

BROWN UNIVERSITY
Department of Computer Science
Master's Thesis
CS-92-M1

“Mapping a 3D Surface to the UV Plane for Texture Mapping, Patchifying
and Metamorphosing”

by
Brian Knep

Mapping a 3D Surface to the UV Plane for Texture Mapping, Patchifying, and Metamorphosing

Brian Knep

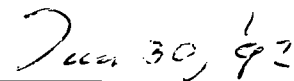
Department of Computer Science
Brown University
Providence, RI 02912
bk@cs.brown.edu

January 24, 1992

This thesis by Brian Knep is accepted in its present form by the Department of Computer Science as satisfying the thesis requirement for the degree of Master of Science.



Prof. Andries van Dam



Date

18

153-1

Mapping a 3D Surface to the UV Plane for
Texture Mapping, Patchifying, and
Metamorphosing *

Brian Knep

Department of Computer Science

Brown University

Providence, RI 02912

bk@cs.brown.edu

January 7, 1992

*This work was supported in part by grants from NSF, DARPA, IBM, NCR, Sun Microsystems, HP, and DEC.

Abstract

This paper presents an algorithm for mapping between a 3D surface and a connected subset of the uv plane. The resulting mapping is of interest because (1) it has a high level of continuity, (2) distinct surface points map to distinct points on the plane, (3) only at the discontinuities do surface points map to more than one point on the plane, and (4) the algorithm works on a wide range of surfaces regardless of their topology and connectivity. Several applications are discussed, including texture mapping a surface, wrapping a surface patch around an object, and transforming one surface into another.

CR Categories and Subject Descriptors

I.3.5 Computational Geometry and Object Modeling; I.3.7 Three-Dimensional

Graphics and Realism

Keywords

Surface Topology, Planar Graph, Texture Mapping, Surface Patch, Shape Transformation, Metamorphosing

1 Introduction

Algorithms that create mappings between points on a 3D surface and points on a 2D plane are useful for *texture mapping*, *patchifying*, and *metamorphosing*.

Texture mapping adds detail to objects by mapping a 2D image onto a 3D surface [8]. The correspondence between points on the surface and points in the image is called the *mapping function* [2]. A useful mapping function is one that uses each part of the image exactly once (i.e., it is *bijective* or one-to-one and onto) and maps adjacent 3D surface points to adjacent 2D image points (i.e., it is continuous)¹. These requirements cannot be satisfied for a surface of arbitrary topology, but a function can be created that minimizes discontinuities and maintains the one-to-one mapping everywhere except at the discontinuities. For example, if a surface is defined with a continuous parameterization, the uv coordinates of the surface define a mapping that is continuous and bijective except at the limits of the parametrization (where $u = 1$, for example).

For arbitrary polyhedra, however, most algorithms produce functions that are many-to-one, do not completely cover the image (not onto), or are discontinuous (see Figure 1). For example, Peachey [17] projects a surface point onto the plane by ignoring its z coordinate. Obviously, surface points having the same xy coordinates map to the same 2D coordinates, and parts of the image may not be used. This technique has been used to create the image in the center of

¹We use the term *image* to refer to a continuous intensity function defined on a rectangular region and not to a finite 2D array of intensity values (pixels)

Figure 1. Note how the mapping in many-to-one (a mirrored image shows up on the bottom of the torus) and not onto.

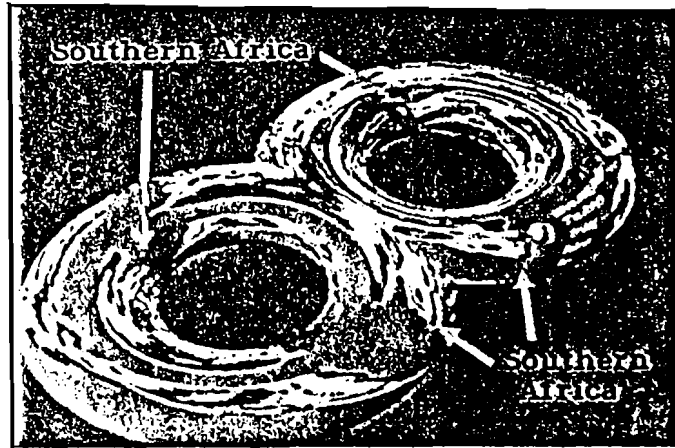
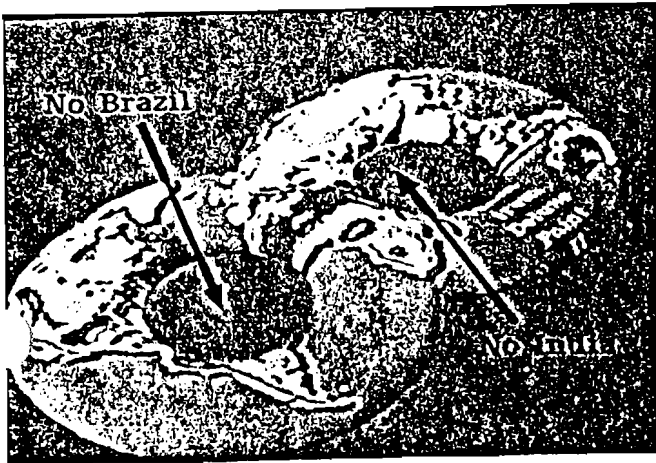


Figure 1: The results of using traditional methods to map the image on the top onto a two-holed torus. The image on the bottom left has been textured by projecting each surface point along the y -axis (i.e., the y -coordinate was ignored). The image on the bottom right has been textured by projecting each surface point along its normal onto an encasing sphere and using the uv coordinates of the intersection point.

