A Client-Server System for the Visualisation of Algebraic Surfaces on the Web

Richard Morris

Department of Statistics, University of Leeds, Leeds LS2 9JT, UK

Abstract. Algebraic surfaces, defined as the zero set of a polynomial function in three variables, present a particular problem for visualising, especially if the surface contains singularities. Most algorithms for constructing a polygonization of the surface will miss the singular points. We present an algorithm for polygonizing such surfaces which attempts to get accurate representations of the singular points. A client-server approach, with a Java applet and a C program as backend, is used to enable the visualisation of the polygonal mesh in a web browser. This system allows algebraic surfaces to be viewed in any web browser and on any platform.

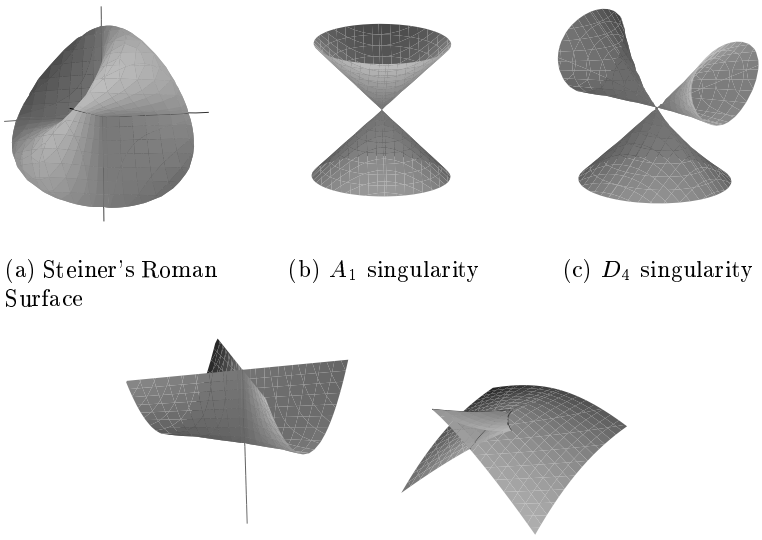
1 Introduction

Algebraic surfaces, defined as the zero set of a polynomial function in three variables, have a long history in mathematics. There are many famous surfaces such as Steiner's Roman Surface, Fig. 1(a), an immersion of the real projective plane, which is represented as the algebraic surface $x^2y^2 + y^2z^2 + z^2x^2 = 2xyz$.

Algebraic surfaces often contain singular points, where all three partial derivatives vanish. For example the double cone, $x^2 - y^2 - z^2 = 0$, has an A_1 singularity or node at the origin, Fig. 1(b). There are other more complicated isolates singularities such as: $x^2y - y^3 - z^2 = 0$ which has a D_4 singularity, Fig. 1(c). Other surfaces are more complicated and can contain self-intersections, $xy = 0$, and degenerate lines, $x^2 + y^2 = 0$. The cross-cap or Whitney umbrella, $x^2z + y^2 = 0$ contains a self intersection along $x = z = 0$, $y > 0$ and a degenerate line along $x = z = 0$, $y < 0$. The line forms the 'handle' of the umbrella, Fig. 1(d). The swallowtail surface, $-4z^3y^2 - 27y^4 + 16xz^4 - 128x^2z^2 + 144xy^2z + 256x^3 = 0$, is even more complicated and contains a cuspidal edge, Fig. 1(e). These and other examples of algebraic surfaces will be further examined in section 4.

These singularities cause particular problems for constructing computer models of the surfaces. Many algorithms will simply ignore the singular points [2,10]. However if information about singularities is included from the ground up it is possible to construct an algorithm, described here, which can produce good 3D models of most algebraic surfaces.

The surfaces are displayed in a web-page using a Java applet which uses the JavaView library [12,13] to allow rotation of the surface. This applet



(a) Steiner's Roman Surface (b) A_1 singularity (c) D_4 singularity

(d) A cross-cap (e) A swallowtail

Fig. 1. Some algebraic surfaces

connects to a server on the Internet which actually calculates polygonization of the surface.

The program described here has been adapted from an earlier program [8,9] which ran as a standalone application on SGI machines and used the Geomview [11] program for visualisation. The principal improvements have been the Java interface and an improved method of finding the polygonization 3.5.

