

CONREC

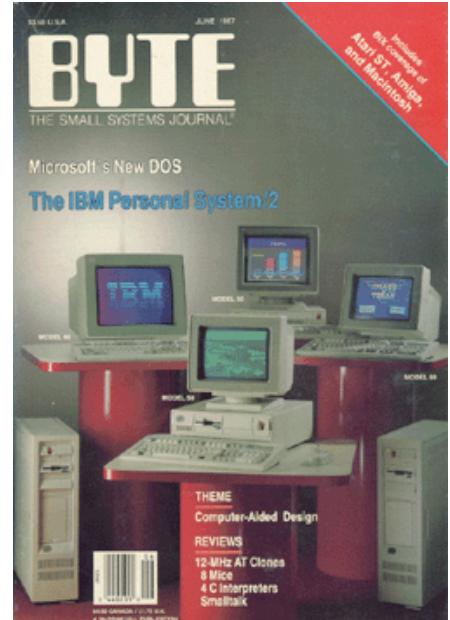
A Contouring Subroutine

Written by [Paul Bourke](#)

July 1987

Source code

- By the author (Paul Bourke)
[C](#), [Basic](#), [FORTRAN-77](#) (the original)
 - Java version: [Render.java](#) and [Conrec.java](#) by Bradley White
 - [Adapted to C++](#) by Nicholas Yue.
 - [Ada version](#) by Gautier de Montmollin.
 - [Visual Basic](#) by James Craig
 - [Delphi translation](#) courtesy of Alexander Weidauer
 - [Visual Basic version](#) by Michelle Smith
-



Introduction

This article introduces a straightforward method of contouring some surface represented as a regular triangular mesh. Contouring aids in visualizing three dimensional surfaces on a two dimensional medium (on paper or in this case a computer graphics screen). Two most common applications are displaying topological features of an area on a map or the air pressure on a weather map. In all cases some parameter is plotted as a function of two variables, the longitude and latitude or x and y axis. One problem with computer contouring is the process is usually CPU intensive and the algorithms often use advanced mathematical techniques making them susceptible to error.

CONREC

To do contouring in software you need to describe the data surface and the contour levels you want to have drawn. The software given this information must call the algorithm that calculates the line segments that make up a contour curve and then plot these line segments on whatever graphics device is available.

CONREC satisfies the above description, it is relatively simple to implement, very reliable, and does not require sophisticated programming techniques or a high level of mathematics to understand how it works.

The input parameters to the CONREC subroutine are as follows :

- The number of horizontal and vertical data points designated iub and jub.
- The number of contouring levels, nc.
- A one dimensional array z(0:nc-1) that saves as a list of the contour levels in increasing order. (The order of course can be relaxed if the program will sort the levels)
- A two dimensional array d(0:iub,0:jub) that contains the description of the data array to be contoured. Each element of the array is a sample of the surface being studied at a point (x,y)
- Two, one dimensional arrays x(0:iub) and y(0:jub) which contain the horizontal and vertical coordinates of each sample

point. This allows for a rectangular grid of samples.

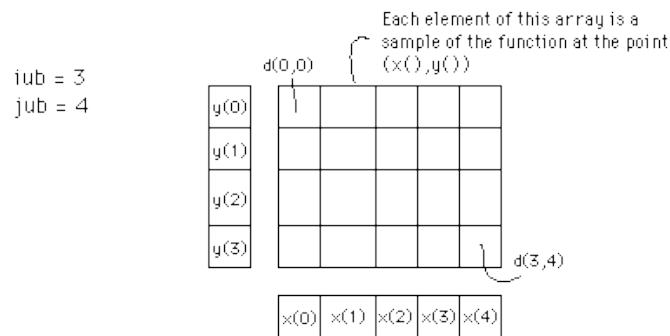


Figure 1 Illustrates some of the above input parameters.

The contouring subroutine CONREC does not assume anything about the device that will be used to plot the contours. It instead expects a user written subroutine called VECOUT. CONREC calls VECOUT with the horizontal and vertical coordinates of the start and end coordinates of a line segment along with the contour level for that line segment. In the simplest case this is very similar to the usual LINE $(x1,y1)-(x2,y2)$ command in BASIC. See the source code listing below.

