Surface Graph Sketching

D. Hutchinson, F. Lin, W.T. Hewitt

Computer Graphics Unit,
Manchester Computing,
University of Manchester,
Manchester, United Kingdom.

email [Dave.Hutchinson | F.Lin | w.t.hewitt]@mcc.ac.uk
WWW http://info.mcc.ac.uk/CGU/

Abstract
The production of patterns and designs upon surfaces has received only limited attention, with texturing techniques being accepted as adequate. Texturing, however, can be a very unsatisfactory and clumsy method of producing complex and accurate designs upon a surface, even more so with surfaces such as NURBS where a texture may appear very much distorted when mapped to a surface. This paper introduces a new approach called surface graph sketching which draws upon ideas from planar graph sketching for the construction of complex curve networks across parametrically defined surfaces. Surface curves can be created by a number of methods and stored in the parameter space of the surface thereby allowing intricate and more importantly accurate surface designs to be created. Regions of the surface may be assigned a property such as colour or texture, or cut away to produce holes. Surface graph sketching has a number of applications, in the design of trimming curves, surface patterning and texture map definitions.

Keywords: Geometric Modelling; NURBS; Patterning; Surface Graph Sketching; Texturing; Trimming;

1. Introduction

In computer graphics one of most useful techniques for creating 2D graphical designs is that of planar graph sketching, first introduced by Gangnet, Herve, Pudet and Thong [5]. A planar graph is a figure formed by a set of intersecting lines and curves which split a plane into a number of regions which can be given a property such as colour or texture. By assigning different properties to defined regions it is possible to construct very complex designs which would not normally be possible with conventional drawing packages. Areas where this technique is useful include the design of logos and cartoon cell drawing and inking.

Now let us consider a similar problem, but this time we wish to design a pattern upon a surface. As we are dealing with an internal representation of a surface, drawing directly onto the surface is problematic. The best we can hope to do is to texture map a design onto a surface, however this has disadvantages. It is difficult to guarantee where or how different parts of a texture will appear on a surface except in the simplest cases. A good example of this occurs for unevenly parameterised surfaces, such as NURBS, where a circle on a texture map may appear very much distorted when mapped to a surface. This problem has been addressed to some extent by Litwinowicz and Miller [8] in the form of interactive texture map placement, and by Hanrahan [6], who allows the painting of texture directly onto the surface of objects. Both of these techniques require some dexterity on the part of the designer, however for some applications form a satisfactory solution.

In this paper we extend the ideas of planar graph sketching into the surface domain and show how it is possible to design complex patterns over parametric surfaces to create impressive graphical effects. We call this technique *surface graph sketching*.

A surface graph is formed by a set of intersecting curves on a surface. Curves on a surface can be inserted into the surface graph using a number of methods, including direct sketching in 2D parametric space, manipulation of curves on the surface, surface/surface intersection or curve projection. Because all the curves on a surface

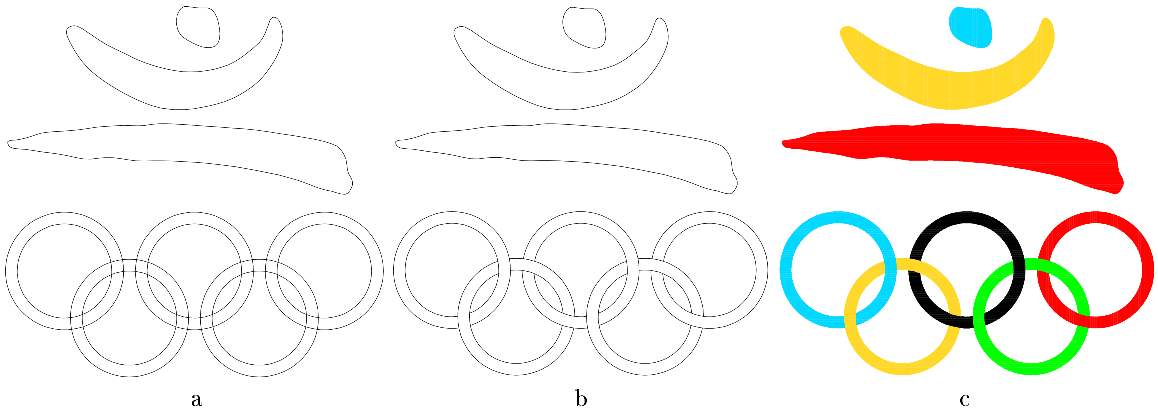


Figure 1: *Stages in the Creation of a Simple Design Using Planar Graph Sketching:: a. A number of curves are drawn; b. Some curve segments are removed; c. Regions are assigned properties.*