2 The client applet

The client side of the system is fairly straightforward. It consists of a Java applet written using the JavaView library. It has two panels, one of which displays the surface and allows the surface to be rotated and scaled using the mouse. The other panel contains an area to input the equation of the surface as well as controls for selecting the region space in which the surface will be calculated. Several predefined equations are provided. These include many well known algebraic surface. The syntax of equations is standard TeX style notation and allows sub-equation to be defined as well as allowing a symbolic differentiation operator and vector operations. The user interface is shown in figure 2.

A button press causes the surface is to be calculated. A CGI-POST request, which encodes the defining equation and options, is sent to the server

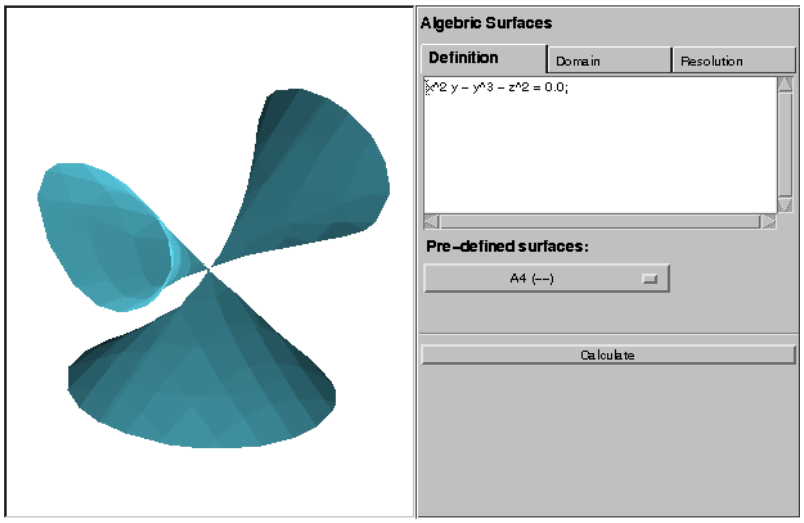


Fig. 2. The user interface for the program

which then calculates a polygonization of the surface. This is then returned to the client in JavaView's JVX format. If the defining equation is very degenerate, say a reducible equation like $x^2 = 0$, then the server can take a long time to calculate the surface. To prevent this happening a maximum calculation time is specified by the user. If this time limit is exceeded then the calculation of the surface will end prematurely. Ideally an interrupt button could be provided to halt the calculation of the surface, but this cannot be achieved using the CGI protocol.

Due to Java security restrictions the Java applet can only connect to servers which lie on the same Internet host. This makes it difficult for users to modify the applet or include it in their own software. This could be overcome by signing the Java code.

3 The server

The server takes the defining equation, $f(x, y, z) = 0$, of an algebraic surfaces and produces a ploygonization of the surface inside a rectangular box specified by the user.

The basic algorithm starts with a rectangular box. Recursive sub-division is used to split that box into 8 smaller boxes, the edge-lengths of the which are half those of the original box. Inside each of the smaller box a test based on Bernstein polynomials (Sec. 3.1) is used to determine whether the box might contain part of the surface. In such case the recursion continues breaking the box into eight more boxes. We found that three levels of recursion, giving boxes whose edge lengths are an eighth of those of the original box, are enough

to give a course representation of the surface and four levels of recursion produce quite a detailed model.

After this recursion each of the smaller boxes is examined in greater detail. Three types of points are found (Fig. 3):

1. Points on the edges on the box where $f = 0$.
2. Points on the faces of the box where $f = 0$ and at least one of partial derivatives, $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$ or $\frac{\partial f}{\partial z}$ vanish. We shall call these *2-nodes*.
3. Points in the interior of the box where $f = 0$ and at least two of the partial derivatives vanish. We shall call these *3-nodes*.

Recursive algorithms are used for each of these steps which are described in sections 3.2, 3.3 and 3.4.