Algorithm

As already mentioned the samples of the three dimensional surface are stored in a two dimensional real array. This rectangular grid is considered four points at a time, namely the rectangle $d(i,j)$, $d(i+1,j)$, $d(i,j+1)$, and $d(i+1,j+1)$. The centre of each rectangle is assigned a value corresponding to the average values of each of the four vertices. Each rectangle is in turn divided into four triangular regions by cutting along the diagonals. Each of these triangular planes may be bisected by horizontal contour plane. The intersection of these two planes is a straight line segment, part of the the contour curve at that contour height.

Depending on the value of a contour level with respect to the height at the vertices of a triangle, certain types of contour lines are drawn. The 10 possible cases which may occur are summarised below

- All the vertices lie below the contour level.
- Two vertices lie below and one on the contour level.
- Two vertices lie below and one above the contour level.
- One vertex lies below and two on the contour level.
- One vertex lies below, one on and one above the contour level.
- One vertex lies below and two above the contour level.
- Three vertices lie on the contour level.
- Two vertices lie on and one above the contour level.
- One vertex lies on and two above the contour level.
- All the vertices lie above the contour level.

In cases a, b, i and j the two planes do not intersect, ie: no line need be drawn. For cases d and h the two planes intersect along an edge of the triangle and therefore line is drawn between the two vertices that lie on the contour level. Case e requires that a line be drawn from the vertex on the contour level to a point on the opposite edge. This point is determined by the intersection of the contour level with the straight line between the other two vertices. Cases c and f are the most common situations where the line is drawn from one edge to another edge of the triangle. The last possibility or case g above has no really satisfactory solution and fortunately will occur rarely with real arithmetic.

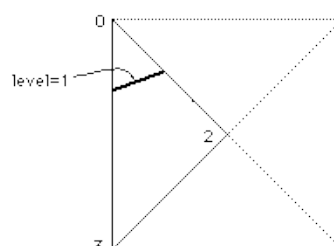
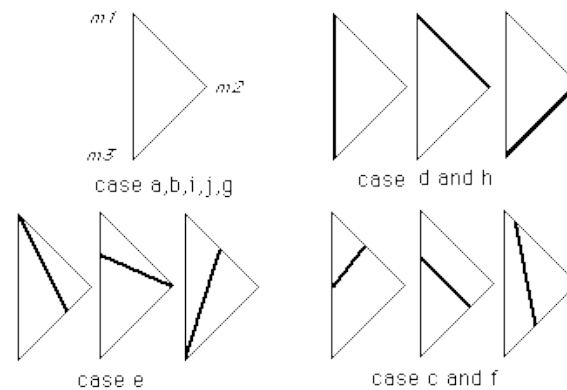


Figure 2

Summarises the possible line orientations.

Example

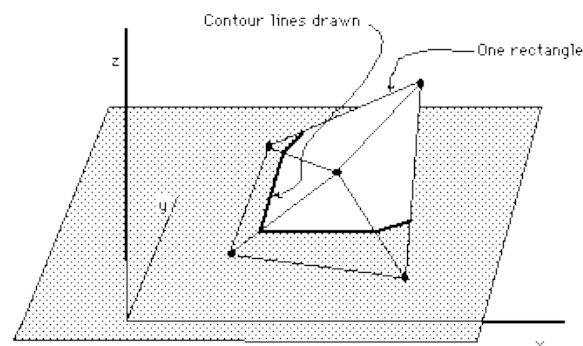
As a simple example consider one triangle with vertices labelled $m1, m2$ and $m3$ with heights 0, 2 and 3 respectively

**Figure 3**

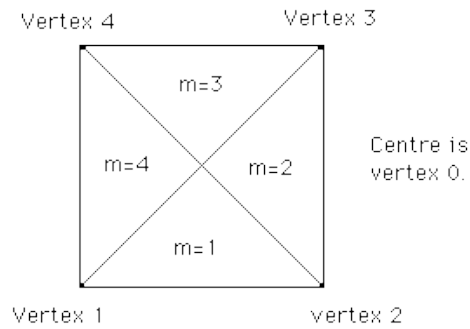
To calculate where a contour line at a height of 1 should be drawn, it can be seen that this is case f described earlier. Level 1 intersects line segment $m1-m2$ half the way along and it intersects line segment $m1-m3$ one third of the way along. A line segment is drawn between these two points. Each rectangular mesh cell is treated this way.