Bier and Sloan [3] produce a more evenly distributed mapping by first encasing the surface inside an object that has a well defined parameterization, such as a sphere or cube. Surface points are then projected onto the encasing object and mapped to the corresponding uv coordinates. If the surface is convex or

star-shaped², it can be projected bijectively onto the encasing object. Otherwise, many surface points project onto a single point on the encasing object, resulting in a mapping function that is many-to-one. This technique has been used to create the image on the right of Figure 1. Note how the mapping is many-to-one.

If the inverse of the mapping function maps each 2D point to a single 3D point, then it can be used to map the control points of a surface patch onto an object (as explained in Section 4). The shape of the resulting surface patch approximates the shape of the original polyhedral object. We call this process *patchifying* an object. The advantage of having a patch rather than a polyhedral object arises when the user wants to make local changes that preserve high-order surface continuity. Moving a control point on a surface patch preserves this continuity; moving a vertex on a polyhedral object does not.

By combining one mapping function with the inverse of another, a correspondence can be made between the surface points of two distinct objects (see Figure 2). This correspondence can be used to smoothly transform from one 3D shape to another 3D shape in a process called *metamorphosis*. 3D metamorphosis is different from the 2D image “morphing” that has appeared recently in movies and television advertisements [12], but both have similar uses.

The metamorphosis of one object into another can suggest a development process, such as the growth of a tadpole into a frog [13]. By interpolating

²A star-shaped object is one in which there exists an interior point p such that a line from p to any other interior point is contained entirely within the object [16].

between two shapes, interesting objects can be created that can spark design ideas. Chen [4], for example, attempted to obtain an aerodynamic-looking car by interpolating between a car and a teardrop. He also used shape interpolation to identify design trends by averaging many similar designs. Several 3D metamorphosis algorithms have been presented, but they usually work on only a limited set of polyhedra [16, 13, 4].

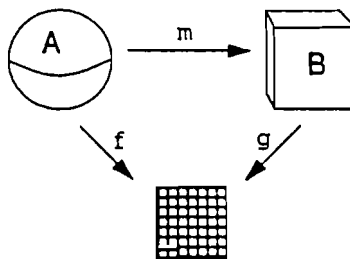


Figure 2: The mapping, m , from object A , a sphere, to object B , a cube, is a composite of the mapping, f , from A to the plane and the inverse of the mapping, g , from B to the plane.

This paper introduces a new method for mapping a polygonal surface in 3-space onto the uv plane. The resulting mapping maintains a high level of continuity while mapping distinct surface points to distinct points on the plane, and it works on a variety of surfaces. Most polygonal surfaces encountered in computer graphics can be mapped, as long as they are closed and connected³. Thus, the method can work on simple surfaces, such as spheres and tori, as well as more complex surfaces, such as ducts, volume data isosurfaces [9], and surfaces of objects built with constructive solid geometry (CSG) [14, 11]. The

³Each surface must be an embedding of a finite, closed, connected, simplicial 2-manifold [7].

algorithm has been used to map a texture onto an object continuously and bijectively except at the image borders, to patchify objects, and to metamorphose between any two of the objects described above.

The algorithm maps a surface by unfolding it onto the plane so that no two faces overlap. The surfaces described above, however, cannot be unfolded without being torn. Therefore, the first step of the algorithm is to cut the surface so that it can be unfolded. The actual unfolding can then be done directly; however, instead of unfolding in the intuitive manner by mapping the center and then flattening out toward the boundary, we first map the boundary and then flatten the rest of the surface. Our approach is different from the way Samek *et al.* [19] unfold a surface. Their algorithm first glues a single face onto the uv plane, and then unfolds outward from that face, resulting in a mapping that is many-to-one since faces can overlap.

The next section introduces the new algorithm, and Sections 3, 4, and 5 describe the application of the algorithm to texture mapping, patchifying, and metamorphosing respectively. Finally, conclusions and future work are presented.

2 Going From 3D to 2D

A bicontinuous one-to-one mapping cannot be constructed between one of the surfaces described above and a subset of the plane. The usual uv mapping of a

sphere, for example, where u is latitude and v is longitude, has a discontinuity where the lines $u = 0$ and $u = 1$ meet. This discontinuity arises because the sphere and the uv plane have different *topologies*. In this paper, the term *topology* refers to the topology of the underlying surface, not to the specific organization of the faces, vertices, and edges of a polyhedral approximation to the surface.

Our strategy is to find a collection of edges along which to cut the surface so that the resulting surface can be mapped continuously onto a subset of the plane. If the surface is cut along these edges, the resulting surface will have the topology of a *closed disc*. A closed disc is a connected subset of the plane that is bounded by a single closed loop and that contains its bounding edges [7]. (This is as opposed to an open disc which does not contain its bounding edges.) The cut-apart surface can be mapped continuously and bijectively onto any other surface with the topology of a closed disc (the filled unit square, for example) [7]. The algorithm presented in the next section finds such a cut.