can be projected into the 2D *uv* parametric space and represented as 2D curves, the problem of surface graph sketching is actually reduced to the problem of planar graph sketching. By constructing a surface graph in the 2D parametric domain of a surface in the spirit of planar graph sketching, and by using a surface rendering method to render the regions with an assigned property such as colour or texture, it is possible to design complex patterns over surfaces with impressive results. Surface graph sketching has further application not only for creating complex patterns across over a surface, but in the creation of texture map images, and also the construction of trimming curves.

This paper is organised as follows. Section 2 introduces the principles of planar graph sketching. Section 3 extends these ideas to surface graph sketching. Section 4 shows the main application areas of surface graph sketching and section 5 provides some closing comments.

2. Principles of Planar Graph Sketching

In this section we review planar graph sketching techniques and briefly outline the methods required for implementation, including curve intersection algorithms, data structures, editing procedures and assignment of properties to map regions. This is a necessary base for our discussion of surface graph sketching as many of the ideas from planar graphs are required.

2.1. Overview

Planar graph sketching techniques, introduced by Gangnet *et al* [5], and also discussed from an end-user point of view by Baudelaire and Gangnet [1], has for some time been used by the graphical artist to design logos or for cartoon cell drawing and inking. These techniques are used to construct complex patterns and shapes in a 2D environment, in a way that most graphical packages are incapable of. For example if a designer draws four lines say in a rectangular shape, in most packages it is impossible to usefully classify the region bounded by those lines. The sketching procedure involves creating or drawing a number of lines and curves some of which may intersect as in figure 1a. Curves or curve segments may then be deleted as required as shown in figure 1b to create differently shaped regions. In the final stage properties are assigned to the regions bounded by curves to give the final image, figure 1c. For this kind of technique to be used interactively the geometry of the lines and curves must be stored carefully in a data structure. One of the data structures that can be used for this is a *winged edge data structure*. Topological information about the planar graph must also be stored, such as the colour of a region bounded by curves.

Planar graphs consist of straight lines and curves in some form. Bézier curves have been used [5] to represent all lines and curves, as straight lines may be represented exactly by Bézier curves as well as a wide variety of curve types. More recently Lin and Hewitt have extended upon this idea by representing all lines and curves internally as NURBS curves [7]. Non uniform rational B-splines (NURBS) curves are more suitable than Bézier curves as they possess all of the properties of Bézier curves, and have the benefit of offering a designer further degrees of freedom, as well as being capable of exactly representing conic sections.

2.2. NURBS Curve Intersection

As we have mentioned already, the intersection of two curves is required for planar graph sketching. There exists a substantial volume of work regarding curve intersection algorithms for rational splines, but as this work does not concern itself with improving upon those techniques we will only briefly cover the relevant aspects.

The intersection of lines and curves in planar graph sketching is important for creating a robust application. For 2D NURBS curves, a number of optimising techniques exist. A good comparison of Bézier curve intersection algorithms is given by Sederberg [11], and these methods may be extended to NURBS curves. The method we use for calculating an intersection point is as follows. A NURBS curve with non-negative weights lies entirely within the convex hull of its control points. If the convex hulls of two curves overlap then there is a possiblitity that the curves intersect. If this is so, then the curves are subdivided and the test repeated. This subdivision continues until the curves approximate line segments, and the intersection point can be found as the intersection of the two approximating lines. A curve can be said to approximate to a straight line when the maximum distance between the curve and its chord is below a predefined tolerance. This process may be optimised to some extent by first splitting the NURBS curves into monotonic segments by finding all points on the curves with vertical and horizontal tangents. In this case, the convex hull overlap is reduced to finding the overlap of axis aligned rectangular boxes, which is a much faster test.

2.3. Winged Edge Data Structure

One of the methods for representing a planar graph internally is to use a *winged edge data structure* introduced by Baumgart [2,3], and this is also the most useful way for our needs. A good discussion on this subject is given by Chiyokura in his book [4]. A winged edge data structure consists of information regarding faces, edges and vertices, and a list of all of these must be maintained. A face in this context is a region bounded by a number of edges and vertices. An edge is a NURBS curve, or more often a segment of a NURBS curve defined by the limits $[t_{min}, t_{max}]$, which has a vertex at each end. It is not always necessary to maintain a list of faces for planar graph sketching, however it is useful to have this data available in surface graph sketching for a more efficient application implementation, this is mentioned in section 3.2.