Subroutine

In summary, CONREC takes each rectangle of adjacent data points and splits it into 4 triangles after choosing the height at the centre of the rectangle. For each of the triangles the line segment resulting from the intersection with each contour plane. A routine is then called with the starting and stopping coordinates of the line segment.

**Figure 4**

An attempt is made at optimization by checking first to see if there are any contour levels within the present rectangle and second that there are some contour levels within the present triangle. The indices i and j are used to step through each rectangle in turn, k refers to each contour level and m to the four triangles in each rectangle.

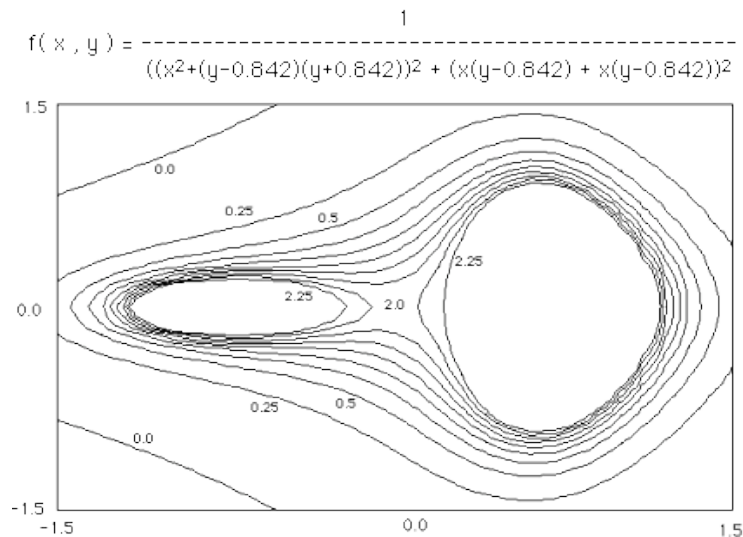
**Figure 5**

Some of the notation used for identifying the rectangles and triangles in the subroutine.

Note that for large arrays the whole data array need not be stored in memory . Since the algorithm is a local one only requiring 4 points at a time, the data for each rectangle could be read from disk as required.

