is target vertex
If existing path length is greater than two and the existing path is canonical, then return the path
Return failure
For all unused neighboring vertices
Find canonical path from neighbor vertex to target vertex with search depth being current search depth minus one and existing path being current existing path plus
neighboring vertex
If canonical path is found, return the path
Return failure

Testing the canonicity of a path:

Given: A path
List of edges
List of vertices
Want: The canonicity of the path

Unmark all edges and vertices
Mark all vertices and edges on the path
Find a vertex which is not part of the path
Mark this vertex
For all edges with one end being this vertex
Mark the edge
If the other vertex is unmarked
Recursively mark other vertex
The path is canonical if and only if all edges are marked

Break concave polygons into convex polygons:

Given: Possibly concave polygons
Want: All convex polygons

Set convex list to nil

```
For each polygon in polygon list
        If polygon has four or fewer sides
            Move polygon to convex list
            Process next polygon
Set threshold to very large number
Set concave pair to nil
For each vertex V of the polygon
        Find the two neighboring vertices
        For all vertices W which are not this one or a
        neighbor
            Compute distance from V and neighbors
            If the distance from V is less than both
                neighbor distances, then V is concave
                relative to W
            If V is concave relative to W and the
                distance between V and W is less
                than the threshold, set the threshold
                to the V-W distance and set concave
                pair to V and W
If concave pair is not nil
        Create edge between the two vertices
        Split the old polygon into two new polygons
        Delete the old polygon from polygon list
        Add new polygons to polygon list
Else
        Move polygon to convex list
```

Split small polygons into triangles by adding edges

```
Given:  List of convex polygons
Want:   List of triangles and large polygons

Set large list to nil
For each polygon
        Find closest pair of vertices which are not
            neighbors
        If the distance between these vertices is greater
            than four thirds the desired distance, then
            move polygon to large list
        Else
            Create edge between the two vertices
            Split the old polygon into two new
                polygons
            Delete the old polygon from polygon list
            Add new polygons to polygon list or
                triangle list, which ever is appropriate
```

Split large polygons into triangles by adding a new vertex

```
Given: List of convex polygons
Want:  List of triangles

For each polygon
        Compute center of mass of all vertices
        Find dead point closest to center of mass
        For each edge in polygon
            Create triangle from the vertices of the
                edge and the dead point
```