A Simple Procedure to Generate Curves and Surfaces

Alan Sutcliffe
4 Binfield Road
Wokingham
RG40 1SL, UK
E-mail: nsutcliffe@ntlworld.com

Abstract

A method is presented of synthesizing irregular but smooth curves for display at screen resolutions of the order of 1200 by 1200 pixels. The method uses the addition of differences of differences of differences in one coordinate for unit increments in the other coordinate. The curves are extended laterally by simple shading to produce the appearance of smooth forms in 3-dimensions. Drawing in the XOR mode gives unexpected benefits in this context. There is a brief note on the procedural approach to composition compared with conventional methods. A final note describes the operation of XOR applied to grey-scales and colours.

Background

A 2-dimensional density plot (figure 1) of a mathematical function suggested that displays with the appearance of smoothly shaded 3-dimensional forms could easily be made from linear curves. The first notion was to make the curves from smoothly joined segments of circles and ellipses, but this looked awkward to implement.

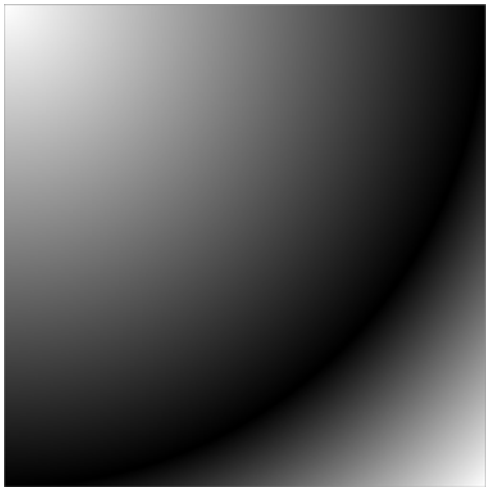


Figure 1: *A density plot.*

A simple algorithm starting with differences of the third order came to mind, together with a straightforward way of extending these into areas using shading to give the appearance of 3-dimensional forms. At this stage there was no intention to make shapes of any particular sort and only abstract forms were envisaged, with a small hope that some might suggest human body parts.

From making figure 1, as part of a mathematical paper, and producing the three art works shown here took three days working about half time. That is less than half the time it has taken to produce this paper, which is not so much a description of the headlong process of conception, coding, trials, modifications, and graphic manipulations, but more reflection and analysis of what was done. The mathematical paper was not completed.

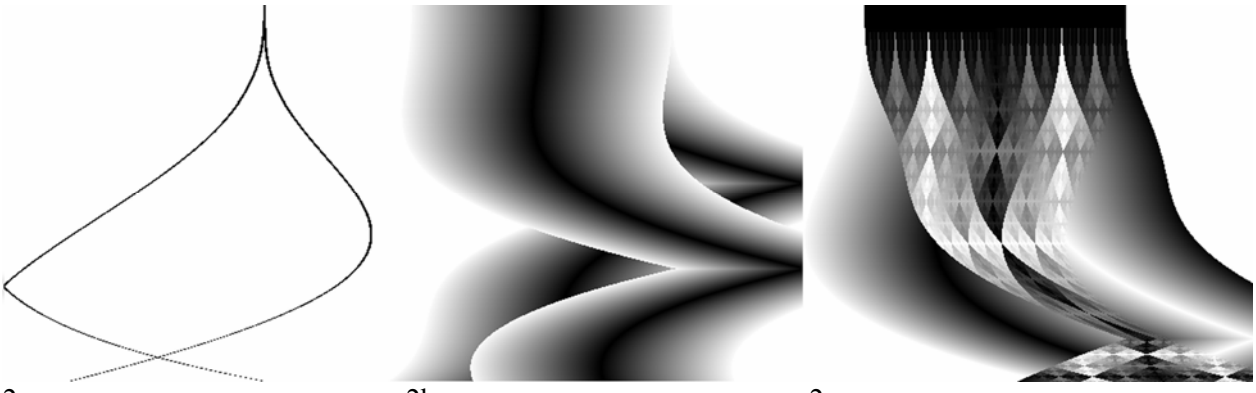
Synthesizing Curves and Shading

A digital curve can be characterised by the changes in the x coordinate for fixed (unit) changes in the y coordinate. Such a curve is monotonic, having only one x value for each y value. Given an initial value, the x coordinates can be determined from the differences between successive values of x, the first order differences, d. These can in turn be derived from the second order differences, dd, and so on. Starting with the third order differences, ddd, the values of x can be derived, given initial values for dd, d, and x. This is shown in the scheme below where the initial values are set to zero. Each subsequent value of dd, d, and x is the sum of the element above it and the element above and to the left.

ddd	dd	d	x
	0		
r_1		0	
	r_1		0
r_2		r_1	
	r_1+r_2		$r1$
r_3		$2r_1+r_2$	
	$r_1+r_2+r_3$		$3r_1+r_2$

and so on.