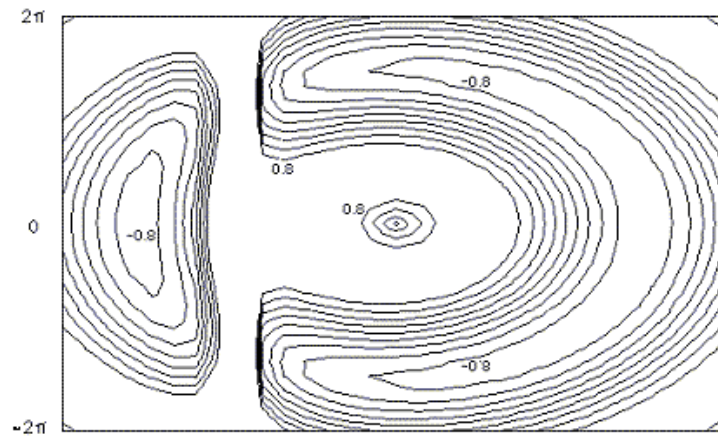
Example 1

Contour map and the following function

**Example 2**

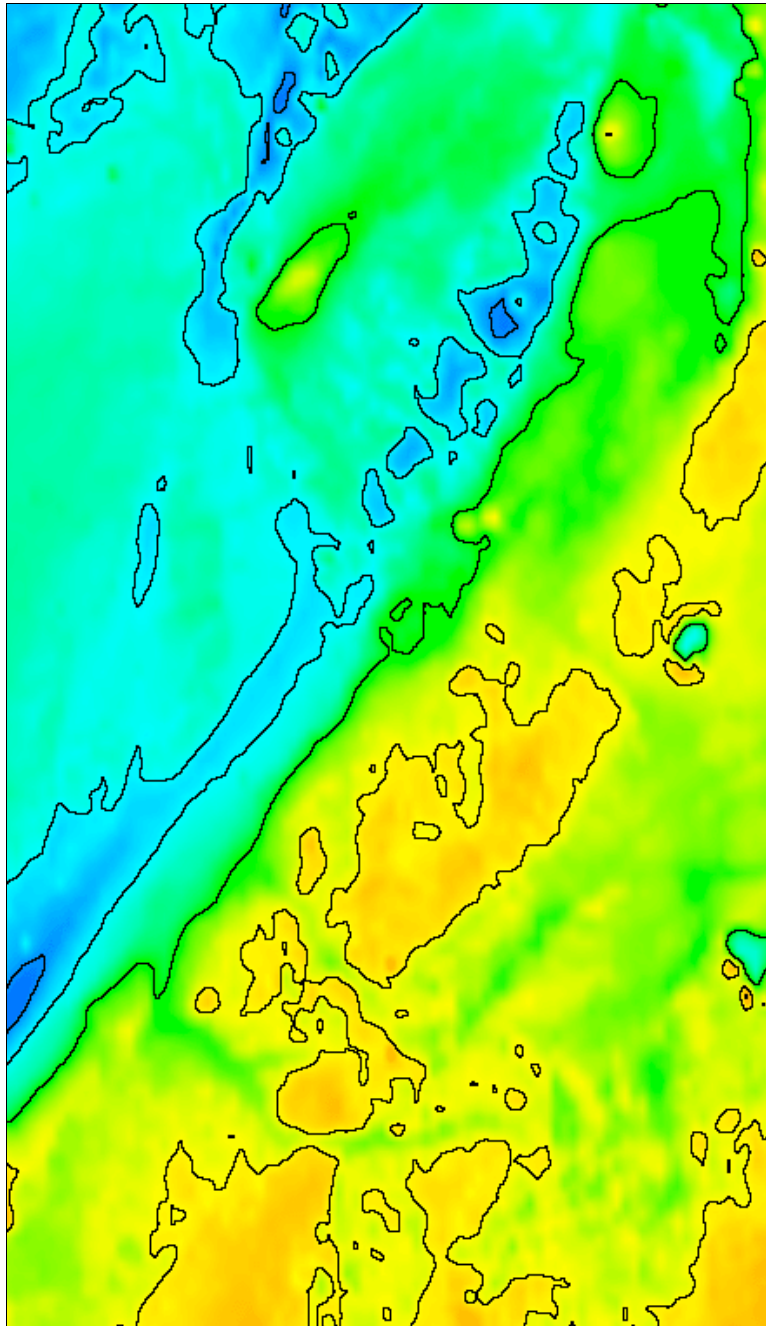
Contour map and the following function

$$f(x, y) = \sin((x^2 + y^2)^{1/2}) + \frac{1}{((x - c)^2 + y^2)^{1/2}}$$



Example 3

This is a more "real life" example where a C version of CONREC has been used to contour a piece of landscape, the result is given below.



