Drawing lines using the Amiga blitter

The Amiga has a blitter coprocessor that can perform a (small) set of tasks, one of which is to draw lines. Drawing lines using the blitter is usually quite a bit faster than drawing lines using the Amiga CPU (the Motorola 68000).

The CPU programs the blitter by writing control information to custom chipset registers (registers in the $DFF000 range). When the BLTSIZE register have been written to, the blitter starts execution of the line drawing, and runs independently until the drawing is finished. In the meantime, the CPU can perform other calculations (subject to contention over chip memory cycles).

A line is drawn into a framebuffer. A framebuffer is a piece of chip memory that corresponds to a grid of pixels (a bitplane) with width W and height H. Each bit in the framebuffer corresponds to a pixel being set or clear. The blitter operates on words (a word is two bytes, or 16 bits), and each row of the framebuffer must therefore have a width that is a multiple of 16 pixels.

As a running example, let's consider a framebuffer with W=320 and H=256. Each row is 320/8 = 40 bytes, or 20 words. The first pixel of the framebuffer corresponds to bit 15 (the most significant bit) of the first word. The bit that corresponds to the pixel at x=75, y=19 is bit (15 - (75 mod 16)) = 4 in the word at address y*40 + 2*floor(x/16) = 768 relative to the address of the framebuffer.

Direction

Lines can be drawn in any direction. Let (x1, y1) be the start point of the line and (x2, y2) be the end point. Consider a vector from (x1, y1) to (x2, y2). The vector has components dx=x2-x1 and dy=y2-y1 along the x and y axes, respectively. Take the line from (20, 15) to (120, 50) as an example, for which dx=120-20=100 and dy=50-15=35. As both dx and dy are positive, we see that the direction of the line is to the right and down. As the absolute value of dx is greater than the absolute value of dy, the line's major direction is to the right, and its minor direction is down. The possible combinations of major and minor directions can be summarized by an octant, as seen in Figure 1 and Table 1.

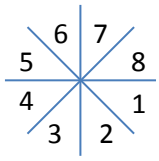

Figure 1: Octants

Octant	Major direction	Minor direction	Code
1	Right	Down	4
2	Down	Right	0
3	Down	Left	2
4	Left	Down	5
5	Left	Up	7
6	Up	Left	3
7	Up	Right	1
8	Right	Up	6

Table 1: Octants and corresponding major and minor directions

How the blitter operates

The blitter is set up with the number of iterations that should be performed of the following loop. For each execution of the loop the blitter does the following:

1. Update the bit corresponding to the current (x, y) position in the framebuffer.
2. Updated the current position with a step either in the major direction, or in the combined major and minor direction.

How the bit corresponding to the current position is updated depends on some parameters that the blitter is programmed with; we explain those later. For now, we can think update = set.

When the current position is updated, a step is always taken in the major direction. For example, if the current position is (x, y) and the major direction is right, then the new current position is (x+1, y). If the value of the *accumulator* is greater than or equal to zero, a step is also taken in the minor direction. The accumulator is a register that is written to before the drawing starts. If a step was taken in the minor direction then one value is added to the accumulator, and if a step was not taken in the minor direction then another value is added to the accumulator.

These three values (the initial accumulator, the increment of the accumulator if a step was taken in the minor direction, and the increment of the accumulator if a step was not taken in the minor direction) are all written to custom chip registers before the drawing starts. If these values are set correctly then blitter will step along the line and update the correct bits of the framebuffer.

The values that must be written before the blitter is started are specified in Table 2.