2.1 Finding a Place to Cut

The algorithm for finding a place to cut is simple to implement. It starts by identifying a small subset of the surface having the topology of an open disc and repeatedly enlarges this subset until it contains all of the surface except for a few edges. These excluded edges form the boundary of the subset, and if the original surface is cut along these edges, the resulting surface can be unfolded

onto the plane.

Because the subset is an open surface, it does not contain the lines along its border. (Likewise, in this paper, faces are open surfaces and do not contain their bounding edges.) This subset is recorded as a collection of vertices, edges, and faces. Its boundary, which it does *not* contain, is stored as a list of edges ordered cyclicly around the faces in the subset.

The algorithm starts with a subset containing a single face of the original surface—any face can be used. The edges around that face form the initial boundary, and to enlarge the subset, faces are added and the boundary is extended (see Figure 3). A face can be added only if it shares a bounding edge with the subset. When a face is added, the shared edge is also added to the subset (and deleted from its boundary), and the other edges that border the face are added to the boundary. If a face touches the boundary along multiple edges, only one is added to the subset; the remaining edges are added to the boundary. Thus, edges can appear on the boundary twice.

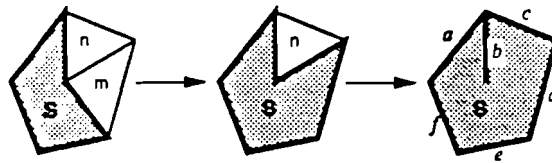


Figure 3: Two faces, m and n , being merged with the subset S . Note that only one edge bordering n is added to the subset. The final boundary, labeled on the bottom figure, contains, in order, edges a , b , b , c , d , e , and f . Edge b appears twice in the list.

The enlarging process stops when no more faces can be added to the subset—

that is, any face that touches the boundary is already in the subset. Since the original surface is connected, this occurs only when every face on the original surface is contained in the subset—the only surface points not contained in the subset are those along its boundary.

The merging operation does not change the topology of the subset. Thus, the subset always has the topology of an open disc and can be unfolded onto the plane. The final boundary of this subset contains the edges we are looking for: if the original surface is cut along these edges, the resulting surface can be unfolded onto the plane.

However, our goal is to produce a mapping that is highly continuous. Since the discontinuities arise along the boundary, it is advantageous to reduce the length of the final boundary before cutting the surface. One operation that can considerably shorten the boundary is the removal of edges whose two occurrences along the boundary are consecutive, as illustrated in Figure 4(a). Another shortening operation involves switching the boundary from one side of a face to the other, as illustrated in Figure 4(b). Since these operations maintain the topology of the bounded subset, the final, “shortened”, boundary still contains a set of edges along which to cut the original surface. Figure 5 shows the lines of the cut found by this algorithm on the surface of a torus.

There is one problem with these shortening operations. If the surface has the same topology as a sphere (genus 0), the resulting boundary will become empty. (This makes sense, since a sphere minus a single vertex has the same

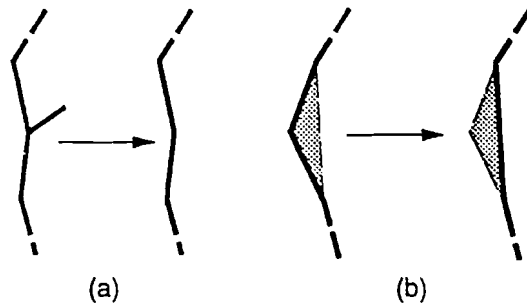


Figure 4: The boundary can be simplified by (a) removing edges whose two occurrences are consecutive, and (b) switching the boundary from one side of a face to the other.



Figure 5: Front and back views of a cut torus with areas on and near the cut highlighted in red.

topology as a disc—no edges need be cut.) To unfold the surface as described below, however, a boundary with at least 3 vertices is needed. Therefore, if the surface is a topological sphere, any two edges that share a single vertex can be used as the boundary. Cutting the original surface along these two edges gives a surface with the same topology as a disc.

To determine if the original surface is sphere-like, Euler's equation can be used: $G = 1 - (F + V - E)/2$, where G is the genus of the object (number

of holes in the object) and F , V , and E are the number of faces, vertices, and edges, respectively, of the surface. If $F + V - E = 2$, the surface has the same topology as a sphere, and any cut along two neighboring edges yields a surface that can be unfolded onto the plane. The algorithm checks this initially and treats it as a special case.

2.2 Unfolding Onto the Plane

The *skeleton* of a polyhedral surface is the set of vertices and edges of the surface and can be thought of as a graph. An *embedding* of a graph in the plane specifies the cyclic order of the edges around each vertex and face without specifying the 2D coordinates of the vertices; a *drawing* of a graph on the plane specifies these 2D coordinates. If a graph can be embedded in the plane with no edge crossings, the graph is *planar* and the embedding is called a *planar embedding* [18]. Likewise, a drawing of a graph in the plane without any edge crossings is called a *planar drawing*. In this section, we show that if a surface has the topology of a disc, planar drawing algorithms can be used to map the surface continuously and bijectively to the plane.

Remember that a surface with the topology of a disc can be mapped continuously and bijectively onto a subset of the plane. Therefore, there exists a planar embedding of the skeleton of the surface, and the skeleton is a planar graph.