Images from the BYTE magazine version

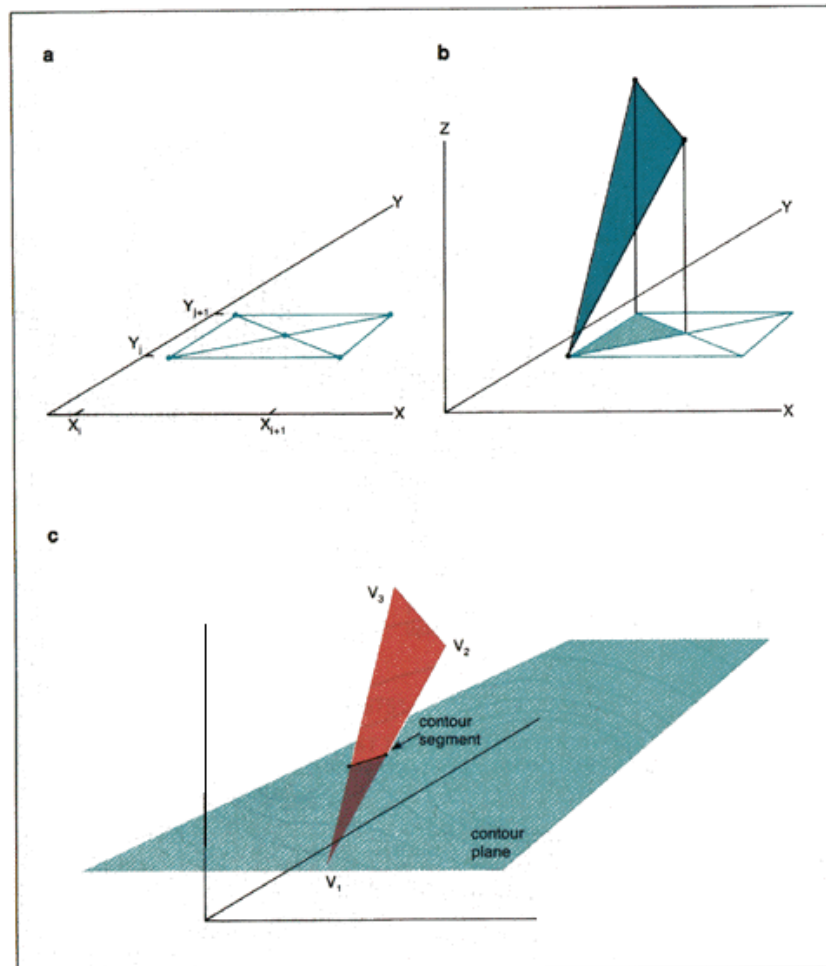


Figure 2: The data points are analyzed in groups of four contiguous points. These rectangles are divided into triangles (a). The centerpoint C 's coordinates are interpolated, giving three triangles in three-dimensional space (b shows one of them). The intersection of these triangles with the contour plane determines the location of the contour segments (c).

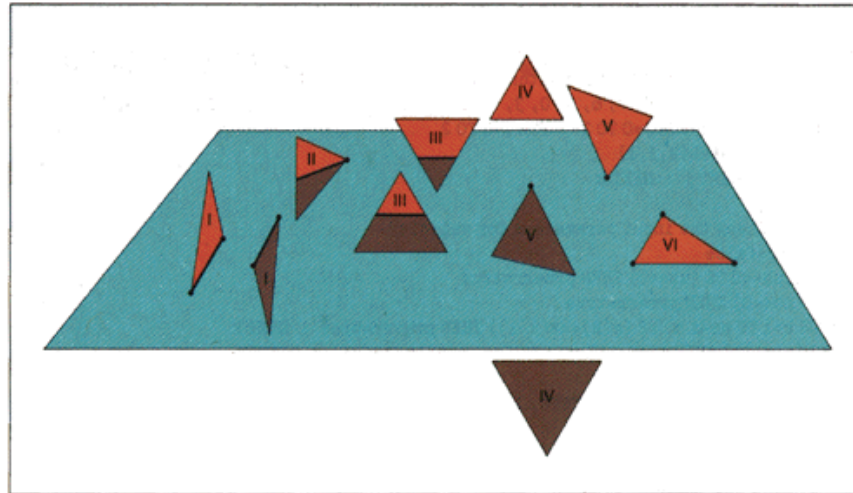


Figure 3: Six types of intersections between triangles and the contour plane; only three types result in contour line segments.

Table 1: Classification scheme for the intersections between a triangle and the contour plane.

Type	Intersection of triangle and the contour plane	Point locations with respect to the contour plane
I	Line through one side of the triangle	1 above, 2 on plane 1 below, 2 on plane
II	Line bisecting the triangle through one vertex	1 above, 1 below, 1 on plane
III	Line bisecting the triangle through two sides	1 above, 2 below plane 2 above, 1 below plane
IV	No intersection	3 above plane 3 below plane
V	Point intersection	2 above, 1 on plane 2 below, 1 on plane
VI	Plane intersection	3 on plane

Note

On occasion users have reported gaps in their contour lines, this should of course never happen. There is however a pathological case that all local contouring algorithms suffer from (local meaning that they only use information in the immediate vicinity to determine the contour lines). The problem arises when all four vertices of a grid cell have the same value as the contour level under consideration. There are a number of strategies that can be employed to overcome this special event, the correct way is to consider a larger region in order to join up the contours on either side of the problem cell. CONREC doesn't do this and just leaves the cell without any contour lines thus resulting in a gap. This special case essentially never happens for real values data, it is most commonly associated with integer height datasets. The simplest solution is to offset the contour levels being drawn by a very small amount.

