THE DESIGN OF AN EFIS ATTITUDE INDICATOR FOR A FLIGHT SIMULATOR

ICAS-94-8.1.4

S J Handley and D J Allerton
Department of Avionics
College of Aeronautics
Cranfield University, UK

Abstract

A significant proportion of the work load of the symbol generator in an aircraft electronic flight instrument system (EFIS) is in maintaining the display of aircraft attitude. Substantial processing is required to compute the filled sky and ground regions within the attitude indicator and typically, for EFIS displays, the filling of these regions is achieved by special-purpose hardware within the symbol generator dedicated to this function. This paper presents an algorithm which minimises the computations inherent in this problem and which simplifies the hardware required to maintain the real-time indication of aircraft attitude in an EFIS display.

The characteristics of the spatial adjustment resulting from changes in pitch and roll are analyzed and classified into twelve readily detectable categories. The analysis indicates that there are six primary cases to consider, for which the ordering of the intersection points of the old and new horizon lines with the instrument boundary is used to determine the appropriate filling strategy. The remaining six cases cover large excursions in aircraft pitch where the horizon line is outside the instrument boundary. For all cases, the incremental adjustment from the previous frame can be made by filling at most two polygons selected from just three distinct types. A further efficiency of the method derives from filling the polygons using only vectors drawn between points on the instrument boundary.

The performance of the algorithm is presented in the form of results obtained from emulation of the display on a PC which confirm its efficiency. The algorithm has been implemented on a flight simulator at Cranfield University, where the anticipated improvement in the real-time display performance has been demonstrated.

Introduction

In recent years, the electro-mechanical instruments in many aircraft cockpits have been replaced by electronic flight instrument system (EFIS) displays in the form of colour CRT monitors.[1] These displays operate on a similar principle to the graphic displays used in personal computers and workstations; images are formed with raster scan lines of the CRT; the intensity or colour of points (pixels) along the lines is controlled by the binary contents of a video memory or frame store. The bit patterns are written by a graphics processor; the frame store is continuously accessed to convert the bit patterns to a video signal; no further action is necessary to refresh the image.

The visual animation of the display is achieved by altering the contents of the frame store at a frequency which is compatible with the required image refresh rate. In practice, to avoid noticeable flicker, aircraft displays need to be refreshed at least 50 times per second which implies that all graphical changes to the bit image in the frame store must be completed within a 20 ms frame time.

Under visual flying rules (VFR), a pilot uses the horizon as a reference for the attitude of the aircraft. Under instrument flying rules (IFR), an attitude indicator, provides a pilot with a synthetic display of the horizon to indicate the aircraft attitude in pitch and roll. The electro-mechanical version of this instrument, also known descriptively as an artificial horizon, comprises a small gyro-stabilised sphere marked with an equatorial horizon and lines to indicate angles of roll and pitch. The hemispheres above and below the horizon line are coloured to represent sky and ground. In EFIS displays, the pitch and roll angles are derived from aircraft sensors and a computer-generated two-dimensional image of the electro-mechanical instrument is displayed. (Fig. 1)

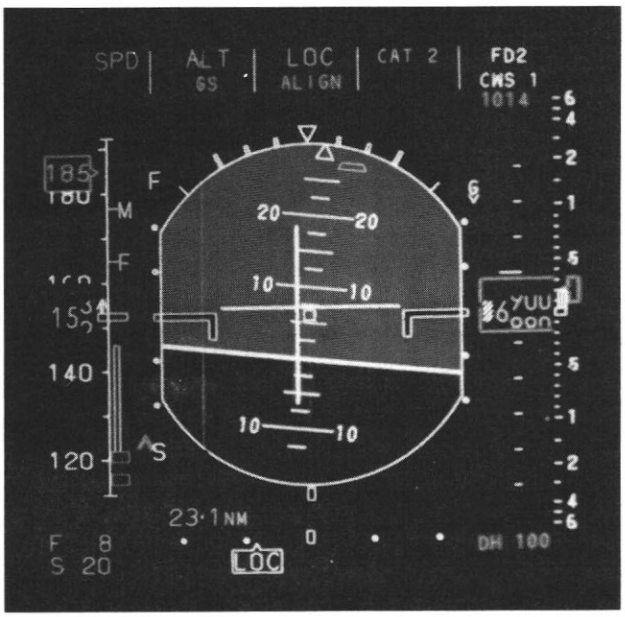

Fig. 1. A typical EFIS attitude indicator. (Courtesy of Smiths Industries, Civil Systems, Cheltenham, UK)

In aircraft, as in other real-time systems, it is essential that changes to displays are implemented at sufficient rates to avoid the introduction of any perceived latency which a pilot might interpret incorrectly as an actual system response. For the majority of aircraft instruments, this requirement does not pose a significant problem; a processor can derive the changes to the instrument and, in conjunction with a graphics processor, perform the graphical operations to re-draw the display in its updated form. For many instruments, the only graphical operation is the simple rotation of a line or narrow pointer to a new position. However, changes to an attitude indicator can require substantial computational and graphical operations. This paper discusses an algorithm to minimise the computation inherent in the presentation of an attitude indicator in a real-time EFIS display.

The Graphical Display of Aircraft Instruments

There are principally two approaches to implementing graphical changes in real-time. Firstly the display could be completely redrawn. This option is very demanding, not only in terms of the two-dimensional geometric processing inherent in aircraft displays but also in the number of pixels to be written to the framestore within the refresh period. A 1000 by 1000 pixel display refreshed at 50 Hz implies a frame store update rate of the order 50 million bits per second. This is clearly inefficient when the changes only affect a small part of the display. The alternative approach is to modify the contents of just those parts of the framestore corresponding to the elements of the image that have changed. While this method is appealing, there are several disadvantages.

It is essential to maintain an accurate record of the graphical state of the display - this ensures that graphical changes are only implemented for objects which have altered since the previous frame, but necessitates the provision of complicated and possibly large data structures to provide a consistent record of all objects within a display. A display object is moved by erasing it at its previous position and then re-drawing it at its new position. In effect, this doubles the number of graphical operations. Where objects overlap, it may be difficult to avoid erasing parts of other objects when a particular object is moved. While this deficiency can be overcome by the provision of prioritised colour planes which facilitate the allocation of objects to planes, this form of hardware solution is not always available and its software counterpart may require complex data structures in order to minimise the total amount of re-drawing. In order to maintain a real-time display, the software must cater for worst-case situations even though typical changes may only necessitate redrawing a few objects. The worst-case situation may be no different from totally redrawing the display.