This is the method used in the generation of curves in the work presented here. The values of ddd are generated as small random numbers, r_1, r_2, r_3 ... , in a range $-a \leq r \leq a$, where a is positive and less than 1. Since the values of ddd can be positive or negative, so can the values of dd, d, and x. Third order differences were chosen initially as the starting point by intuition, as likely to produce smooth curves in an image of 1200 by 1200 pixels, and this they did. Examples of curves using this method are shown in figure 2. Tests have not been done starting with differences of higher or lower order.



2a 2b 2c

Figure 2. *Two lines:* 2a *plain,* 2b *shaded, and* 2c *shaded with XOR.*

Each graphic in figure 2 uses two curves. As is usual in screen displays the origin (0,0) is at the top left. The curves are drawn pixel by pixel, with grey level 0, which is black. The curves are started at the top centre of each graphic. In figures 2b and 2c shading is added for each pixel of the curve by drawing, on each side of it and with the same *y* value, pixels with grey level 1 to 255, going from near-black to white. In figure 2b shading for the second line is drawn over that for the first. In figure 2c it is drawn using XOR, sometimes going over existing shading where the harlequin patterns appear, sometimes going directly onto the white background of the screen, causing the plotted values to be inverted, darker for lighter. This accounts for the soft white edge of the first shading still visible on the left of figure 2c and the hard black edge from the second shading on the right.

The differences and plotted coordinates are not stored. Later values depend on earlier ones only through the accumulation of differences and incrementing of the value of *x* for each new point on the curve. The calculation and plotting of each curve with its shading takes two to three seconds with the program running in the Visual Basic environment from semi-compiled code on a personal computer of modest power. This could be improved by an order of magnitude using compiled code from a more efficient language on a more powerful personal computer. The program used for the examples in figure 2 is less than 30 lines of code, excluding comments. An extra few lines are used for two of the artworks to plot curves across the display as well as down, done simply by exchanging *x* and *y*.

Boundary Conditions

In figure 2 when a line reaches the edge of the display area it is reflected. This is achieved simply by reversing the sign of *d*, the first order difference. This action was chosen to maintain interest in the picture. Other possible actions are:

- end the line and any shading
- let the line run out of the display area unseen, with or without any associated shading being shown
- fix the line to run along the boundary
- start another curve, from the boundary or elsewhere, with the next value of *y*

A different sort of method would be to reduce the values of the first order differences as the curve approached the boundary, preventing the curve from reaching the boundary. But this could lead to *x* becoming static, giving a less interesting curve. Alternatively an arc of a circle or ellipse could be introduced to turn the curve away from the boundary, but matching its tangent to that of the curve where they join, to give a smooth transition, might be awkward.

Art Works

In the three works shown below, the curves were started in the centre of the image and generated in opposite directions simultaneously to give symmetry about a vertical line through the starting point.

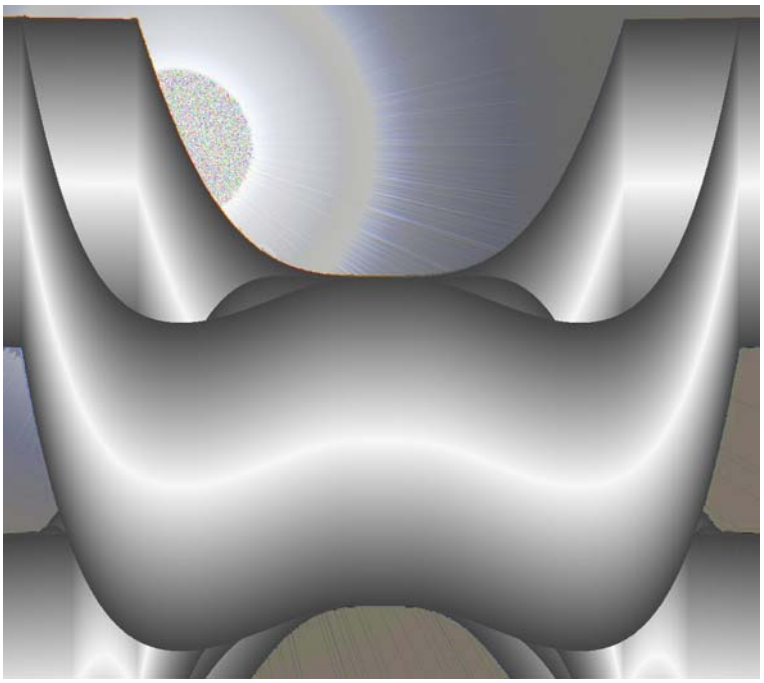


Figure 3: *Central part of* Sunrise Torso.