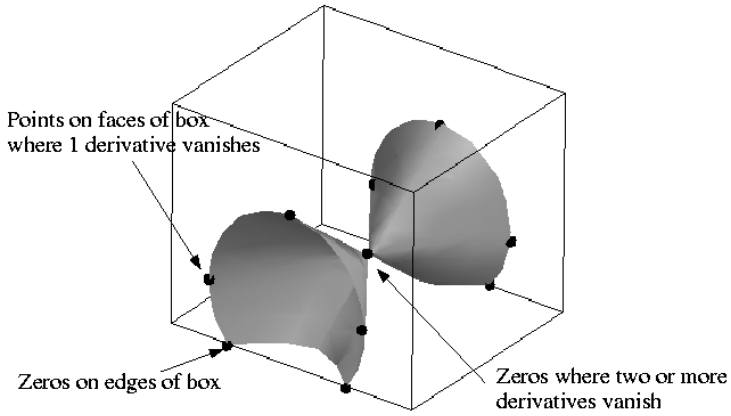


Fig. 3. The types of solutions found in a box

Finally the points found are connected together to give a polygonization of the surface which is returned to the client (Sec. 3.5).

A few assumptions about the surface are necessary to avoid degenerate cases: that the surface does not intersect the corners of the box; that none of the partial derivatives vanish at the solutions on the edges of the box; and that the 2-nodes on the faces of the box are isolated. Provided that the polynomial is not reducible, i.e. not of the form $h(x, y, z)(g(x, y, z))^2 = 0$, then all these assumptions can be satisfied by putting the surface in general position. This can always be achieved by slightly changing the bounds of the box. Typically the domain needs to be constructed with unequal bounds so that the origin, which is often a singular point does not lie at a corner of a box.

3.1 Bernstein polynomials

The computations involved in the program are made much simpler by the use of Bernstein polynomials. These offer a quick test to see if a polynomial might have a zero inside a domain. All the results in this section are well known and the algorithms have been taken from a method for drawing algebraic curves in 2D, described by A. Geisow [6] and details of the implementation can be found in [8].

A 1D **Bernstein polynomial** $B(x)$ of degree n is written as

$$B(x) = \sum_{i=0}^{n} b_i \binom{n}{i} (1-x)^i x^{n-i}.$$

The b_i's are the **Bernstein coefficients**. We are only interested in Bernstein polynomials which are defined over the range $[0,1]$. In three dimensions the Bernstein representation of a polynomial of degrees l, m and n in x, y, and z is

$$\sum_{i=1}^{l}\sum_{j=1}^{m}\sum_{k=1}^{n} \binom{l}{i}\binom{m}{j}\binom{n}{k} (1-x)^i x^{l-i} (1-y)^j y^{m-j} (1-z)^k z^{n-k}.$$

A test for zeros

If all the Bernstein coefficients of a 1D Bernstein polynomials have the same sign, all strictly positive or all strictly negative, then the polynomial has no zeros between 0 and 1. A similar result happens in the 3D case. This is easily proved by noting that $(1-x)^i x^{n-i}$ is non-negative for $x \in [0,1]$ and $0 \leq i \leq n$. Note the converse does not always hold and it is possible to construct a Bernstein polynomial which has coefficients of different signs but no zeros on $[0,1]$.

Other algorithms

Several other routines are necessary for the operation of the program:

- constructing a Bernstein polynomial from a standard polynomial, this involves rescaling the domain so that it fits $[0,1]$,
- evaluating a Bernstein polynomial at a specific point,
- calculating the derivative of a Bernstein polynomial,
- splitting the domain into two equal halves and constructing Bernstein polynomials for each half.

The last of these algorithms is necessary for the recursion steps, where a box is split into 8 smaller boxes.

3.2 Finding solutions on edges

A simple 1-dimensional sub-division algorithm is used to find the solutions on an edge of the box. A 1-dimensional Bernstein polynomials is constructed by restricting of the function to the edge. If all the coefficients of the Bernstein polynomial are the same sign then there is no solution on the edge. Otherwise Bernstein polynomials are constructed for each of the partial derivatives. If the Bernstein coefficients for any of the partial derivatives are not all of the same sign then there may be more than one solution on the edge. In such cases the edge is split into two and the process repeated for each sub-edge. Otherwise the signs at the end points are examined to determine whether there is a solution on the edge. If so, the solution is found by repeatedly sub-dividing the edge and looking for a change of sign. The sub-division is carried out until sub-pixel level is reached.

3.3 Finding nodes on faces

Another recursive procedure is needed to find solutions on the faces of the box where one or more partial derivatives vanish. This routine is also used to find lines connecting solutions on the face and its edges.