The winged edge structure is designed in such a way that given any face, edge or vertex, the local topology around that region can be found. This is because the faces, edges and vertices store information about each other. Given a face for example, an ordered list of all of the edges and vertices surrounding it may be found. A face knows one of the edges that bounds it. An edge has knowledge of the vertices at both ends, the face that is on its left and right sides, and at least one edge on both of those faces. Finally a vertex knows of one edge that is incident at it.

There are two versions of the winged edge description, one involves an edge storing four adjacent edges, the other has an edge only storing two adjacent edges. We use the latter form in our implementation and figure 2a shows a schematic. Given an edge on a face it is possible to traverse about the edges of that face in the directions indicated by the arrows, and an example (figure 2b) winged edge structure is also shown with the arrows indicating the direction in which loops are followed.

If a new curve is introduced into a planar graph, then all of the intersection points between it and the edges already present must be found exactly. These points are added to the winged edge structure as vertices, as are the start and end points of the new curve. These vertices are connected by edges, and the winged edge structure updated. Edges in the winged edge model may be selected and deleted from the data structure. A frequent occurrence of this is the removal of *hanging edges*, that is edges which have no adjacent edges at one or both of their vertices. This operation is defined as the inverse of adding an edge. It is a useful operation as hanging edges are of little use after a planar graph is complete.

As well as storing information about the curves, in a planar graph further information is required concerning

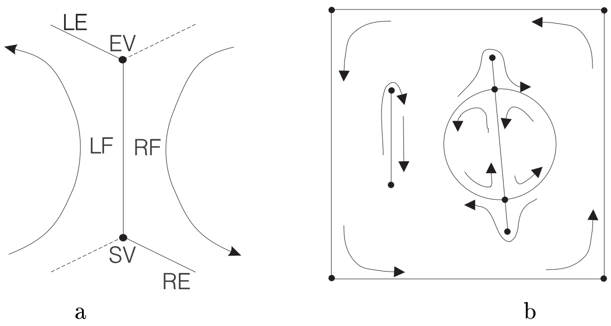


Figure 2: *A Schematic of the Winged Edge Data Structure. a: start and end vertices SV,EV of an edge, the face on the left and right of the edge LF,RF, and the adjacent edges at each vertex LE,RE. b: an example set of edges and vertices with the traversal directions.*

the topology. An edge must know the curve that it is a segment of and a face should have information about whether it is to be coloured textured or removed, although this is more important in the final stages of the design process.

3. Surface Graph Sketching

We will now describe how to apply NURBS planar graph sketching techniques [7] to the construction of surface graphs on parametric surfaces. By a parametric surface we mean that a position in 2D parameter space with parameters (u,v) may be mapped by some function S to a 3D position in (x,y,z). In general, the parameter space of a surface is taken in the range $[0,1]$ in both the u and v directions.

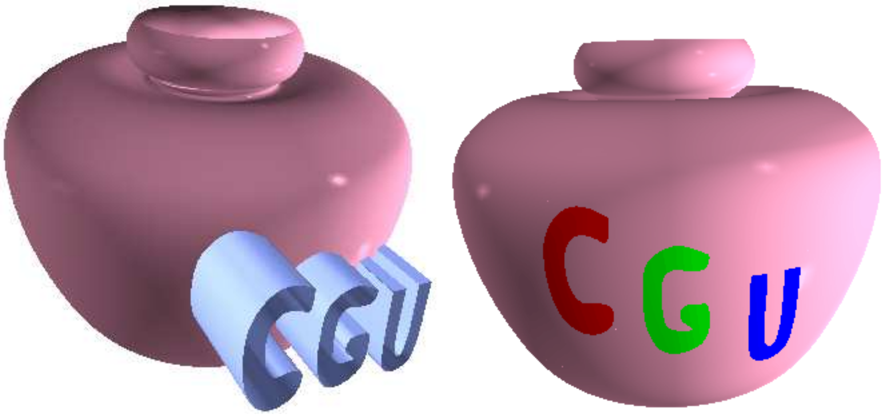
So then, instead of constructing lines and curves in a 2D environment as we described in the previous section, we are constructing surface curves in 3D which are stored in the 2D parameter space of the surface. A surface graph is actually represented as a 2D planar graph. This is a consistent method for storing surface curves, as in geometric design for example, trimming curves on a NURBS surface are stored as NURBS curves in the parameter space. Surface graph sketching in effect then divides a surface up into a number of different regions bounded by curves and curve segments which we may colour or texture as required. Surface graph sketching can be used throughout many fields of computer graphics to increase the effects available to a designer.