If a planar graph is tri-connected, that is, at least 3 edges must be cut to

split the graph, then it has a *unique* planar embedding given the cyclic order of the edges around the outer face [6]. For a tri-connected graph, all planar drawings that have the same boundary edges (in the same order) have the same embedding. If the skeleton of a surface is tri-connected, then the planar drawing produced by mapping the surface to the plane continuously and one-to-one has the same embedding as all other planar drawings that have the same boundary edges as the surface. Vertices, edges, and faces on the surface correspond to vertices, edges, and faces in the planar drawings. Therefore, every planar drawing that has the same boundary edges as the surface specifies a continuous and bijective mapping between the surface and a subset of the plane. This means that planar-drawing algorithms can be used to map a surface to the plane if the skeleton of the surface is tri-connected. To ensure tri-connectivity, the original surface should be triangulated.

Planar-graph drawing has been thoroughly examined in graph theory [6]. Which solution to use depends on the application. Many of the algorithms, addressing the desire for a visually pleasing graph layout, try to maximize the aesthetics of the drawing. Some place vertices on an integer lattice to avoid precision problems. Some add bends to the edges to achieve a more evenly distributed layout. Unfortunately, no polynomial-time algorithms are known for many of the desired properties of a drawing: uniform edge lengths, even vertex distribution, and uniform face areas [6]. Therefore, which algorithm produces the “best” layout can be as dependent on the particular graph as it is

on the application.

For surface unfolding an even distribution of vertices, in a geometric sense, is often desirable. If two faces on the original surface have the same area, then the corresponding faces on the unfolded surface should have areas that are the same or similar. Keeping edge-length ratios the same is also desirable, as is minimizing how much the surface is stretched. While these are unsolved problems, some algorithms do better than others.

Here is a survey of four algorithms that find a planar drawing of a graph, and so can be used to unfold the cut-apart surface. Figure 6 shows the results of applying these algorithms to the same graph.

1. Tutte [21]: This algorithm places each vertex in the center of its neighbors by solving a system of linear equations. Because it involves matrix inversion, it runs in $O(n^3)$ time and uses $O(n^2)$ space. Although this algorithm is slow, it gives an even layout that does not cause irregular stretching of the original surface. Unfortunately, however, the ratio of the area of the largest face to the area of the smallest face can be exponential with respect to the number of vertices in the graph [6]. Therefore, even if all the faces on the original surface are of the same size, the faces on the unfolded surface could be of drastically different sizes. The algorithm can also run into precision problems because it works in the space of real numbers.
2. Chiba *et al.* [5]: This algorithm gives layouts similar to those achieved by Tutte, but in linear time. The layouts, however, are not as symmetric

and evenly distributed as the Tutte layouts. This algorithm is easy to code, but it also works in the space of real numbers and thus has precision problems.

3. Relaxation [6]: This method treats the vertices as mass points and the edges as springs. The border vertices are fixed in place, and the dynamics of the system are simulated until the springs stabilize. If the springs are of the same strength, the final layout places each vertex in the center of its neighbors—just like Tutte's algorithm. However, the strength of each spring can be made proportional to the original edge length to get a more even vertex distribution. If the initial positions of the vertices are close to the final ones, this method can run a lot faster than Tutte's. Chiba's algorithm can be used to provide a good first guess.
4. Battista, Tamassia and Tollis [1]: This linear-time algorithm works only on directed acyclic graphs (DAGs). Converting a nondirected graph to a DAG is simple. First a planar drawing of the graph is produced using another algorithm. Then directions are assigned to edges on the basis of their slope: edges are directed upward and to the right. This algorithm places vertices on an integer lattice and so does not run into precision problems. It also splits some of the edges so that they can be bent. Of the four algorithms considered, this one seems consistently to generate the most evenly distributed drawings, but the bent edges cause irregular stretching of the unfolded surface.

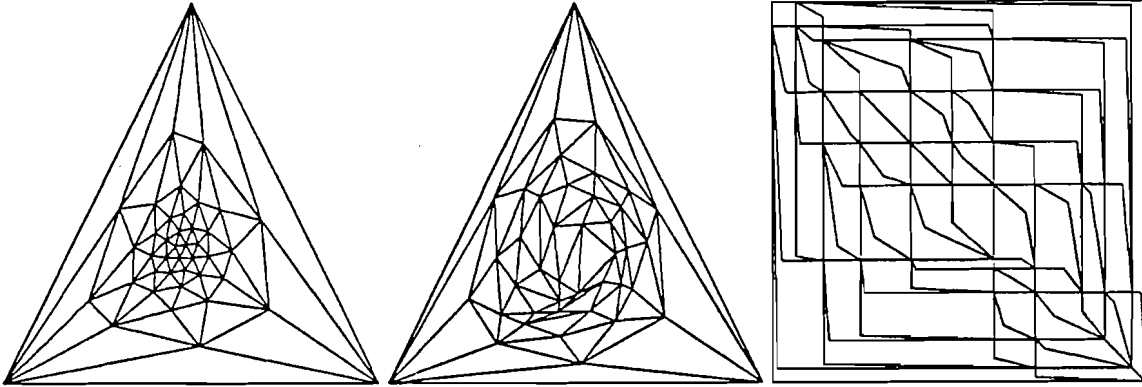


Figure 6: Planar drawings of the same graph produced by (from left to right) Tutte's, Chiba's, and Battista's algorithms.