For a given face the 2-dimensional Bernstein polynomial b is constructed. Bernstein polynomials are also constructed for the three partial derivative functions. There are a number of case shown in Fig. 4.

- If the coefficients of b are all of the same sign then the surface does not intersect the face and the face is ignored, Fig. 4(a).
- If the coefficients of b are not all of the same sign and the coefficients of each of the partial derivative are all of the same sign, then there are exactly two solutions on the edges of the face. These solutions are connected by a line on the face and the recursion end, Fig. 4(b).
- If the coefficients of any one of the derivatives fail to be all of the same sign then the face is divide into four smaller faces. Each of these face, and its edges, is then recursively tested, Fig. 4(b).

This process is carried out recursively until a pre-defined depth, typically pixel level, is reached.

When the bottom level of recursion is reached the face may contain a node and further processing is needed to deduce the geometry. If only one derivative vanishes then there may be a turning point, where $f = \frac{\partial f}{\partial x} = 0$ say. Typically there will be one of the situations shown in Fig. 5. These can be distinguished by counting the number of solutions on the edges of the face and examining their derivatives.

- If there are no solutions then the face is ignored, Fig. 5(a).
- If there are two solutions and the signs of the derivatives match then they are linked by a line, Fig. 5(d).

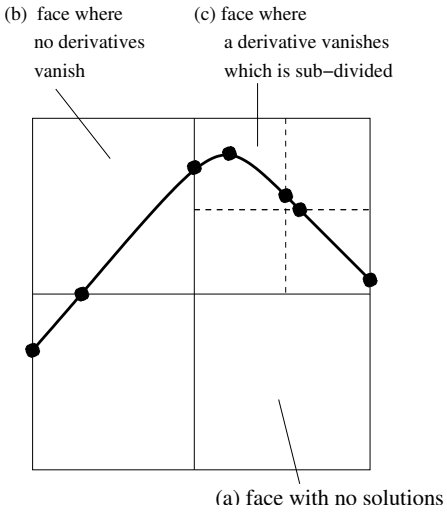


Fig. 4. Sub-dividing a face

- If there are four solutions then they are the tested for signs of their derivatives pair-wise. Pairs with matching derivatives are linked by lines. Fig. 5(c).
- If there are two solutions which have different signs for the partial derivative then a 2-node is constructed in the centre of the face and this is linked to each of the solutions, Fig. 5(b).

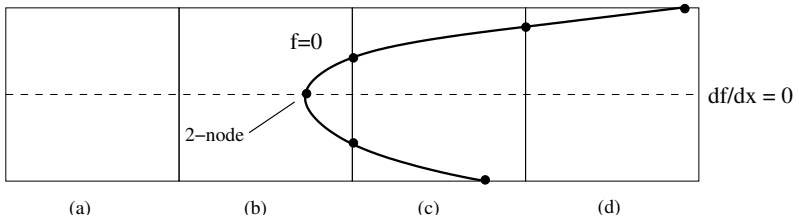


Fig. 5. Faces where one derivative vanishes

Consider the case shown in figure 6. Here two partial derivatives vanish in both faces, yet only one contains a 2-node. To distinguish between the two cases observer that the two curves $\frac{\partial f}{\partial x} = 0$ and $\frac{\partial f}{\partial y} = 0$ only cross in the face which contains the 2-node. In this face a 2-node is created in the centre of the face and linked to the each of the solutions on the edges. In the other face the solutions on the edge are linked pair-wise. This situation typically occurs when a self-intersection of the surface crosses a face, in which case all three derivatives will vanish. A similar situation occurs when a degenerate

line passes through the face: the zero sets of all three derivatives will intersect in a single point. This example illustrates the limits of using recursion, finer levels of recursion would not help resolve this case as the geometry looks similar even under greater magnification.

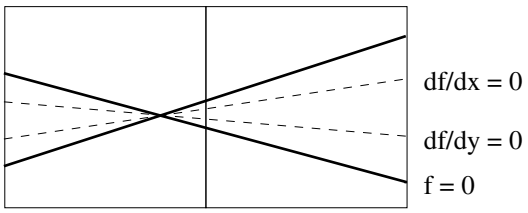


Fig. 6. Two faces where two derivatives vanish, only the left-hand one contains a node

3.4 Finding singularities inside a box

A recursive procedure is used to find the 3-nodes inside a box where two or more derivatives vanish. These can either be singularities where all three derivatives vanish or points like the north-pole of the sphere, $x^2+y^2+z^2=1$, where two derivatives vanish. Including the latter type of point helps produce better polygonization as it does not truncated the surface.