It should be emphasised that the type of surface we are considering does not need to be a NURBS surface, and any parametric surface type can be used. It is however useful to consider NURBS surfaces alone for surface graph sketching as they can be used to represent most surface types. A polygonal mesh for instance that may be used for animation can be expressed as an order 2 NURBS surface, or a network of Bézier patches can also be expressed in NURBS form. Due to the property of NURBS to express many surface types both free-form and analytical, we will from this point talk in terms of surface graph sketching onto NURBS surfaces.

The remainder of this section discusses the process of building up a surface graph as well as a number of application details. It also looks at the problems of sketching directly onto parameter space and shows how it can be resolved by using other methods such as surface/surface intersection, curve projection and direct manipulation of trimming curves on a surface.

3.1. Constructing Surface Curves

How then is it possible to construct surface curves in such a way that a designer can usefully specify their position on a surface? If we allow the sketching procedure described for planar graphs on the 2D parameter space, then the problem immediately arises in a similar way to texture mapping, that a circle for example in parameter space does in no way guarantee a circle on the surface except in simple cases. It can be useful however to define a curve in this way as a first guess, and then manipulating it as described below. More complex high order NURBS make this procedure more difficult. Fortunately there are a number of methods at our disposal which allow better control over the way in which our surface graph is constructed, and these are outlined below.



a b

Figure 3: *Three Shapes Intersecting with a Base Surface and the Resulting Coloured Effect*

3.1.1. Direct manipulation of Curves on a Surface

Given a curve on a surface, it may sometimes be desirable to adjust its shape in some way. This could be performed in the parameter space of the surface by selecting control points of the curve and moving them in the uv space. However, the resulting curve movement on the 3D surface for a given move in parameter space can be unpredictable. To overcome this difficulty we can allow a curve to be manipulated directly on the surface in 3D by displaying the control points of the surface curve. Then by allowing a designer to select those points, the curve can be dragged across the surface as though it were a 3D curve except that it is constrained to move on the surface. When we are happy with the curve, it may be added to the surface graph.

3.1.2. Adding Surface/Surface Intersection Curves to the Surface Graph

Another way of defining curves in the surface graph is to intersect surfaces with the base surface. A surface may be positioned using an editor such that it forms an intersection with the base surface. Then the curve which prescribes the intersection line on the base surface between the two may be calculated as a NURBS curve and added to the surface graph. Using surface/surface intersection is also consistent with our surface curve description as the intersection curves between two NURBS surfaces are generally expressed as NURBS curves. Surfaces such as spheres, cylinders or custom made shapes can be used in order to get very exact and precise control over the surface curve being added, see figure 3. This method of adding curves overcomes all of the problems mentioned earlier about parameter space sketching giving unpredictable results, for example, prescribing circular holes is a sphere or cylinder intersection. The surface used for the intersection need not be closed and the intersection curve also need not be a closed curve. So then planar lines across a surface may be constructed using a planar surface.

A Note on NURBS Surface Intersection

Rational surface/surface intersection has received even more attention than the curve case discussed earlier due to its importance in geometric design and manufacturing [9]. Two of the main and simplest methods to implement for computing the intersection of rational spline surfaces are the marching method and the subdivision method, although other numerical solutions have been discussed.

Subdivision techniques do not require a starting point, and in the simplest case involve subdividing the surface into ever smaller patches until a limit is reached where it is considered that a patch is planar, thus the problem is reduced to finding the plane/plane intersection of the two patches. All of the intersection segments found from the subdivided regions must be gathered together in a final phase and assembled into a complete intersection curve or number of curves. This method deals with multiple intersection curves very well.