In *Sunrise Torso* (see figure 3) five curves were overlaid and the result rotated by 90°. A sunburst was added and a title invented.

In *Screaming WarBot* (see figure 4) symmetrical curves were generated alternating vertical and horizontal generation and using the XOR drawing mode - see the section below. About 20 curves were generated in each direction. There were two wholly unexpected effects. In the central area the curves were so near together, having all started at the same central point, that the harlequin pattern largely disappeared and only dark tones remained. These were made more visible by increasing the brightness in this area. The second effect was that the crowding of subdivisions in the region around the centre has the effect of 3-dimensional shaping of a snout protruding from the flatter surrounding area.

Far Horizon, a section of which is shown in figure 5, was made by rotating *Screaming WarBot*, cutting it in two, rotating the lower half and joining these two parts side by side to make a wide format. An edge-finding filter was then applied and the grey-scales reduced to three levels.

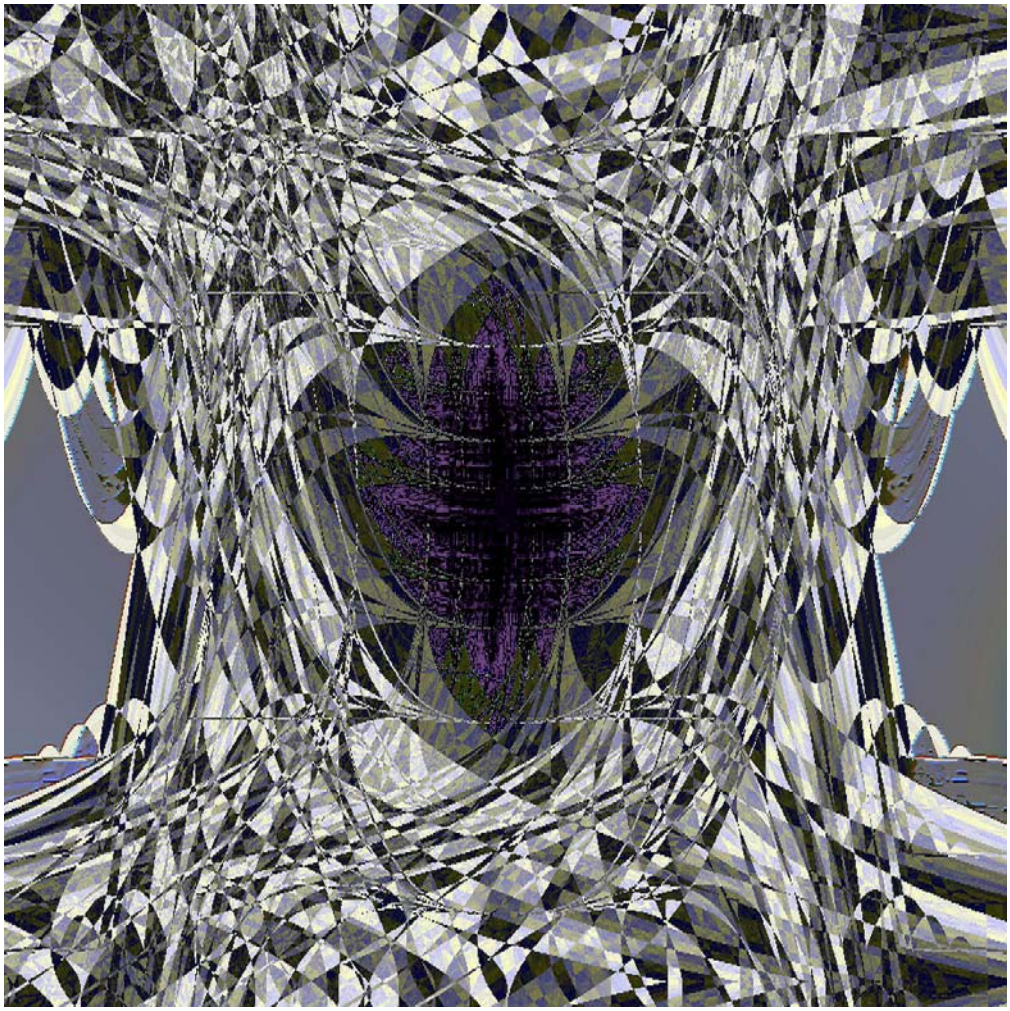


Figure 4: *Central part of* Screaming Warbot.