This recursion splits each box into eight sub-boxes and the Bernstein test is used to tell whether the function f or its derivatives vanish.

- If f does not vanish then the box is ignored.
- If none of the derivatives vanish then the box is ignored.
- If only one derivative vanishes then the number of 2-nodes on the faces of the box is found and the signs of their derivatives is examined.
  - If there are no 2-nodes the box is ignored.
  - If there are two 2-nodes and the signs of their derivatives match then the 2-nodes are linked by a line and the recursion ends.
  - If there are four 2-nodes then the signs are compared pair-wise to see how they link together. Matching pairs are linked by lines.
- Otherwise, when two or more derivatives vanish, the geometry can not be easily be established and the recursion continues.

When the bottom level of recursion is reached it is assumed that the box contains a singularity (or point like the north pole of a sphere). A 3-node is constructed in the centre of the box and linked to each of the 2-nodes on the faces of the box. It may also be an isolated point where all three derivatives vanish but there are no 2-nodes on the faces.

The test for 3-nodes is too strong and it is possible that some points are incorrectly marked as singularities. This is illustrated by the swallowtail surface where several incorrect isolated points are found near the cuspidal edge, Fig. 7.

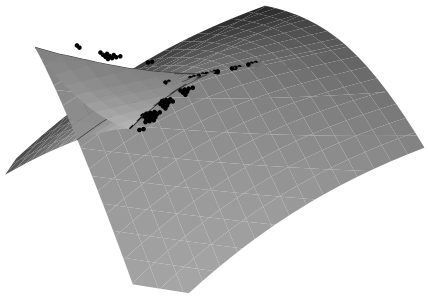


Fig. 7. Incorrect isolated points found near the cuspidal edge of a swallowtail surface

3.5 Constructing a polygonization

The final stage in the algorithm is to construct a set of polygons which approximate the surface. This is carried for each box found in the first stage of the recursion. At this stage there is a set of points linked by lines. Some of the points lie on the edges and faces of the box and others (3-nodes) may lie in the interior. However there is no information about which of the lines form the boundaries of which polygons. It would be possible to gather such information while finding the 3-nodes inside the boxes. However, this would require many more sub-boxes to be examined which would slow down the algorithm. Instead a more ad hoc algorithm is adopted, for most surfaces this will give a reasonable polygonization of the surface and there are only a few cases where it does not produce a correct polygonization. These cases typically occur when more singularities than really exist have been identified in Sec. 3.4.

The basic idea behind the algorithm is to construct polygons whose edges just consist of the lines on the faces of the box and then modify the polygon so that they include the internal lines. As a precursor to the main algorithm two sets of lines are found:

- *Cycles:* closed loops of lines which lie on the faces of the box.
- *Chains:* connected sets of lines joining 3-nodes in the interior of the box and 2-nodes on its faces. The end-points of each chains will be 2-nodes on the faces of the box.

For many simple cases where there are no internal points there will be no chains and the cycles will form the boundaries of the polygons. For other cases some refinement is necessary:

- If the cycle forms a figure of eight shape, the cycle is split into two cycles which contain no self intersections. This situation occurs when the surface has a self-intersection.

- If there are two disjoint cycles which are linked by a two or more non-intersecting chains then the surface will form a cylinder. In such cases two new cycles are formed. Each form half the cylinder split along the chains Fig. 8.
- If two points on a cycle are linked together by a chain then two new cycles are formed which include the lines in the chain and some of the edges of the original cycle.