The marching method involves finding a start point on an intersection curve of a surface, and *tracing* or following the intersection curve along its length. This is done by constructing infinitesimally small patches in the estimated direction of travel, before calculating the next intersection point exactly. This proceeds in an iterative manner until either the curve meets up with the initial intersection point or comes to the limit of the surface. Some problems with this technique are that for cases with more than one intersection curve it is not always easy to guarantee that two starting points do not lie on the same intersection curve, although for only one intersection curve on a surface this method is more efficient than subdivision.

In our implementation a standard subdivision method is used to evaluate points on the intersection curve. The parameter values of these points are found and a NURBS curve interpolated through them. Although it is not often the most efficient method of intersection, for our needs it provides an easy to implement solution giving predictable results for most cases, i.e. ones where singularities are not involved and very small loops in the surface need not be accounted for.

3.1.3. Curve Projection

A technique which has emerged recently has been the projection of a curve onto a surface to define trimming regions. A curve is first constructed in 3D space and positioned in a suitable way relative to the surface to be trimmed. A direction is then chosen in which to project the curve onto the surface. A surface curve can be constructed where this projection intersects with the surface. It is in essence the same as using surface/surface intersection with an extruded surface, that is one which was formed by extruding a curve in a given direction. This is also a useful method for defining curves in a surface graph. A curve can projected onto the surface under design and the resulting intersection curve added to the surface graph.

3.2. Property Assignment and Application Considerations

Having constructed a number of curves on the surface graph and removed any unwanted curve segments and hanging edges, there will be a number of distinct regions left, and we are in a position to begin assigning properties to them. A method is required for selecting the faces in the graph. There are two choices, we can allow the user to select regions on the real 3D model for property editing, however this involves rotating the model to see obscured faces and is a clumsy method. The other is for the designer to click within regions of the surface graph in 2D *uv* parameter space, and calculate which face has been selected, this is called point location. By following a line in one direction from a point on the 2D graph and intersecting it with all edges, a list of intersection points can be found for intersected edges and the closest one chosen, in this case the face on the point side of the edge is the selected face. A number of properties can now be assigned to the surface region. These can be a colour or a texture map, also surface properties such as diffuse, specular and transmission coefficients may be specified. The region may also be labelled as cut away, in which case it will not be displayed.

In order for this technique to be useful for a designer, the user interface should be constructed to provide adequate information as the graph is constructed, that is the changes on the surface graph should be visualised on the real surface as editing proceeds. Here the advantage of our data structure becomes apparent. In any operation we can know precisely which regions of the surface are affected and only these regions need be recalculated and re-rendered.

In our implementation there are two windows, a 2D window which allows curves to be sketched and the surface graph edited, and a 3D window which shows the surface with its surface pattern developing as changes are made to the surface graph. It is possible for curve segments to be deleted in both the 2D and 3D windows, and as we have already mentioned, curve segments may be modified using their control points in both windows also.

4. Applications of Surface Graph Sketching

Having described some of the techniques for constructing surface graphs, we can go on to describe a number of useful application areas. These are the construction of complex trimming curves, surface pattern design and texture map creation.

4.1. Trimming Curve Construction

Although in general trimming curves are not required to be of a complex nature, in most applications we can use surface graph sketching for trimming curve construction. A surface graph can be built up as we already