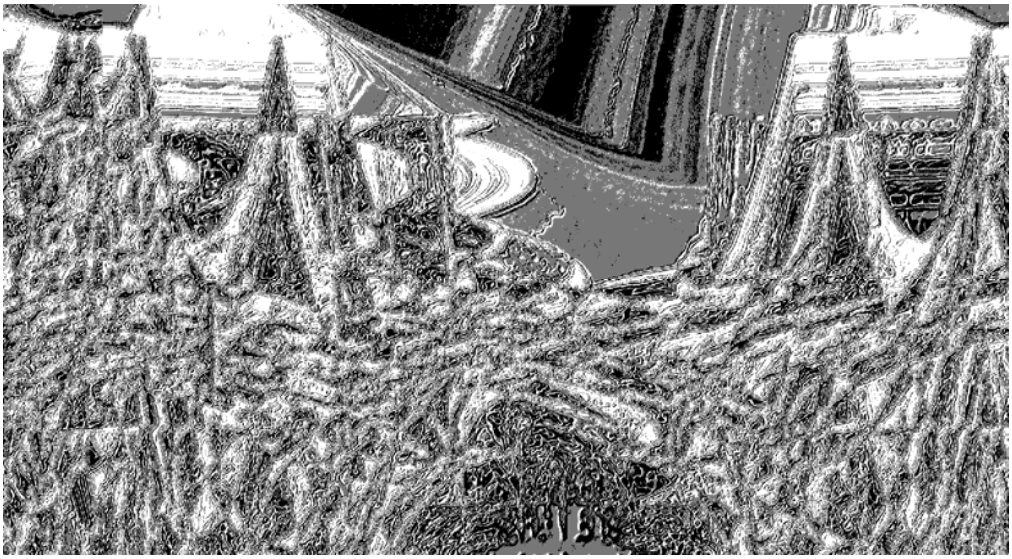


Figure 5: *A section of* Far Horizon.

Procedures and Arts

Procedural art was best characterised by Herbert Brün in 1969 [1].

> It is one thing to aim for a particular timbre of sound and then to search for the means of making such sound and timbre audible. It is another thing to provide for a series of events to happen and then to discover the timbre of the sounds so generated. In the first case one prefers those things to happen that one wishes to hear; in the second case one prefers to hear those events one wishes would happen. These are not just two different approaches to the composition of music but also two different political attitudes.

This applies to graphics and other arts just as much as to music. It challenges the commonly cherished view that the artist, in whatever medium, conceives an outcome and sets out to realise it, knowing what she is doing. And this is then called communication. In my view art evokes, just as a waterfall or the night sky can: it does not communicate. I have found that it is best for me not to know what I am doing: the other approach generally leads to failure. I want the outcome to be a surprise. I do not prescribe for others.

Gustav Metzger [2] in a paper presented at Computer Graphics 70 (sadly, not in the Proceedings) had another insight that goes beyond the arts. He writes, in a paragraph on Self-organising Drawings:

> We need machines [that] tell us things we do not know, and cannot understand. That must be one of the goals of computer art.

With the procedural approach I have produced works that I could not have conceived and implemented using the conventional model of making art. I do not hold strictly to the procedural principle, but sometimes allow myself to use conventional means such as a paint package to elaborate a work, as well as for routine tasks such as rotation and contrast enhancement. It is as though I were a vegetarian wearing leather shoes.

XOR Applied to Grey-scales and Colours

In Visual Basic and in some other languages XOR is both logical/arithmetic operator and a drawing mode. In both cases it operates in the standard way found in mathematical logic according to the truth table shown in Table 1.