A simple resolution of the screen management problem is available in many graphic processors where the bits written to the framestore can be combined with the existing contents for the pixel using the logical exclusive-or operation. This mitigates the need to retain a record of pixel contents when moving some graphical object. This approach may be suitable for monochrome or vector-based displays, but the large areas of spurious colour, which would occur with two-dimensional filled objects, render this approach unacceptable for the attitude indicator.

An attitude indicator comprises both static and dynamic graphical items. Of the dynamic objects, whose position may change from frame to frame, bank lines move with displacement in roll angle, pitch lines move with displacements in pitch and roll. Although there are relatively few graphical objects to be re-drawn on an attitude indicator, a change in pitch and roll can necessitate large areas of the ground and sky regions being refilled.

With the exception of the solid colour-filled regions, the objects are represented as lines or vectors which undergo translation and rotation from frame to frame. The graphical operations on these vectors are straightforward in two senses. Firstly, the two-dimensional computations of rotation and translation are simple and secondly, the number of pixels erased and written is relatively small. Currently, drawing rates of $1\mu s$ per pixel are achievable and moreover, rendering the vector (writing the pixels to the framestore) may be accomplished independently of the processing of the vector coordinates. In some systems, graphical objects are entered into a display list and rendered autonomously. In other systems, knowing the rendering speed and the processor performance, it is possible to interleave the periods of rendering with vector processing in an intelligent manner.

The rendering of the sky and ground regions poses a much more profound problem for two conflicting reasons. In order to minimise the latency associated with framestore access, the number of pixels written to the frame store should be minimised, and yet computing the precise geometrical changes to the two areas is demanding.

Objectives

This paper addresses the problem of drawing the filled regions of an attitude indicator, and moving the boundary between them, in real-time. Although EFIS attitude indicators are often rounded in appearance, it is straightforward to exclude the writing of pixels outside a predefined boundary either by hardware of software clipping. The problem can be simplified, therefore, by considering the problem of adjusting the two trapezoidal

regions of sky and ground within a square. The boundary between these two regions is the actual artificial horizon (or zero-degree pitch line), often overlaid as a white line.

The objectives are three-fold: The first objective is to attempt to adjust the position of the artificial horizon by filling regions of the attitude indicator in a minimal number of graphic operations. For most graphic processors, the basic operation is to draw a vector between the coordinates (x_1, y_1) and (x_2, y_2). The graphic processor generates the line of pixels from (x_1, y_1) to (x_2, y_2) and sets the corresponding locations within the framestore to the appropriate colour. The processing task is then restricted to defining line end point coordinates in such a way as to move the horizon with a minimum number of line draw operations, but without incurring an excessive amount of geometrical computation.

Secondly, the method must be simple. The algorithm should minimise the computational overhead inherent in the geometrical computation of vector coordinates. Any reduction in graphical operations must not be at the expense of excessive time spent in computation. A satisfactory algorithm will be a careful balance of computational simplicity with a minimal number of graphical operations.

Further, the method must be robust. It must cater for worst-case situations; for a potentially safety-critical application, it must ensure that no pixels are coloured incorrectly and that the algorithm operates in all regimes without terminating in some undesirable manner. It should cope with interruption of data from the attitude sensors and restore to the correct attitude indication within one frame period.

The efficiency of the method therefore focuses on the strategy used to fill various regions or polygons within a square bounded by the motion of the horizon line. There are well known methods for filling polygons, but the real-time constraints restrict the applicability of specific methods.

Graphical Fill Methods

The boundary of a polygon can be represented by straight lines joining its vertices. Lines that are vertical or horizontal with reference to the display screen are represented as an actual line of thickness one pixel. Diagonal lines are approximated by a sequence of adjacent pixels. Clearly, pixels on an integer grid cannot exactly match the 'fractional pixels' that are needed to replicate an arbitrary line. These limitations are well understood; line drawing algorithms based on Bresenham's algorithm are widely adopted, including incorporation in graphics processors.[2] The polygon is filled by setting pixels, within the boundary defined by the edge lines, to the appropriate colour. Therefore edge coordinates must be computed.

Various methods of filling a polygon can be considered.[3,4] One method commonly adopted for non real-time applications is 'flood filling'. An internal pixel is selected from which to start the fill operation. Its neighbouring pixels are filled unless they are already set to the fill colour. Variations on this technique employ different methods for selection and ordering of neighbouring pixels, some of which may be applied recursively. There are two common problems with this method. Firstly, it is necessary to select an (albeit arbitrary) internal pixel; this can lead to considerable computations for complex shapes. More importantly, the filling proceeds one pixel at a time and each pixel written has resulted from several framestore accesses.

An alternative method commonly adopted for real-time applications is the scan-line method in which the polygon is traversed vertically, one pixel line at a time. For each horizontal line, a left-edge coordinate and a right-edge coordinate is computed, and the intermediate points written to the framestore. If the framestore is organised for 'horizontal' addressing, multiple pixels can be written as bytes, words or multiple words, with a significant gain in speed. The primary disadvantage with this method is that the derivation of the left and right edge coordinates can require considerable computation. In the general case, a horizontal line may intersect a polygon boundary at more than two points. In addition, the logical computation that is required to format multiple words from pixel coordinates can impose a severe penalty in terms of hardware cost or time. For horizontal scan-line in-fill, the number of lines written to the framestore varies with the orientation of the polygon. For example a rectangle 100 pixels by 2 pixels can be filled by 2 lines of 100 pixels, but the same rectangle rotated through 90 degrees, requires 100 lines of 2 pixels. The overhead of setting up the graphics processor for each line may dramatically reduce the performance in the latter case.