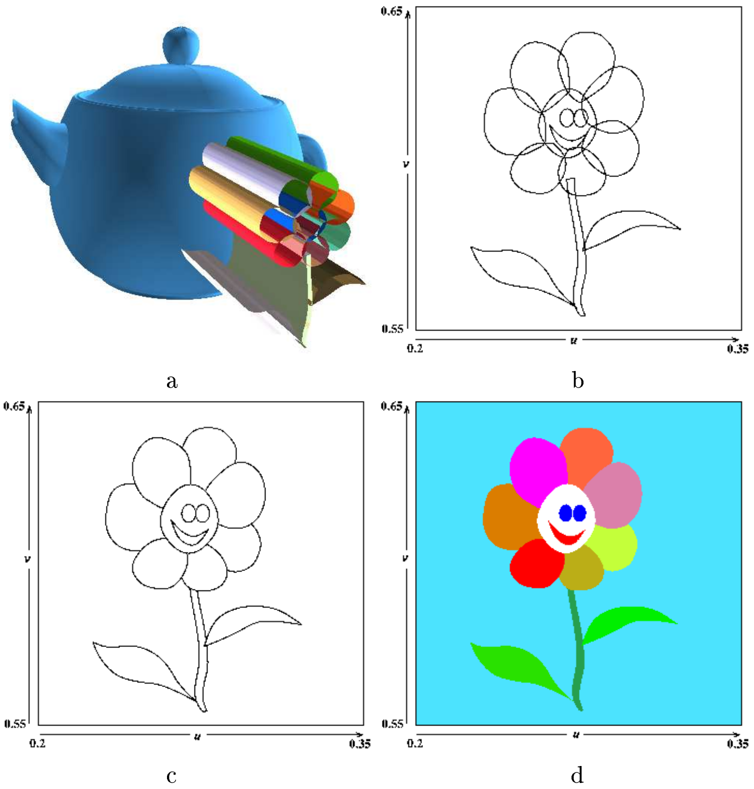


Figure 4: *Stages in the design of a Surface Graph*

Figure 5: *The Resulting Design On the Teapot*

described, edges deleted if required and then regions specified which are to be trimmed away. By generating 2D parameter space curves which define the perimeter of a trimmed region, the surface may be rendered with that section cut away.

Trimming curves are extracted from a surface graph in the following way. Given a face in the surface graph that we wish to trim away, it is possible to find a list of edges bounding that face as a list of NURBS curve segments. For the specification of a trimming curve our graphics hardware requires a closed curve to be given in parameter space, which runs in a clockwise or anti-clockwise direction, depending on whether the region within the curve is to be cut away or kept. At the moment what we have is a list of curve segments around a face, not necessarily the same curve order, and not necessarily following in the same direction around the face.

The list of curves must be concatenated into one closed curve. As the curves may not be of the same order, the first step of the operation is to elevate [10] all of the curve segments to the same order as that of the highest order curve in the list. We can then traverse along the list taking each curve segment in turn and appending it to the new trim curve. If the direction of the curve segment is not in the traversal direction then that segment should be reversed, that is the control points and knot vector reversed. Further care is taken to ensure that the final trimming curve has a knot vector normalised to 1, and that the continuity between each segment is accounted for.

4.2. Surface Patterns

In the spirit of planar graph sketching, for surface graph sketching we can assign different properties to the faces in our surface graph. We can use solid colours with diffuse and specular effects specified, texturing, or the region may be classed as invisible and cut away.

In the first instance a number of curves are introduced to the surface graph, in this example those curves are formed by intersection, figure 4a. The resulting initial surface graph is shown in figure 4b, along with the parameter range that it covers. Some curve segments are removed to produce the regions which will be coloured, figure 4c. Figure 4d shows the surface graph after regions have been assigned a colour. Finally each face is rendered as a trimmed region of the base surface. The results of this can be quite impressive as shown in figure 5. It should be noted that this particular example does not highlight the advantages of surface graph sketching over texture mapping due to the relative simplicity of the model used. The shape of the flower in 2D parameter space is very similar to the actual 3D surface pattern, although positioning the flower in the correct position would still be problematic. In section 4.3 we give an example of why surface graph sketching is advantageous over normal texture mapping techniques.

The way in which the surface is rendered may be optimised by grouping together regions which have the same surface properties and assigning multiple trimming curves to each surface rendering. Most graphics hardware does not deal with overlapping trimming curves very well, but we can be assured by virtue of the technique that this will not occur, assuming that all hanging edges have been removed.

4.3. Defining Texture Maps

A texture map is an image in some format which can be mapped onto a parametric surface. We have already mentioned that texture mapping is a poor technique for prescribing specific shapes on a surface, especially for surfaces with uneven parameterisation or having peculiar geometry. To show this we give two examples. In the first case we produce a circular shape on a surface by intersecting it with a cylinder. The resulting surface graph is shown in figure 6a, and the circular shape on the surface in figure 6b. The second example shows the result of attempting to map a circle in parameter space, figure 6c, to the same surface in approximately the same position. As we can see in figure 6d the circular shape becomes very much distorted. Even then for this simple example we have shown that texture mapping is an unpredictable method of producing accurate images.

It is possible to use surface graph sketching to produce texture maps. Because surface graph curves are stored in the 2D parameter space of a surface, it is possible to effectively inverse map from a 3D surface graph to produce a 2D image or texture map. The image can be produced by processing the stored 2D planar graph (or parameter space) of the surface, which represents the 3D surface graph, using scan line conversion techniques.