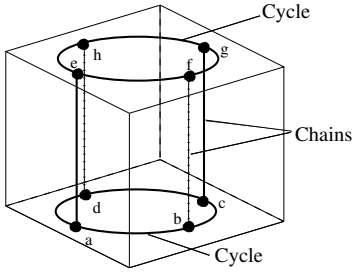


Fig. 8. Constructing a cylinder. The cycles a-b-c-d and e-f-g-h and the chains a-e, c-g are linked to form two cycles a-b-c-g-f-e and c-d-a-e-h-g

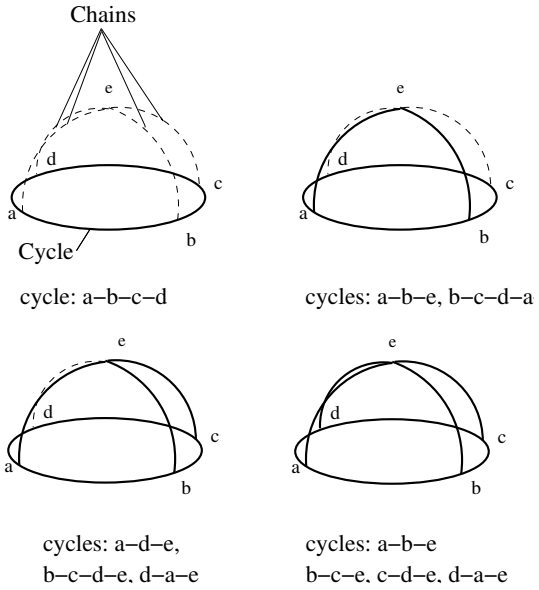


Fig. 9. Steps in the process of creating a polygonization of the top half of a sphere. Stating with a cycle and four chains the lines in the chains are progressively added to create four cycles used for the polygonization

These refinement happen until no more refinement are possible. The cycles then form the boundaries of the polygons. An example of this process is shown in Fig. 9 where three steps are needed to produce the final set of cycles. In practice the geometry is often more complicated that this example, Fig. 10 shows the polygonization for four boxes near a D_4 singularity. Note that several 3-nodes have been found near the singularity and the topology of the object is not quite correct.

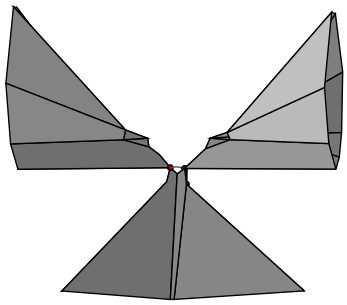


Fig. 10. A close-up of the D_4 singularity showing the polygons found. Note how too many 3-nodes have been found leading to a topologically incorrect polygonization

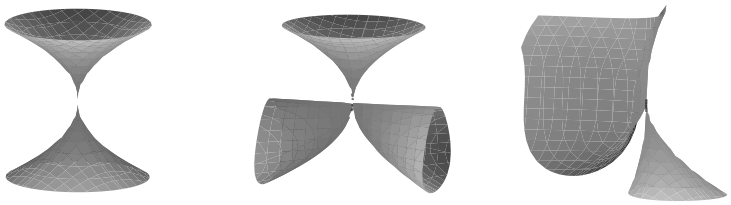
4 Examples of Algebraic Surfaces

One area of study involving algebraic surfaces is singularity theory [3]. An important theorem of V. I. Arnold, [1, pp. 158–166] classifies the types of simple singularities which occur for functions $\mathbf{R}^n \to \mathbf{R}$. These consist of two infinite sequences: $A_k, k \geq 1$, $D_k, k \geq 4$ and three other singularities E_6, E_7 and E_8. The normal forms of these for functions $\mathbf{R}^3 \to \mathbf{R}$ are

- A_k: $\pm x^{k+1} \pm y^2 \pm z^2$, where $k \geq 1$,
- D_k: $\pm x^{k-1} + xy^2 \pm z^2$, where $k \geq 4$,
- E_6: $\pm x^4 + y^3 \pm z^2$,
- E_7: $x^3y + y^3 \pm z^2$,
- E_8: $x^5 + y^3 \pm z^2$.

Further singularities exist which are not technically simple, however these have higher co-dimensions and are less frequently encountered. The zero sets of some of these normal forms are shown in Figures 1 and 11.

The singularities mentioned above are the only ones which occur in generic families of functions $\mathbf{R}^n \to \mathbf{R}$. In particular the singularities are always isolated. However, many of the well known functions are decidedly non-generic and can contain self intersections, triple points, degenerate lines, cross-caps



(a) A_3 (b) D_6 (c) E_7

Fig. 11. Zero sets of the normal forms of some singularities

and cuspidal edges. Steiner's Roman surface is an example which contains six cross caps.