Simpler methods have been adopted for more regular shapes. For example, hardware 'blitters' are used in 'windows' applications where the high-speed drawing, erasing and moving of rectangular windows necessitates very fast fill methods. This is accomplished with the provision of hardware to copy large blocks of memory at maximum memory access rates, in a manner than is independent of processor operation and caters for windows that are not aligned on byte or word boundaries. However, this method of filling is not appropriate where the boundary between the sky and ground regions represents a range of roll angles from -180° to +180°.

Objects such as a parallelograms, rhombuses or right-angled triangles can be filled efficiently using methods which exploit a knowledge of their geometric form. For example, a right-angled triangle, orientated along the x-y axes, can be filled by drawing internal lines parallel to the longest side. The simplicity of this method derives from the fact that the computation of the coordinates of the other two sides is trivial; one end of the fill line is

incremented in y for a constant x value, the other for x along a constant y value.

Further discussion and selection of an appropriate filling techniques will be deferred pending a detailed analysis of the format of an attitude indicator in the following section.

Graphical Properties of an Attitude Indicator

While the shape of the attitude indicator is most likely to be a circle, square or rounded box, and, as suggested above, there may be advantages in making it square, the boundary of the attitude indicator can be arbitrary. The only restriction is that it must be a convex hull to ensure that a horizon line will intersect the boundary at a maximum of two points. Fig. 2 illustrates the position of the horizon line on an EFIS attitude indicator where Θ is the pitch angle and Φ is the roll angle. The aircraft is shown pitched up and rolled to the right. The line OA, representing the pitch angle, is perpendicular to the horizon line and of length $l = a\Theta$ where a is the scale factor relating pitch angle to displacement.

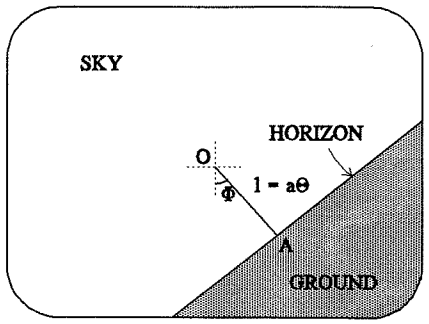

Fig. 2. Position of horizon on EFIS attitude indicator.

To adjust the horizon as a result of say, an increase in pitch angle, a new line will have to be drawn below the existing line and the trapezoid bounded by the old and new horizon lines filled with the colour representing sky.

Assuming that the old horizon line intersects the boundary at two points (1,2) and that the new horizon intersects the boundary at a further two points (3,4), the number of cases to be considered is given by the number of distinct orderings of the points (1,2,3,4) around the boundary. The six such combinations (a)-(f) are illustrated in Fig. 3 where point 1 (■) represents the right-hand end of the old horizon line, point 2 (●) represents the left-hand end of the old horizon line, point 3 (○) represents the left-hand end of the new horizon line, and point 4 (□) represents the right-hand end of the new horizon line.

It can be seen that the diagrams contain various regions, some of which must be refilled with either sky or

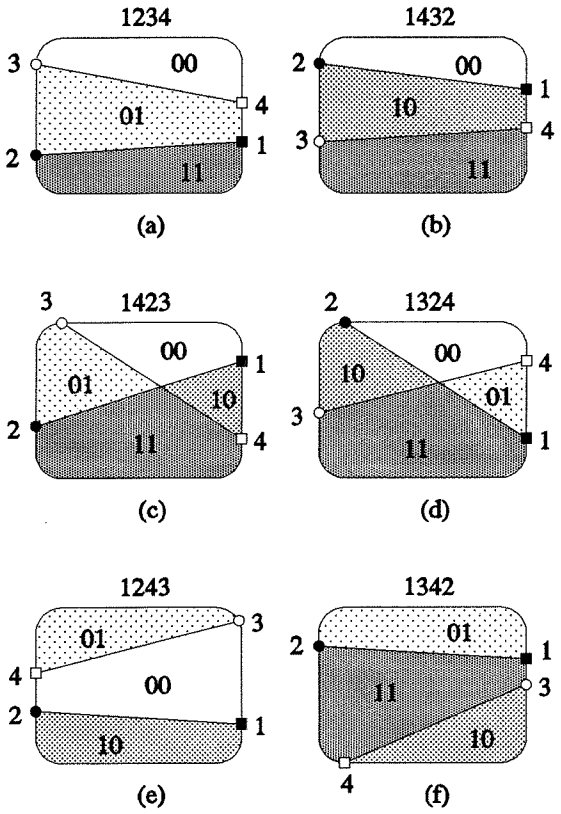

Fig. 3. Primary cases for the attitude indicator.

ground to effect the movement of the horizon. If 0 is used to indicate sky and 1 to indicate ground then

00 represents a region that remains as sky,
01 represents a change from sky to ground,
10 represents a change from ground to sky, and
11 represents a region that remains as ground;

these primary cases are summarised in the following table.

Case	Code	Zones	Zone codes	Description
(a)	1234	3	00 01 11	Increasing ground
(b)	1432	3	00 10 11	Increasing sky
(c)	1423	4	00 01 10 11	CW rotation
(d)	1324	4	00 10 01 11	CCW rotation
(e)	1243	3	01 00 10	Roll pitch up
(f)	1342	3	01 11 10	Roll pitch down

Only regions labelled 01 or 10 need be considered to adjust the position of the horizon line.

For large pitch angles, either the old horizon line or the new horizon line (or both) may not intersect the boundary at all, in which case either the pair of points 1-