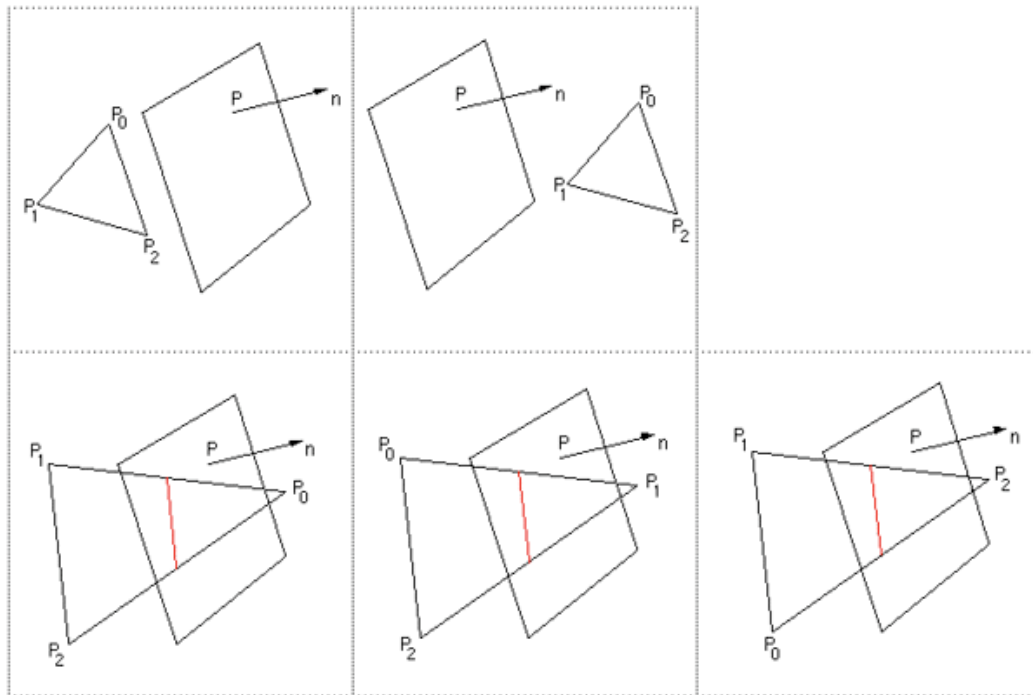
Contouring Facet Based Models

Written by [Paul Bourke](#)
February 1997

This is a short note describing a method for contouring a facet based model, that is, drawing line segments along the intersection of the facets with the contouring plane(s). It is not assumed that the contour plane is parallel to the x-y plane, indeed a totally general contour plane description is employed.

The facet is described by its three vertices, P_0 , P_1 , P_2 . The plane is described by its normal \mathbf{n} and a point on the plane P . The routine given below returns 0 if there is no intersection of the facet with the contour plane. It returns 2 and the two vertices of the intersection line if the facet does intersect the contour plane.

As reflected in the source below, there are 5 basic cases to consider. The first two are when all the vertices of the facet are on one side of the contour plane. The other 3 cases involve finding out which of the three facets is on a side by itself.