Once the vertices and edges of the original surface have been mapped to the plane using one of the above algorithms, any point on the original surface can be mapped. The 2D coordinates of a point are computed by interpolating the 2D coordinates of the vertices on the border of the face that contains the point⁴.

For Tutte's and Chiba's algorithms and the relaxation methods, the user specifies not only the edges that will be on the boundary, but also the positions of the boundary vertices. It is crucial that the boundary is a convex polygon with no internal angle $\geq 180^\circ$ since, otherwise, internal edges that connect boundary vertices might overlap or extend beyond the boundary. To satisfy this requirement, the vertices can be placed around the circumference of a circle. The unfolded surface will be a disc in the plane.

For texture mapping and patchifying, however, it is more useful if the unfolded surface has the shape of a square. To prevent internal edges that connect

⁴Barycentric coordinates can be used for non-triangular faces.

boundary vertices from overlapping the boundary, any such edge and the faces on either side of it are split (as illustrated in Figure 7). The vertices are then distributed evenly around the edges of a square with one vertex at each corner, and the rest of the surface is unfolded.

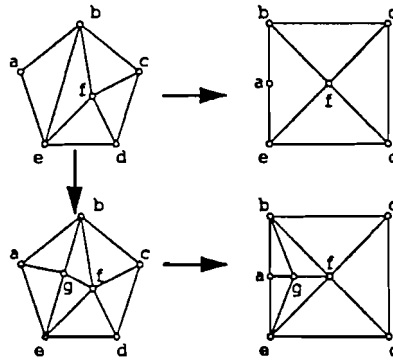


Figure 7: If the graph on the upper left is mapped to the unit square, the edge between b and e will overlap the boundary of the resulting graph (on the upper right). To avoid this overlap, the edge and the two faces it borders are split, and the resulting graph (on the lower left) is mapped to the unit square. The final graph drawing is shown on the lower right.

For Battista's algorithm, the user can control which edges appear on the boundary, but not their final positions. This lack of control makes it more difficult to use the mappings created by this algorithm. However, it often produces graphs with square boundaries, and we believe that it can be modified to always produce such graphs.

3 Texture Mapping

The cutting and unfolding algorithm described above can provide the correspondence between points on the surface of an object and points in a 2D image. The uv coordinates that result from the mapping can be used as indices into the 2D texture map. Using this method, each point on the surface maps to a unique uv point, and the mapping is continuous except at the border. The result is that the texture is literally wrapped around the object, as illustrated in Figure 8.

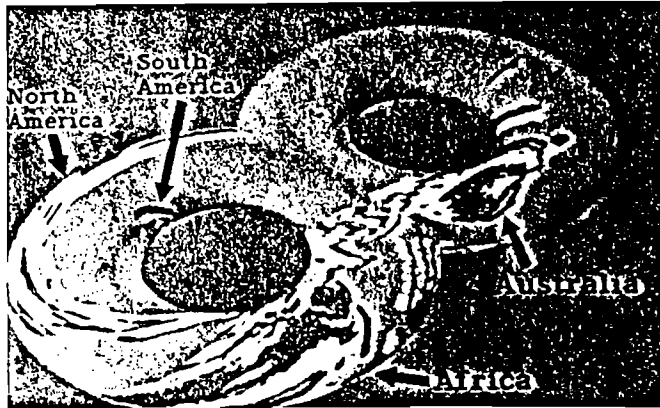


Figure 8: The image from Figure 1 mapped onto a two-holed torus by cutting and unfolding the torus onto the plane.

As with most mapping functions, the resulting texture is distorted because of the stretching that occurs when the surface is flattened. To minimize distortion, the unfolding algorithm should minimize how much the surface is stretched and avoid stretching the surface in irregular or abrupt ways.

A uv mapping is also useful for direct WYSIWYG painting and texturing [10]. WYSIWYG methods allow the user to directly manipulate the surface

properties of points and sets of points on an object. The changes can be stored in a 2D map. The painted figures and textures may be distorted on the map, but the distortion does not matter since the map need only record the changes made to each point of the surface. The uv mapping introduced in this paper is particularly useful because it generates a one-to-one mapping. (If the mapping is many-to-one, changing one part of the surface affects other parts.)

For WYSIWYG painting and texturing, the algorithm that unfolds the surface should maintain the ratios of face areas. If not, given two equal-area faces, one might cover only a few pixels in the 2D map, and the other might cover many pixels; this could lead to aliasing problems and different levels of detail on the two faces.

4 Patchifying

Surface patches are extremely useful in modeling objects that require continuity along their surfaces. When the position of a control point is changed, the surface deforms continuously. Surface patches, however, are not ideal in modeling all types of objects. Objects of arbitrary topology, for example, are often easier to build with CSG or by sculpting volumetric data [9] (although S-patches may help build these types of objects [15]). These methods are sometimes more natural and intuitive than surface-patch modeling because they manipulate volumes rather than surfaces. The panel of a car door, for example, is well suited to

surface modeling, while the Venus de Milo is well suited to volume modeling.