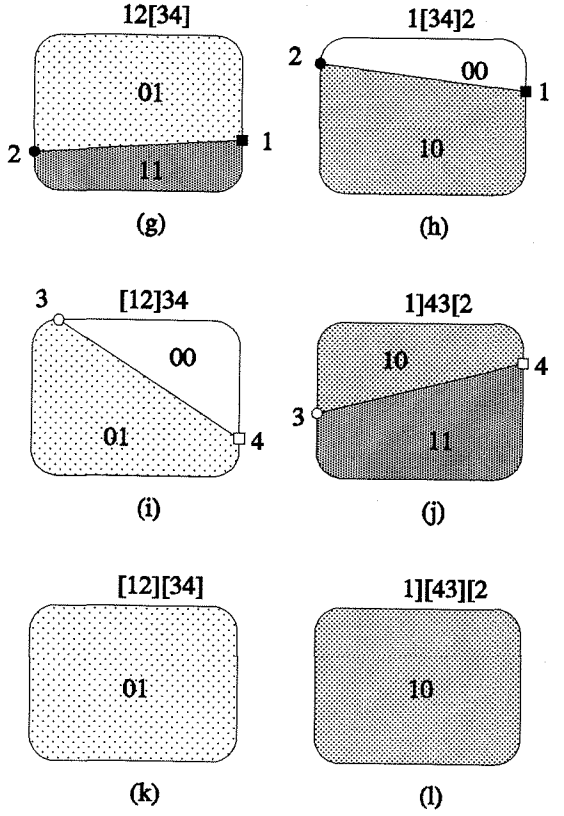

Fig. 4. Additional cases for the attitude indicator

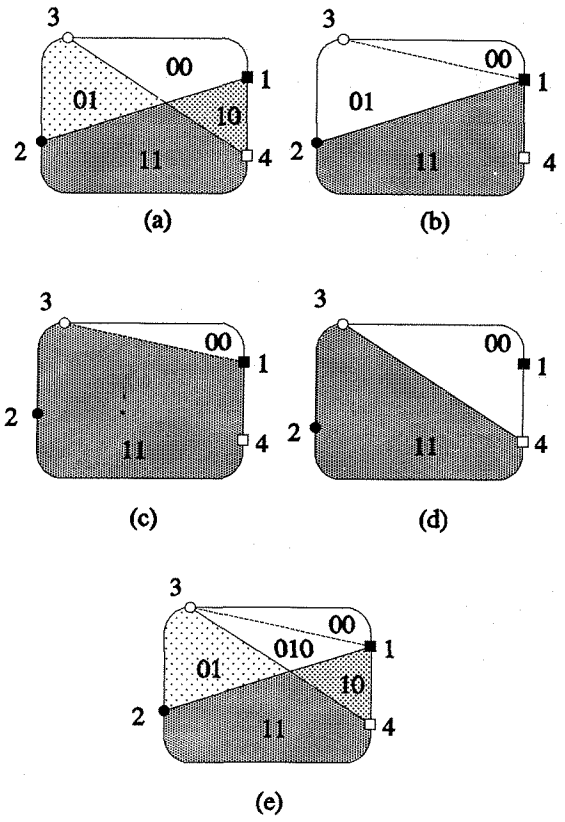

Fig. 5. Alternative strategies for cases 1423 and 1324.

2 or 3-4 will not exist. These cases (g)-(l) are illustrated in Fig. 4 and summarised below where brackets are used to indicate the non-intersecting points.

Case	Code	Zones	Codes	Description
(g)	12[34]	2	01 11	Leaving to pitch down
(h)	1[34]2	2	00 10	Leaving to pitch up
(i)	[12]34	2	00 01	Entering from pitch up
(j)	1]43[2	2	10 11	Entering from pitch down
(k)	[12][34]	1	01	Off screen pitch up to down
(l)	1][43][2	1	10	Off screen pitch down to up

From Figs. 3 and 4 it can be seen that the regions that require to be filled fall into a small number of distinct cases. For (a) and (b) the area is bounded by the old and new horizon lines and two sections of the instrument boundary. Cases (e) and (f) require two regions to be filled bounded by a horizon line and the instrument boundary. Cases (g), (h), (i) and (j) have only one region of this type. (k) and (l) require the whole area within the boundary to be filled in. The most complex case is for (c) and (d) where two wedges must be filled that are bounded by parts of both horizon lines and one section of the instrument boundary.

Cases (e) and (f) are deserving of further attention. For all cases other than these, the region or regions to be redrawn correspond to the area apparently swept by the horizon in going from (1,2) to (3,4). For (e) and (f) the motion of the horizon line would appear to have been a rotation with point 1 moving to 4, while point 2 moves in the same sense to 3, whereas, the areas to be filled correspond to a change in pitch to remove the line from the instrument and then restoring it with the aircraft inverted. Clearly the latter strategy is easier and probably quicker to implement than the rotation of the horizon on the instrument.

This observation suggests a method of simplifying cases (c) and (d) where change in roll dominates over change in pitch. This is illustrated in Fig. 5(a) for the case 1423. Starting with the horizon line at (1,2) (Fig.5(b)), the strategy would be to sweep the horizon in a fan centred on point 1 from 2 to 3 filling with ground (Fig.5(c)) and then sweep it in a fan centred on 3 from 1 to 4 filling with sky (Fig.5(d)). Although this has the penalty of having to fill the region 010 (Fig.5(e)) twice, the simplification of the computation of the in-fill regions may outweigh the additional time spent filling. The merit of this approach will become clear when discussion of fill

techniques is resumed.

Notice that the colour of any one of these regions is determined uniquely by the rotational sense of the intersection points. For example, point 1 moving counter-clockwise towards 4, or 2 moving clockwise towards 3 will be a boundary of ground, and conversely 1 moving clockwise towards 4, or 2 moving counter-clockwise towards 3, will be a boundary of sky.

At this point it is important to establish the appropriate direction for a point to rotate when defining a region boundary. There are clearly two paths round the circumference between any two points: clockwise and counter-clockwise; the shortest path may not be the appropriate route. For cases (a), (b), (c) and (d) the path of point 1 towards point 4 must not include 2, and the path of 2 to 3 must not include 4. In the case of (e) and (f) the path 1 to 2 should not include 3 or 4, and the path of 3 to 4 must not include 1 or 2.

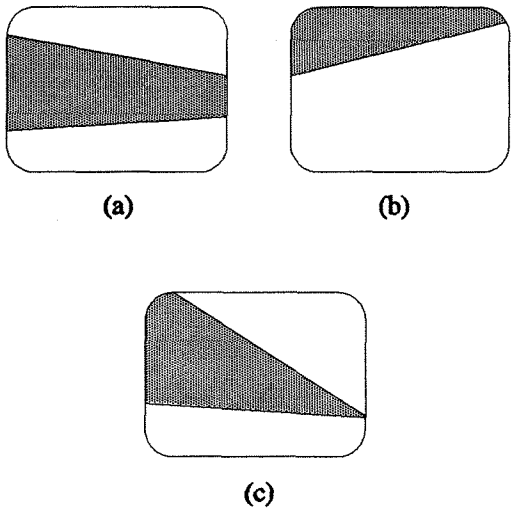

Fig. 6. Graphical objects: (a) a block, (b) a segment, (c) a wedge.