Discriminant surfaces are an important class of surfaces which are not generic when viewed as functions. Consider the family of quartic polynomials $f(t) = t^4 + zt^2 + yt + x$, which will have a repeated root whenever $f(t) = 0$ and $\frac{df}{dt} = 4t^3 + 2zt + y = 0$. Solving these equations for t gives the swallowtail surface $-4z^3y^2 - 27y^4 + 16xz^4 - 128x^2z^2 + 144xy^2z + 256x^3 = 0$ (Fig. 12). Points of this surface will give values of x, y, z for which $f(t)$ will have a repeated root. Furthermore, if the point lies on the cuspidal edge then $\frac{d^2f}{dt^2} = 0$ and $f(t)$ has a triple root. The self-intersection of the surface gives those polynomials where $f(t)$ has two repeated roots. There is also a tail which gives polynomials which have two complex conjugate repeated roots. Finally the swallowtail point $x = y = z = 0$ corresponds to the polynomial $t^4 = 0$.

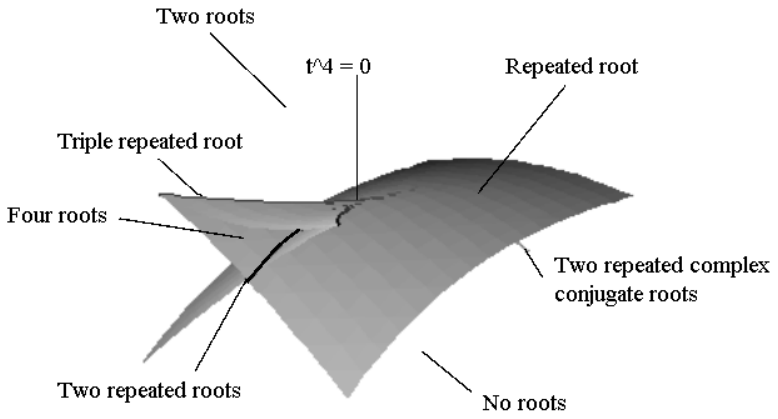
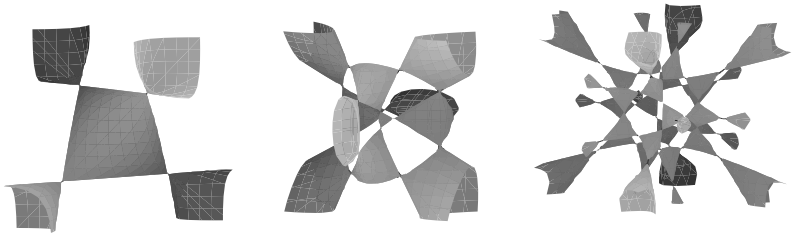


Fig. 12. The discriminant surface for $t^4 + zt^2 + yt + x$ showing the types of roots which can occur.

An interesting area of study is to find low degree algebraic surfaces which contain many nodes [4]. Some examples of these include:

- Cayley's cubic, a cubic surface with the maximum of four nodes, $4(x^2 + y^2 + z^2) + 16xyz = 1$, Fig. 13(a).
- Kummer surfaces, a family of quartic surfaces some of which have 16 nodes $(3 - v^2)(x^2 + y^2 + z^2 - v^2)^2 - (3v^2 - 1)(1 - z - x\sqrt{2})(1 - z + x\sqrt{2})(1 + z + y\sqrt{2})(1 + z - y\sqrt{2}) = 0$, Fig. 13(b).
- Barth's sextic with 65 nodes $4(\tau^2x^2 - y^2)(\tau^2y^2 - z^2)(\tau^2z^2 - x^2) - (1 + 2\tau)(x^2 + y^2 + z^2 - 1)^2 = 0$ where $\tau = (1 + \sqrt{5})/2$, Fig. 13(c).

These prove to be good test cases for the algorithm, which produces good but not perfect representations of the surface.



(a) Cayley's cubic (b) Kummer Surface (c) Barth's sextic

Fig. 13. Algebraic surfaces with many nodes

5 Conclusion

This program can produce good models of many algebraic surfaces including those that contain singularities and it can even find the handle on a cross-cap. It offers considerable advantages over many other algorithms which often miss the singular points completely. The improvements in the polygonization step has considerable improved it performance over previous versions.

The adaptation of the Java interface has considerably improved the usability of the software and it can now display the surfaces on most platforms without any special hardware or software requirements. It can also be used as a stand-alone program on windows machines without needing an open Internet connection. This greatly opens up the potential of the program as it could easily be used as an educational tool in schools and colleges.