With CSG and other volume methods, however, it is often difficult to make local changes that maintain surface continuity. An obvious compromise is to model the basic shape of the object using volume methods, and then cover the object with patches and continue modeling using surface methods. The mapping presented in this paper can be used to wrap a surface patch around an object.

First both the surface and the patch are mapped to the filled unit square. Then point-location techniques [18] are used to determine into which face of the unfolded surface each control point has been mapped. The 3D coordinates of the vertices around a containing face are interpolated to get the 3D coordinates of a control point. Figure 9 shows the results of wrapping a patch around the surface of an object sculpted with volume data [9].

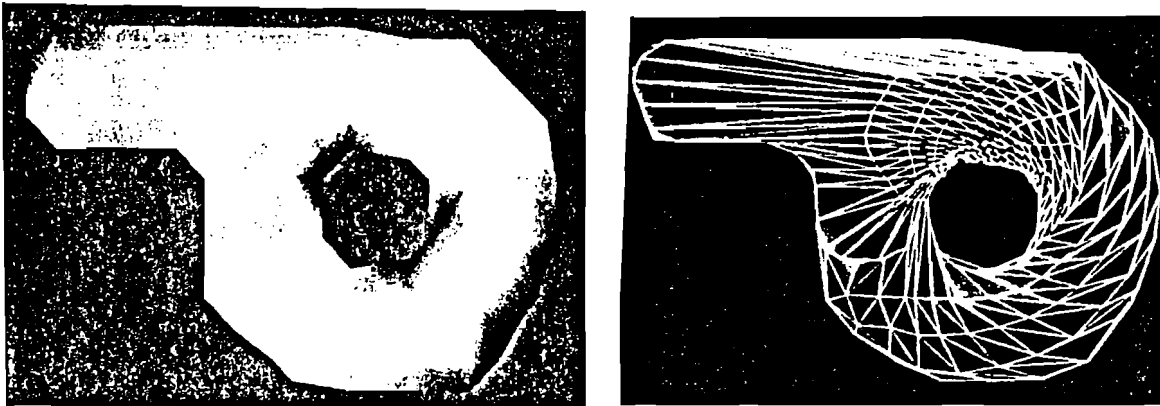


Figure 9: An object created with volumetric data and a 70×70 Bézier surface patch wrapped around that object.

The distribution of the control points across the surface should be even and fairly regular. If it is not even, large surface protrusions and dents might be

completely missed. If it is chaotic and not regular, moving a control point affects remote parts of the surface. The unfolding algorithm is what determines this distribution, and so should be chosen with care.

Turk [20] describes an algorithm for distributing n points somewhat regularly around a surface. His method could be modified to create a more even distribution of the control points over the surface of the object.

5 Metamorphosing

Metamorphosing can be divided into two parts: finding correspondences and interpolating. The first part involves establishing correspondences between the surface elements of one object and those of the other. The second part involves interpolating between corresponding surface elements [16].

5.1 Finding Correspondences

Most of the literature on 3D metamorphosis deals with the correspondence problem and not the interpolation problem. Chen [4] slices each object into a finite number of parallel slices and uses 2D techniques to find correspondences between the border vertices of matching slices. His method works on only a limited set of objects because the border of each slice must be a single loop. Parent [16] breaks each surface into sheets of connected faces and metamorphs between corresponding sheets. His algorithm, however, requires the user to intervene for objects with genus greater than zero. Kent *et al.* citekent:topo-

merging, use a method similar to Bier and Sloan's [3]: they project the surface elements of each object onto a surrounding sphere and find correspondences on the sphere. Although the authors claim that their method can be extended to nonconvex, non-starshaped objects, they provide no indication of how this might be done.

Our cutting and unfolding algorithm can be used to create a mapping between two closed, connected surfaces of arbitrary genus. The mapping will be continuous and bijective except along a single closed loop on each surface (where the surface is cut).

The technique used to map the vertices of one surface onto the other surface is the same as that used to wrap a patch around an object (in Section 4): both objects are mapped to the unit square, point-location techniques are used to determine into which face of one surface each vertex of the other surface has been mapped, and the 3D coordinates of the vertices around a containing face are interpolated to get the 3D coordinates of a contained vertex.

If the vertices of one object (*objA*) are transformed onto the surface of another object (*objB*), then the shape of *objA* will approximate the shape of *objB*, but the two objects will not match exactly. Discrepancies arise because the mapping function does not map surface details of *objA* one-to-one onto surface details of *objB*. Sharp edges and other details (like the tip of a cone) of *objB* may not show up on the transformed *objA*.

So as not to lose any surface details when metamorphosing, all the vertices

and edges of objB are mapped onto objA (on the plane), and objA is modified (by splitting its faces and edges) to include these elements. Likewise, all the vertices and edges of objA are mapped onto objB (on the plane), and objB is modified to include these elements. This gives us two objects (modified-objA and modified-objB) that have the same number of faces, vertices, and edges connected in the same way. Linearly interpolating between the positions of the vertices metamorphs one surface into the other. Figure 10 shows several frames of a metamorphosis produced using our method.

—
SEE ATTACHED PAGE
—

Figure 10: These frames are taken from an animation of the torso and head of a robot metamorphosing into the body of a race car.