Thus it can be seen that for all cases except (c) and (d), the position of the horizon can be adjusted by filling just two objects: a 'block' - bounded by the old and new horizon line and two sections of the instrument boundary; and a 'segment' - bounded by a horizon line and one section of the instrument boundary (Fig. 6). The fill colour is either sky or ground dependent on the rotational sense. The approach can be extended to cases (c) and (d) if the strategy illustrated in Fig. 5 is employed. This requires the introduction of a third object, a 'wedge', bounded by the old and new horizon and just one section of the boundary (Fig. 6(c)). Once the particular case has been identified the appropriate fill procedures can be called to achieve the correct adjustment.

A possible approach to identifying the particular case is to determine the ordering of the intersecting points around the instrument boundary and hence recognize which of the strategies so far discussed is appropriate. The exceptional cases (g) to (l) will be excluded for the moment because, as will be shown later, they are relatively simple to deal with. The ordering may be achieved as follows. Assume that each of the four intersection points is distinct and that each is allocated an index s_1 to s_4 which specifies its position on the circumference. The index could be the distance around the circumference from some arbitrary origin, or an integer representing one of N discrete points defining the boundary. Having adjusted the indices modulo-N, so that s_1 is the smallest value, the indices are compared. A truth table illustrates the six cases (a) to (f) showing the outcomes of the comparisons of the three indices s_i. Notice that there are two combinations of outcomes that cannot occur. (These are shown on the truth table for completeness)

	$s_2<s_3$	$s_3<s_4$	$s_2<s_4$	
1234	1	1	1	(a)
1432	0	0	0	(b)
1423	1	0	0	(c)
1324	0	1	1	(d)
1243	1	0	1	(e)
1342	0	1	0	(f)
x	1	1	0	
x	0	0	1	

Thus the appropriate fill strategy may be determined as a result of at most six comparisons and three additions.

Cases (g) to (l) can be identified as follows. 'On' is a Boolean variable that indicates whether the new horizon intersects the instrument boundary. This would most likely be a result returned by the routine used to clip the horizon to the boundary.[5] 'old_On' indicates whether the previous horizon intersected the boundary. 'Pitch' and 'old_Pitch' indicate whether the new and old pitch angles are positive. The additional cases can be recognised with simple tests on the values of 'On', 'old_On', 'Pitch' and 'old_Pitch' as indicated in the following truth table, where 'x' represents a redundant condition.

On	old_On	Pitch	old_Pitch	
0	1	0	x	(g)
0	1	1	x	(h)
1	0	x	1	(i)
1	0	x	0	(j)
0	0	0	1	(k)
0	0	1	0	(l)
1	1	x	x	(a to f)

This understanding of the possible cases that can occur within an attitude indicator is now applied to selecting a suitable technique for polygon filling.

Object Filling

The analysis of the previous section indicates that, for any combination of changes in pitch and roll angles, the attitude indicator can be adjusted by filling simple, well-defined shapes within the boundary. Cases (a) and (b) require a four-sided object (a 'block') to be filled, bounded by the old and new horizon lines and two sections of the boundary. For cases (e), (f), (g), (h), (i) and (j) the object (a 'segment') is two-sided, bounded by a horizon line and one section of the boundary. (k) and (l) require the whole instrument to be refilled and is a special case of the 'block'. Cases (c) and (d) require two 'wedges' to be filled bounded by both horizons and a section of the boundary.

To ensure minimal graphic access time the objects should be filled using horizontal lines. The implication of adhering to this approach would be to severely complicate the computation of the end points of the fill lines. Some will be on the instrument boundary, others will be on one of the horizon lines; the computation will be highly orientation dependent. Whereas the horizon line is dynamic, the boundary is static and there is negligible computation in retrieving the coordinates of a point moving along the boundary. Clearly an advantageous simplification of the algorithm results from filling the objects with vectors drawn between boundary points. If the boundary is defined by an array of points x_s,y_s the index of the array can also be used as an index to all points defining the boundary. The filling process is now independent of orientation of the sky and ground regions relative to the horizontal raster lines of the display CRT.

As described in the previous section, the correct strategy for adjusting the horizon can be determined using the indices of the points where the horizon intersects the boundary. Thus the indexing used to determine the correct strategy can be also be used to fill the polygons efficiently with the aid of a look-up table of the boundary in screen coordinates.

The simplification in the computation associated with filling when using boundary points can be seen with reference to Fig. 7(a), where the 'segment' has to be filled between the line AB and the boundary. Filling the segment with horizontal or vertical lines requires the computation of the intermediate points along the diagonal line. Some lines terminate on the diagonal while others terminate on boundary points, adding to the complexity of the filling algorithm. Alternatively, the segment can be filled with lines drawn parallel to AB, incrementing the end points from A and B until they coincide at C. With the boundary point coordinates in the form of an array it is a simple matter to increment the end point indices. There is no calculation involved in this inner loop.

Fig. 7(b) illustrates the case of a 'block', which can be filled in a similar manner using only lines between boundary points. Lines are drawn, initially parallel to

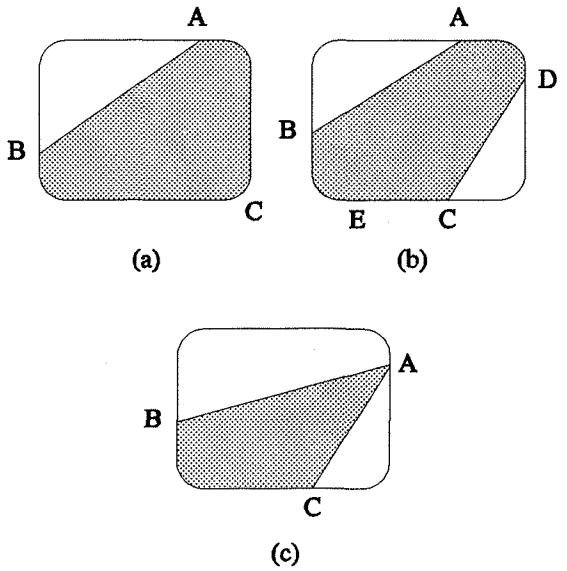

Fig. 7. Object filling: (a) a segment, (b) a block, (c) a wedge.