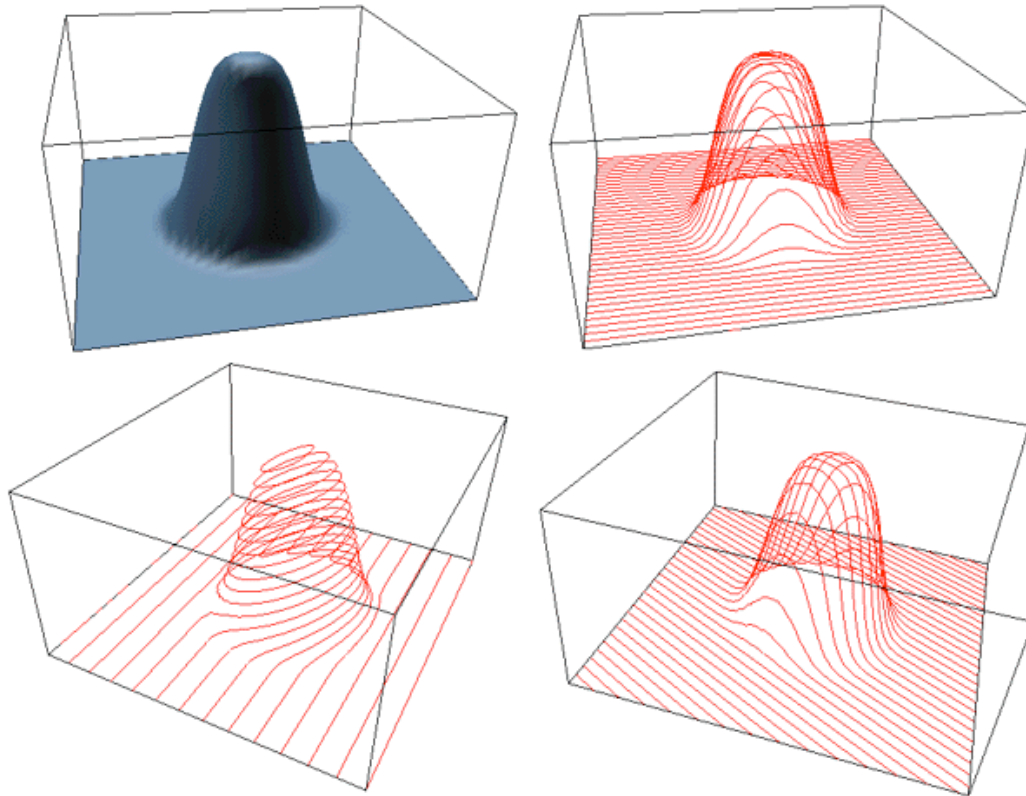


Notes

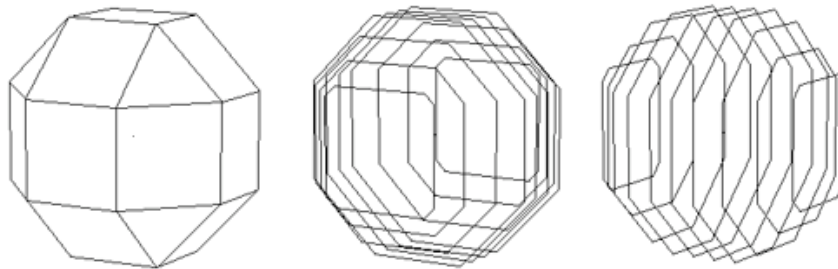
- If the polygons making up the model are more complex than the simple 3 facet ones considered here then they need to be split into simpler ones before using this algorithm, there are "standard" ways of doing this. For example 4 vertex facets can be split into 2 triangular facets by dividing along any two apposite vertices. Similarly, convex polygons can be triangulated by forming triangular facets with each edge and the centroid of the polygon. It gets a bit more difficult with complicated convex polygons.....
- The most common height contours are achieved by simple setting the contour plane normal \mathbf{n} to (0,0,1) and the point on the contour plane $\mathbf{p0}$ to (0,0,contourlevel).

Examples

The following examples result from contouring a 2D Gaussian with a range of different orientated contour planes.



The technique is obviously not limited to traditional contouring of "height surfaces". Closed forms can equally be contoured. The following is a rhombicuboctahedron with contours perpendicular to the x axis and y axis.



Source Code

The way the above code might be called in order to draw contour lines at multiple levels through a model containing a large number of polygons might be as follows

```
Determine and create the contour plane "n"
  For each contour level
    Set the value of p0 appropriately
    For each polygon in the model
      Possibly triangulate the polygon if it has more than 3 vertices
      For each triangular polygon
        Call ContourFacet()
        If there were 2 vertices returned draw the line segment
      End for
    End for
  End for
```