The adoption of the client-server system was primarily motivated by ease of porting. The original code was written in C and it would have been a

considerable task to convert this to Java. Adapting the code to run as a server and produce JVX format models was relatively straightforward. This system does have the advantage of making installation trivial. The system does also establish the program as a mathematical server which could potentially be used by other applications and incorporated into larger programs.

Whilst the models are visually good they are not always topologically correct. There are inherent problems with the algorithm as the Bernstein test for zeros of polynomials is too week and can incorrectly identify regions as containing zeros. This is particularly evident in the detection of singular points around the more complicated singularities such as the swallowtail surface. Some improvement could be made by paying more attention to the behaviour of derivatives around the singular points. One possible path for improvement is to use a particle based approach [14,15] where a set of particles surrounding the surface is allowed to converge to the surface. Another method might be to try to determine the type of singularity and use that information to inform the polygonization.

The approach we have taken here can be contrasted with ray-tracing approaches [5,7]. They produce single high quality images from a single direction. The image quality of such algorithms is better than those produce by our algorithm. However producing a 3D model which can be rotated and scaled can give a better feel for the surface and can allow particular points to be inspected.

Oliver Labs has integrated our program with the surf ray-tracer [5]: First the 3D model is calculated and rotated to produce a good view of the surface. The viewing parameters are then passed to surf to generate a high quality image from that direction.

There are several extensions to the package and the algorithm has been adapted to produce algebraic curves in 2D and 3D. The applet can be found online at `http://www.comp.leeds.ac.uk/pfaf/lsmp/SingSurf.html` and `http://www.javaview.de/services/algebraic/`.

References

1. Arnold, V. I., 1981: Singularity Theory. *London Mathematical Society Lecture Notes* **53**
2. Bloomenthal, J., Implicit Surfaces Bibliography, `http://implicit.eecs.wsu.edu/biblio.html`
3. Bruce, J. W., Giblin, P. J., Curves and Singularities (second edition), Cambridge, 1992
4. Endrass, S., Surface with Many Nodes, `http://enriques.mathematik.uni-mainz.de/kon/docs/Eflaechen.shtml`
5. Endrass, S., Huelf, H., Oertel, R., Schneider, K., Schmitt, R., Beigel, J., Surf home page, `http://surf.sourceforge.net`
6. Geisow, A., (1982), Surface Interrogation, Ph.D. Thesis, University of East Anglia

7. Kalra, D., Barr, A. H., Guaranteed Ray Intersection with Implicit Surfaces. *Computer Graphics*, **23(3)** 1989, 297–304.
8. Morris, R. J., A New Method for Drawing Algebraic Surfaces. In Fisher, R. B. (Ed.), Design and Application of Curves and Surfaces, Oxford University Press, 1994, 31-48
9. Morris, R. J., The Use of Computer Graphics for Solving Problems in Singularity Theory, In Hege, H.-C., Polthier, K. (Eds.), Visualization and Mathematics, Springer Verlag, July 1997. 53-66
10. Ning, P., Bloomenthal, J., An Evaluation of Implicit Surface Tilers, IEEE Computer Graphics and Applications, **13(6)**, IEEE Comput. Soc. Press, Los Alamitos CA, Nov. 1993, pp. 33-41
11. Phillips, M., Geomview Manual, Version 1.4. The Geometry Center, University of Minnesota, Minneapolis, 1993.
12. Polthier, K., Khadem, S., Preuss, E., Reitebuch, U., Publication of Interactive Visualizations with JavaView. In Borwein, J., Morales, M., Polthier, K., Rodrigues, J. F. (Eds.) Multimedia Tools for Communicating Mathematics, Springer Verlag, 2001
13. Polthier, K., Khadem, S., Preuss, E., Reitebuch, U., JavaView home page, `http://www.javaview.de/`
14. Saupe, D., Ruhl, M., Animation of Algebraic Surfaces. In Hege, H. C., Polthier, K. (Eds.), Visualization and Mathematics, Springer Verlag, July 1997.
15. Witkin, A., Heckbert, P., Using particles to sample and control implicit surfaces, SIGGRAPH'94, Comp. Graph. Ann. Conf. Ser. (1994), 269-277.