AB, incrementing the end points round the boundary until A reaches D and B arrives at C. Once one of the termination points has been reached, in this case D, a fan of lines is drawn up to the other point, from E to C centred on D. Fig. 7(c) illustrates the filling of a wedge with lines drawn from B to C centred on A. Thus the position of the artificial horizon can be adjusted efficiently, filling just three objects, using lines drawn between boundary points.

Up to this point the analysis has not restricted the boundary to any particular shape, save the requirement that the horizon intersects the boundary at a maximum of two points. Although EFIS attitude indicators are implemented with a range of shapes, the basic shape is often square and curved boundaries can be overlaid to obscure a square implementation. This observation simplifies both the clipping algorithm and the coordinate indexing scheme.

Evaluation of the Algorithm

The algorithm that resulted from the analysis of the graphical properties of the attitude indicator was programmed in the Modula-2 language for a 100 pixel square, where the edge of the square is equivalent to a pitch angle of 50 degrees, as illustrated in Fig. 8. The 400 boundary points were indexed from the top left-hand corner where the coordinates were (-50, 50) relative to the centre of the instrument. Separate procedures were written to fill a block, a segment, and a wedge. The

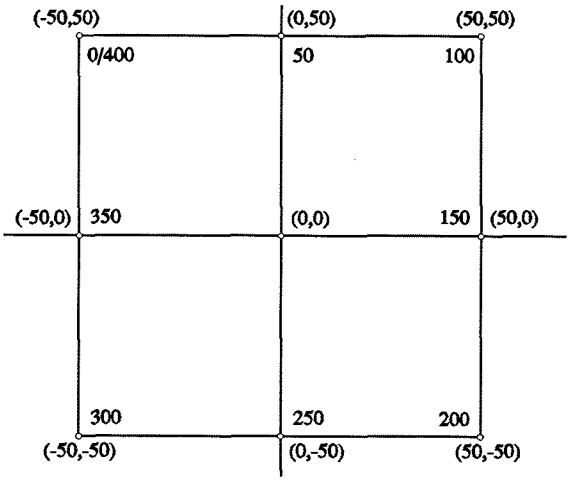

Fig. 8. Coordinate indexing for a square instrument.

algorithm determined which combination of these needed to be called for each adjustment of the horizon. The program was compiled using the Stony Brook QuickMod compiler version 2.2 and run on a CompuAdd 425 PC with Intel i486 25 MHz CPU, bus speed 12 MHz and a CompuAdd VGA graphics card based on the Tseng Laboratories 4000 graphics processor. All filling was achieved using the procedure DrawLine available in the Stony Brook library module Graphics. Fig. 9 indicates the speed of the graphics processor when using the DrawLine procedure for lines drawn at increasing angles to the horizontal. Two cases are shown: Firstly where the moving end point of the line follows a circle of radius 100 pixels horizontally, and the second case where the end point moves around a square of side 100 pixels. It can be seen that for the horizontal line the access time is approximately 1.25 μs per pixel. These results are provide as a simple bench mark to compare speeds of this or other algorithms on different machines.

The algorithm was exercised using first a test harness into which independent values for pitch and roll angles could be inserted. All modes were checked exhaustively and no spurious behaviour was observed. A second test harness simulated the motion of an aircraft which could be flown using the cursor keys. Again no spurious behaviour was observed.

The speed of the algorithm was measured by timing 5000 repetitions of particular manoeuvres. The Modula-2 run-time library contained a procedure to return elapsed time to the nearest 0.05 seconds. Fig. 10 illustrates the update time for various changes in pitch angle as a function of constant roll angle. The maxima in the region of 45 degrees result from the instrument boundary being square. In practice, the dynamics of the aircraft will limit the pitch changes to the order of 1 degree per iteration at 50 Hz, so that a typical update time when using this algorithm would be approximately 1.5 ms. Fig. 11 illustrates update times for changes in roll as a function of initial roll angles. The times are approximately twice that for pitch updates which is to be expected from the nature of the algorithm.

Discontinuities are observed within the results for a given pitch or roll increment. These occur when the number of fill lines for that increment changes. For example, for roll angles up to 29 degrees a change in pitch of 1 degree requires just 2 lines to be drawn. Whereas when the roll angle is 30 degrees, 3 draw operations become necessary. Pitch changes of 2 degrees require 4 lines at 41 degrees of roll angle but only 3 between 42 and 48 degrees. This occurs as a result of the rounding to the nearest boundary pixel during the clipping procedure. The points for a given pitch or roll adjustment are moving between adjacent curves which correspond to different numbers of line draw operations.

To confirm this explanation the number of lines drawn in adjusting the horizon was measured for the same changes in pitch and in roll that had been timed. The results for changes in pitch are illustrates in Fig. 12. It can be seen that the changes in the numbers of lines necessary to make an adjustment correlate exactly with the discontinuities in the timing measurements.

For a 1-degree change in pitch, 2 fill lines are drawn. If the horizon has moved at all, within the resolution of the boundary points, at least 2 lines need to be drawn. This is because the actual horizon line is superimposed in a contrasting colour. A minimal adjustment of the attitude indicator is therefore to delete the old horizon, by drawing one fill line in either sky or ground, and then draw the new horizon line, a total of 2 draw-line operations.

It will be noticed that the measured times are higher than would be predicted for the square boundary from Fig. 9. This is to be expected in that the time taken to adjust the position of the horizon includes the time taken to execute the algorithm, and so decide the appropriate fill strategy, in addition to the time taken to fill the relevant polygons via graphical operations. The times illustrated in Fig. 9 are purely for graphical access and the processing associated with Bresenham's routine within the Modula-2 draw line procedure. The algorithm computation time was separated from the time associated with graphical operations by replacing the draw line procedure with one that immediately returned to the calling procedure. For a 1-degree change in pitch at a roll angle of 0 degrees the computation time was 0.120 ms out of a total of 0.374 ms or 32% of the total, and decreased with increasing pitch change. For a change in roll of 1 degree, the computation occupied 0.132 ms out

of 0.516 ms, or 26% of the total, again decreasing with increasing roll increments. This confirms the efficiency of the algorithm.