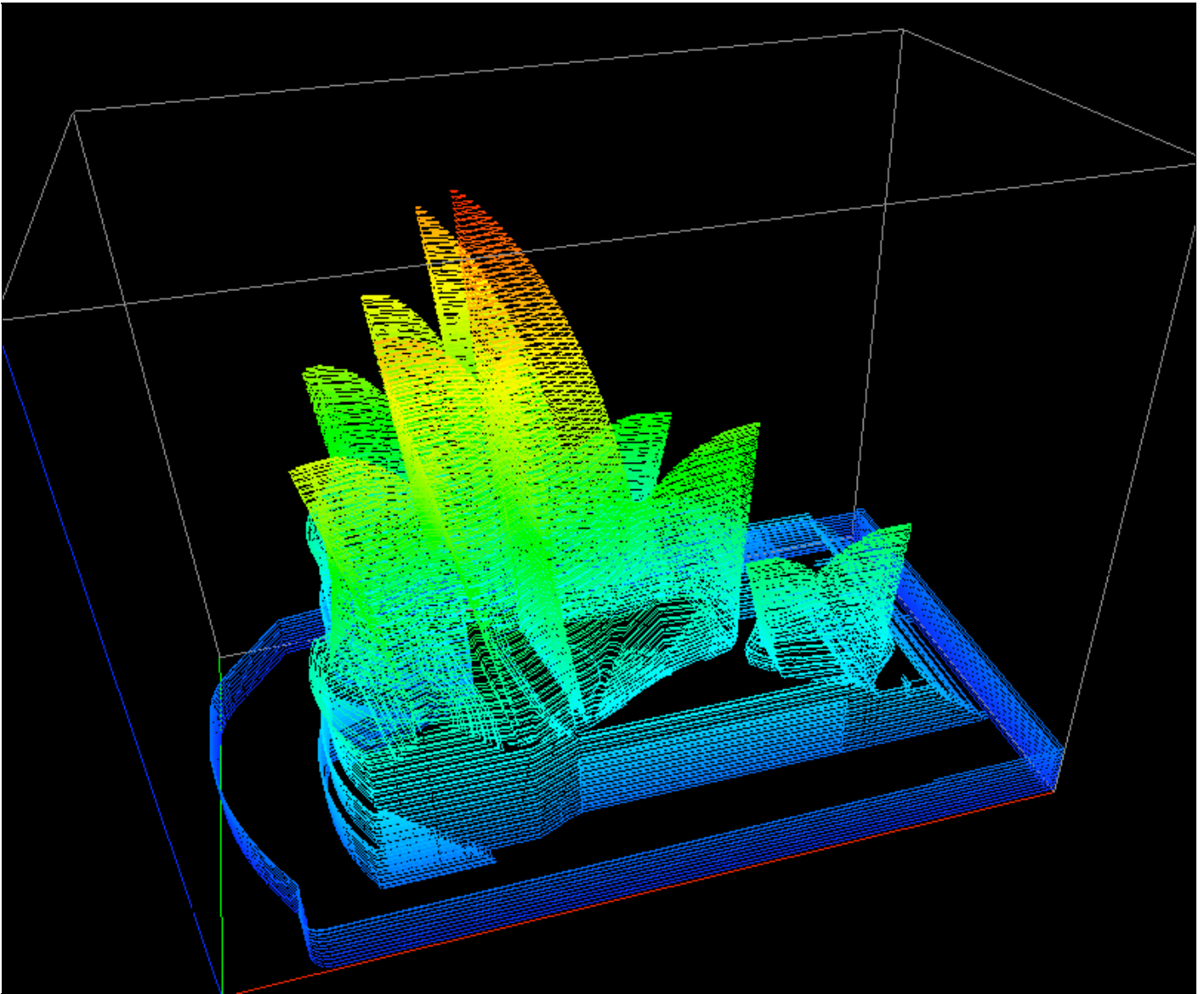
For the simpler case of height contours: [source code](#)

Practical example

Many rapid prototyping machines build models from a number of contour slices, additionally there is a standard file format called [stl](#) for that sort of work which consists of simply triangular polygons. Creating instructions for a contour based rapid prototyping

machine requires that the triangles representing the objects to be constructed are contoured, possibly along an arbitrary axis.

In this example the model is built in miniature in a photonics laboratory (Swinburne University). The model was built in AutoCAD, exported in the STL format, and sliced (129 slices) using software based on the above techniques. The more challenging aspect of this project is dealing with the less than perfect data from AutoCAD, such as the removal of duplicate edges and closing of contour loops.



The resulting physical model viewed through a microscope is shown below, the white bar at the bottom is 10 microns (1/100 mm) long making the total length of the model about 0.08mm.