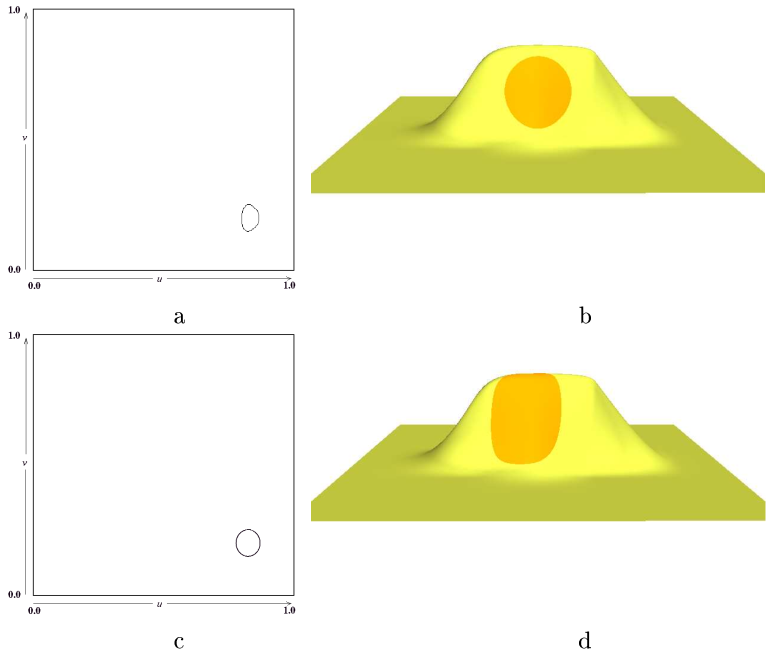


Figure 6: *Surface Graphs on Badly Parameterised Surfaces; a: The surface graph of a circular hole; b: The circular shape on the surface; c: Placing a real circle on the surface graph; d: The resulting deformed shape on the surface.*

Surface graph sketching then is a useful and accurate method for producing texture maps for arbitrary surfaces.

5. Final Comments

In summary we have presented a method for designing a complex and accurate surface graph over a parametric surface by extending planar graph sketching techniques to the surface domain. Curve sketching, surface/surface intersection and curve projection methods have been shown to be a good way of constructing surface graphs. Surface graphs have been applied to the design of trimming curves, patterning and colouring of surfaces and texture map generation and we have given a number of example results from our prototype implementation which show that surface graph sketching can be powerful tool for creating impressive graphical effects.

Acknowledgements

The authors would like thank all the staff at the Computer Graphics Unit for their help and encouragement. Dave Hutchinson would like to thank the Engineering and Physical Sciences Research Council for their financial support during this work.

References

1. P Baudelaire and M Gangnet. Planar maps: an interaction paradigm for graphics design. In *CHI'89*. Addison-Wesley, 1989.

2. B G Baumgart. Geometric modeling for computer vision. *Stanford AI Project memo 249*, 1974.

3. B G Baumgart. A polyhedron representation for computer vision. *Proc. NCC*, 1975.

4. H Chiyokura. *Solid Modelling with DESIGNBASE, Theory and Implementation.* Addison Wesley, 1988.

5. M Gangnet, Jean-Claude Herve, T Pudet, and Jean-Manuel Van Thong. Incremental computation of planar maps. *Computer Graphics*, 23(3):345–354, 1989.

6. P Hanrahan and P Haeberli. Direct wysiwyg painting and texturing on 3D shapes. *Proceedings of SIGGRAPH 90 (Dallas, Texas, August 6-10, 1990). In Computer Graphics, 24, 4 (August 1990) ACM SIGGRAGH, New York, 1990, pp 303–310.*

7. F Lin. *NURBS in CAD and Computer Graphics.* PhD thesis, University of Manchester, 1996. submitted January, 1996.

8. P Litwinowicz and G Miller. Efficient techniques for interactive texture placement. *Proceedings of SIGGRAPH 94 (Orlando, Florida, July 24-29, 1994). In Computer Graphics Proceedings, Annual Conference Series, 1994, ACM SIGGRAGH, pp 119–122.*

9. N M Patrikalakis. Surface-to-surface intersections. *IEEE Computer Graphics and Applications*, 13(1):89–94, 1993.

10. L Piegl and W Tiller. *The NURBS Book.* Springer-Verlag Berlin Heidelberg New York, 1995.

11. T W Sederberg and S R Parry. Comparison of three curve intersection algorithms. *Computer Aided Design*, 18(1):58–63, 1986.