Conclusions

A novel algorithm has been presented for filling the sky and ground regions associated with updating the position of the artificial horizon on an EFIS attitude indicator, based on a detailed analysis of the various combinations of pitch and roll changes that can occur. Just three types of graphical object are necessary to update the display. The algorithm is relatively simple because the process of identifying the particular case and the strategy for filling the appropriate objects are both independent of the orientation of the objects on the CRT. It has been demonstrated that the algorithm is robust in that, following data loss, for example, it is capable of adjusting the horizon to its correct position within one iteration no matter how large the change in pitch or roll. The algorithm is both simple to implement and efficient to compute. Its simplicity enables an attitude indicator to be drawn on a raster-driven video display unit without special purpose graphical in-fill processor cards. The algorithm has been implemented in a real-time flight simulator and evaluated with satisfactory results. The measured performance suggests that the algorithm is applicable both to electronic flight instrumentation systems and real-time flight simulation.

References

[1] Konicke, M. L., 747-400 Flight Displays Development. *AIAA Report 88-4439*, (1988).

[2] Bresenham, J. E., Algorithms for the Computer Control of a Digital Plotter. *IBM System Journal*, **4**, 1 (1965).

[3] Pavlidis, T., Filling Algorithms for Raster Graphics. *Computer Graphics*, **12**, Aug, 161 (1978).

[4] Foley, J. D. et al., *Computer Graphics : Principles and Practice*. 2nd Edition, Addison-Wesley (1990).

[5] Newman, W. M. and Sproull, R. F., *Principles of Interactive Computer Graphics*. 2nd Edition, McGraw Hill (1981).

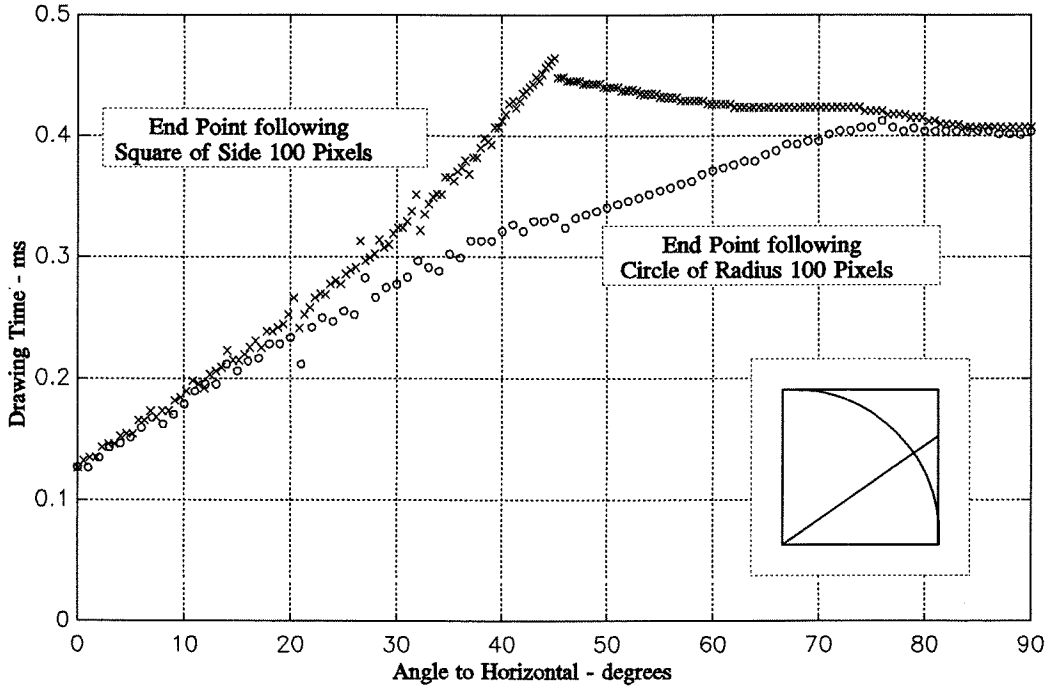

Fig. 9. Speed of VGA graphic processor with Modula-2 Drawline procedure.

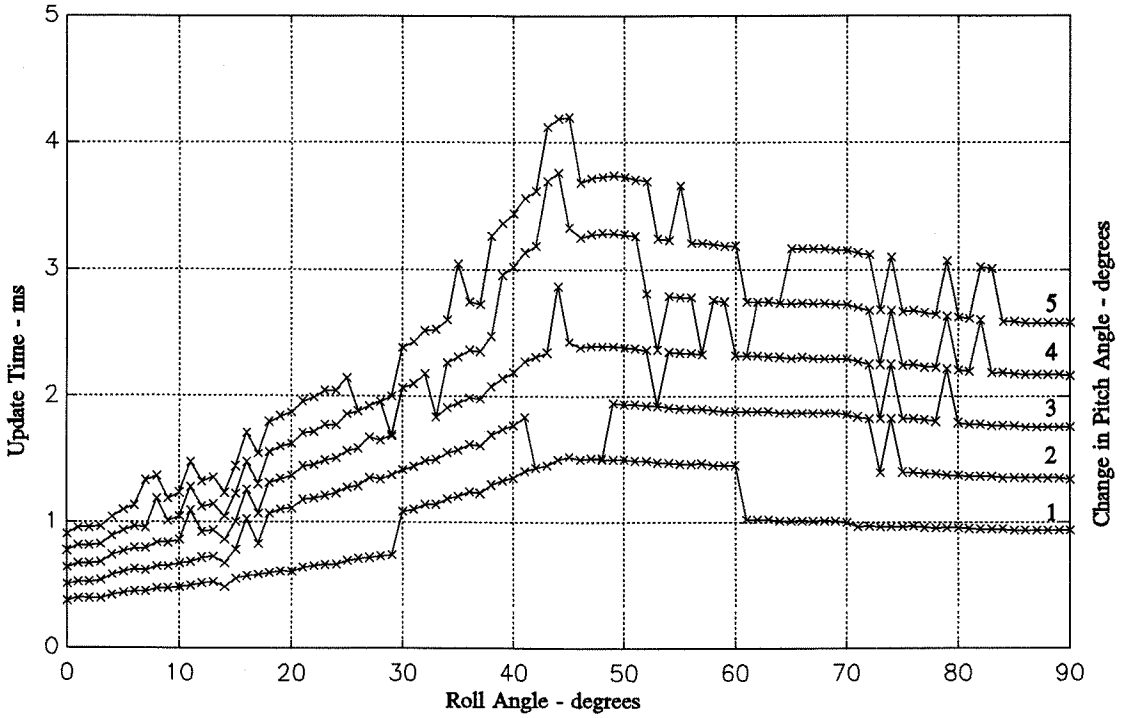

Fig. 10. Time taken to adjust the horizon for changes in pitch in the range 1 to 5 degrees as a function of static roll angle.