5.2 Interpolating

Once the composite object has been created, its vertices can be linearly interpolated between the surface of objA and the surface of objB. The vertex normals, however, cannot be linearly interpolated because they might not correctly follow the movements of nearby faces (see figure 11). The interpolation must take into account the movements of the faces surrounding each vertex. Instead of using world-space normals, each vertex normal is described relative to the average of the normals of the surrounding faces. This is done by creating a basis out of the average face normal, an edge touching the vertex, and the cross product of the two. Each vertex normal is projected into the space defined by this basis.

To compute a world-space vertex normal at any stage of the metamorphosis, the relative vertex normals are linearly interpolated, the new basis is found, and the interpolated vertex normal is projected into world space. Often, however, linear interpolation is good enough, at least for a first pass.

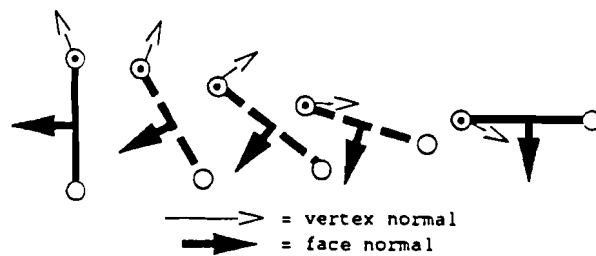
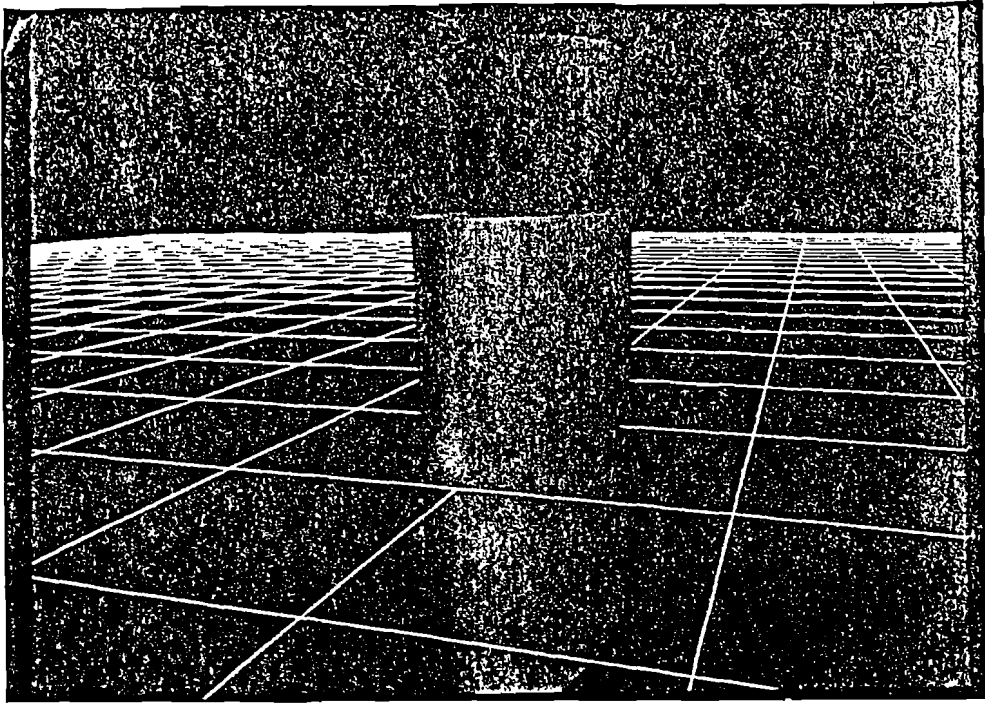
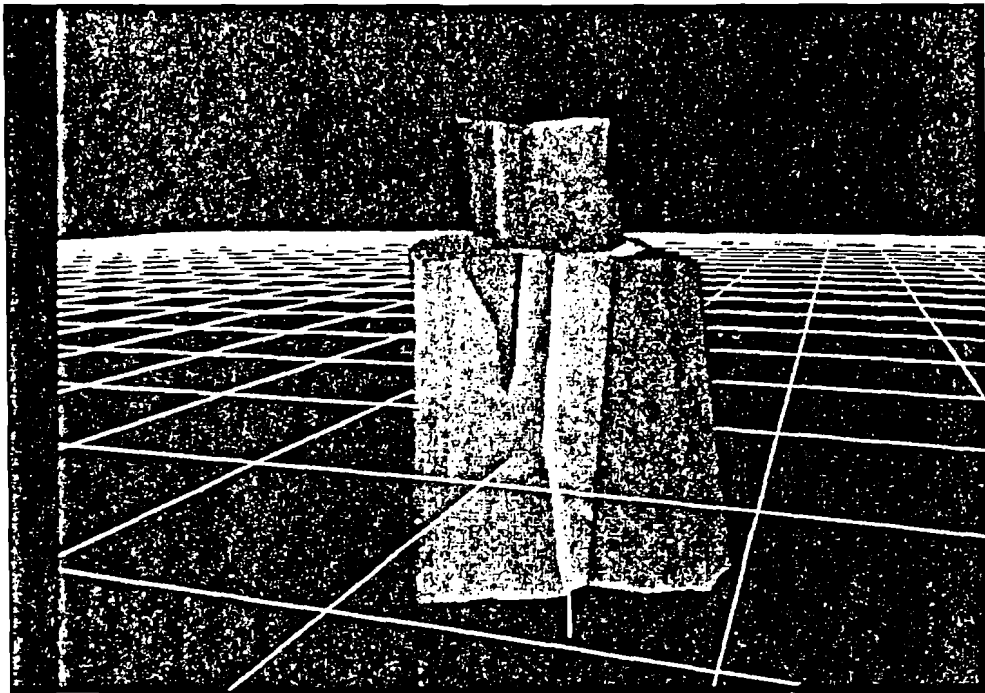