A / B	0	1
0	**0**	**1**
1	**1**	**0**

A XOR $B = 0$ when $A = B$
A XOR $B = 1$ when $A \neq B$

Table 1: *XOR truth table.*

Applied to strings of bits (which is how all entities are stored in conventional computers) XOR operates on each bit position independently, so that 0011 XOR 0101 = 0110: in decimal 3 XOR 5 = 6. The array below shows the values of A XOR B for $0 \le A, B \le 15$: an intriguing pattern.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4
12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table 2: *Values produced by XOR.*

Colours in Visual Basic are represented by a data type RGB(r, g, b) with $0 \le r, g, b \le 255$ and where r, g, and b represent the levels of red, green, and blue. Internally these are stored as three 8-bit values in a 32-bit word with the top 8 bits all zero. Grey-scale colours are simply those with $r = g = b$ and the XOR of two such colours will also be grey. The values in table 2 represent the dark shades of grey in the top left corner of figure 6, which are hardly distinguishable in print. They are the 16 darkest shades in the scale of 256 shades from black to white.

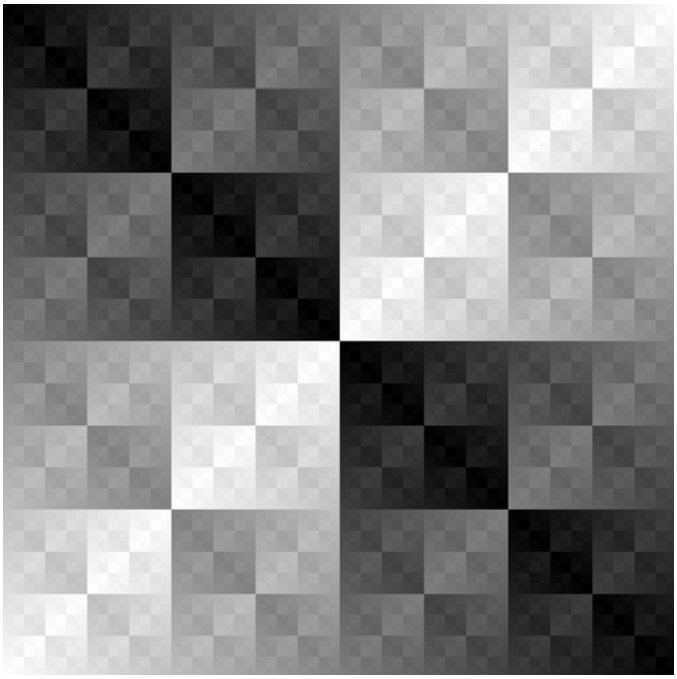


Figure 6: *The array of all XOR combinations given 256 levels of grey in each direction.*

Although there are 256 levels of grey from black to white in Figure 6, they cannot all be distinguished. This is not only because of the limitations of the screen or printer but also because the change from one level to the next, for example from RGB(99, 99, 99) to RGB(100, 100, 100), is at or below the least perceptible difference.

Notice the self-similarity between parts and the whole of the table 2. Consider the square sub-tables starting at the top left and having 2^n rows and columns for $1 \leq n \leq 4$. Each of these has one diagonal of zeros and the other diagonal of the maximum value of $2^n - 1$ for that table. The tables are symmetrical about each diagonal. $n = 1$ gives the truth table of table 1. Figure 6 is simply the equivalent table for $n = 8$ represented in graphic form with one pixel per element.

Such tables of 2^n x 2^n elements can also be formed by the following rules, with the rows r numbered 1 to n:

- for $r = 1$, row 1 has the elements 0 to $2^n - 1$, in order
- for $r > 1$, divide row $r - 1$ into blocks of m elements, where m is the largest power of 2 dividing $2(r - 1)$, then reverse the order of the elements within each block to give row r

For example, row 2 is divided into blocks of 4 elements:

1 0 3 2 5 4 7 6 9 8 11 10 13 12 15 14

and the elements of each block reversed to give row 3:

2 3 0 1 6 7 4 5 10 11 8 9 14 15 12 13

In the first images made with XOR, tints of colour were seen. The explanation was not clear, though once realised, it was simple. The grey background of the Visual Basic form on which the images were being draw was not a true grey-scale shade as there were small differences in the values of red, green, blue, RGB(212, 208, 200). In some areas the operation of XOR exaggerates these differences.

XOR has another property, though it is not manifest in these graphics: (A XOR B) XOR $B = A$. This means that if an image is drawn over another with XOR and then drawn over it a second time, the original image is restored.

References

[1] Herbert Brün, *Infraudibles*, in *Music by Computers*, edited by von Foerster and Beauchamp, Wiley, 1969.

[2] Gustav Metzger, *New Ideas in Plotter Design, Construction and Output*, Computer Graphics 70, 1970, but not in the Proceedings.