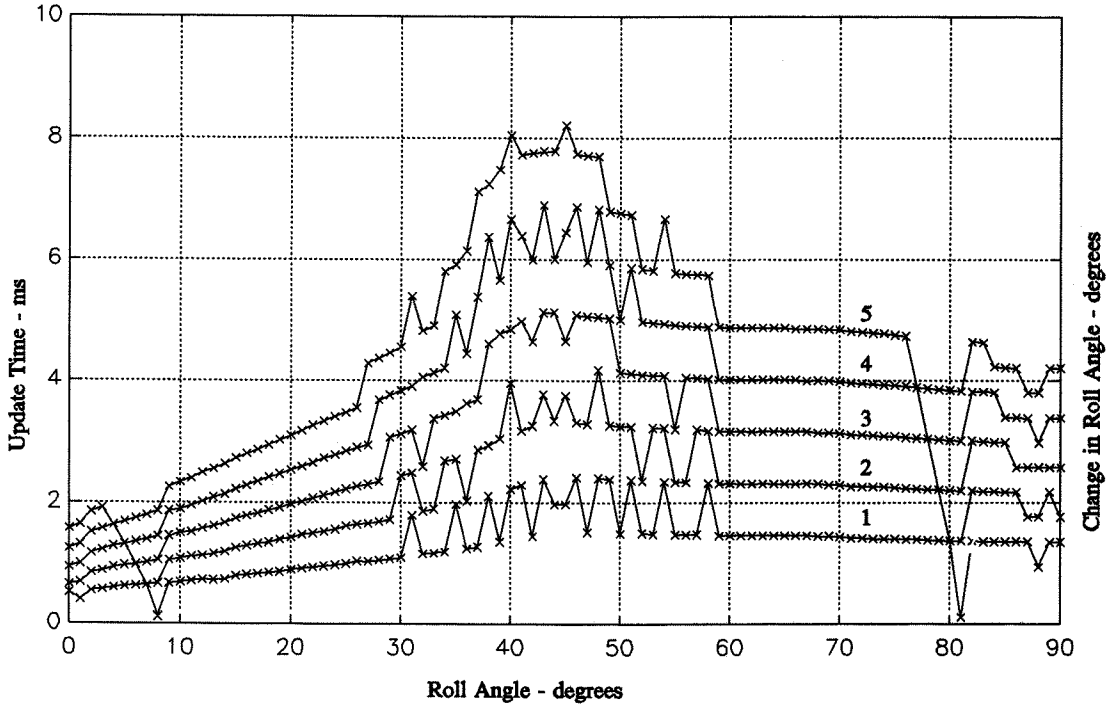

Fig. 11. Time taken to adjust the horizon for changes of roll in the range 1 to 5 degrees as a function of initial roll angle.

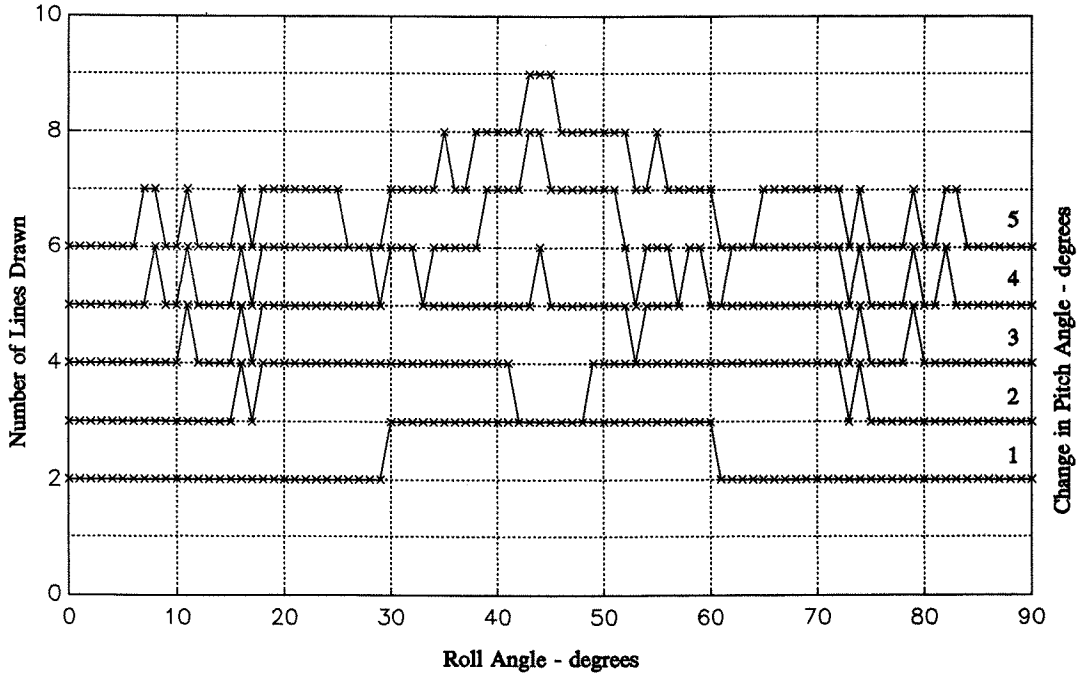

Fig. 12. The number of lines drawn in adjusting the horizon for changes in pitch as a function of static roll angle.