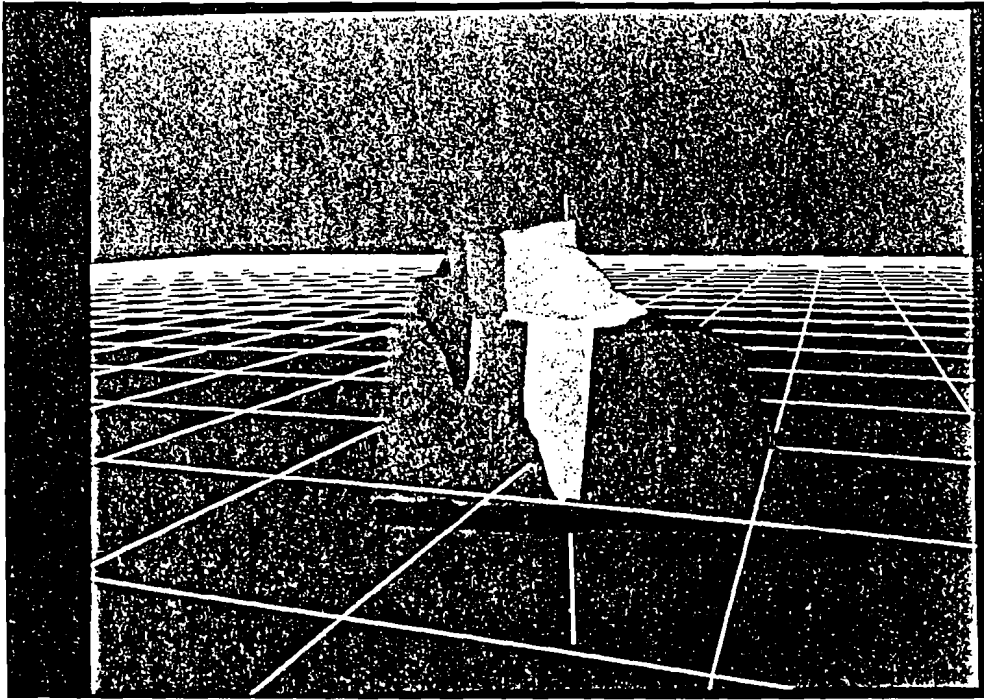
Figure 11: If vertex normal interpolation does not take face movements into account, the normal might swing behind the face.



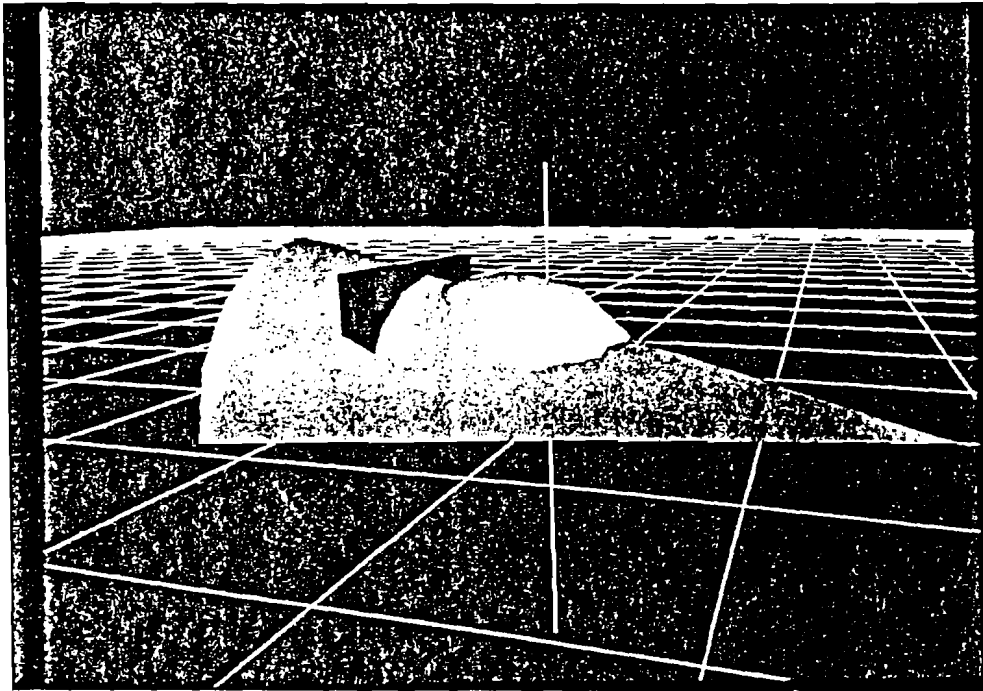
10a



10b



10c



10d