Value	Description
octant	The direction of the line
p	A pointer to the word in the framebuffer containing the first pixel of the line
shift	($8000 >> shift) masks the bit in *p (the word pointed to by p) that corresponds to the current position
acc	The initial accumulator
inc_maj	The increment of the accumulator if no step was taken in the minor direction
inc_majmin	The increment of the accumulator if a step was taken in the minor direction
ymod	The number of bytes corresponding to one row in the framebuffer
count	Number of iterations of the drawing loop; also the number of steps in the major direction
onedot	A flag that (if set) specifies that only the first pixel each row should be updated
xor	A flag that (if set) specifies that the drawing should be done in exclusive or-mode; if a bit was already set when that bit is updated then the bit is cleared

Table 2: Values that the blitter must be programmed with before the drawing starts

The following pseudo-code describes the algorithm that the blitter uses to draw a line.

```
LEFT = 1; RIGHT = 2; UP = 4; DOWN = 8

maj_step = [RIGHT, DOWN, DOWN, LEFT, LEFT, UP, UP, RIGHT]
min_step = [DOWN, RIGHT, LEFT, DOWN, UP, LEFT, RIGHT, UP]

def draw_line(octant, p, shift, ymod, acc, inc_maj, inc_majmin, count, onedot, xor):
    dot_on_row = False
    while cnt > 0:
        if (not onedot) or (onedot and not dot_on_row):
            w = *p
            b = (0x8000 >> shift)
            *p = (w ^ b) if xor else (w | b)
            dot_on_row = True

        if acc < 0:
            step = maj_step[octant-1]
            acc += inc_maj
        else:
            step = maj_step[octant-1] | min_step[octant-1]
            acc += inc_majmin

        if (step & LEFT) != 0:
            shift -= 1
            if shift == -1:
                shift = 15
                p -= 2
        if (step & RIGHT) != 0:
            shift += 1
            if shift == 16:
                shift = 0
                p += 2
        if (step & UP) != 0:
            p -= ymod
            dot_on_row = False
        if (step & DOWN) != 0:
            p += ymod
            dot_on_row = False

        cnt -= 1
```

How to set the initial accumulator and the increments

Now we describe how to derive the accumulator and increments. Consider the line in Figure 2 that starts at (x1, y1) and ends at (x2, y2). As the slope of the line is independent of the absolute positions that the line starts at, we assume that x1=0 and y1=0. When the drawing starts, the current position is then (x', y') = (0, 0). After the first pixel is set at (x', y') the current position should be updated. The question is if the step should be in the major direction only, to position (x'+1, y'), or if the step should be in the combined major and minor direction, to position (x'+1, y'+1). To decide, we should determine which of the two positions is closer to the actual line. By looking at Figure 2 we see that position (x'+1, y'+1) is closer to the line, so a step should be taken in the minor direction.

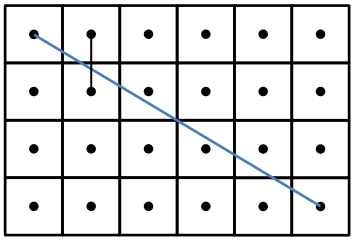

Figure 2: Line from (0, 0) to (5, 3).

From the equation for the line, we find that the y coordinate of the line at x coordinate x'+1 is:

$$y = (x' + 1)*dy/dx$$

If this y coordinate is greater than or equal to y'+0.5 then the line is closer to position (x'+1, y'+1) than to position (x'+1, y'), and the step in the minor direction should be taken. Combining these equations we see that we should take the step in the minor direction if:

$$(x' + 1)*dy/dx \geq y' + 0.5$$

Multiplying both sides with 2*dx and reordering terms we get:

$$(2*dy - dx) + 2*(x'*dy - y'*dx) \geq 0$$

Note that this is precisely the kind of expression that is used by the blitter to decide if it should take a step in the minor direction or not. When the current position is updated the first time, the current position is x'=0, y'=0, and the inequality is then:

$$2*dy - dx \geq 0$$

The initial accumulator should therefore be set to 2*dx - dy. When a step is taken in the major direction only, x' is incremented by one and the accumulator should be incremented by 2*dy, and when a step is taken in both the major and minor direction, both x' and y' are incremented, and the accumulator should therefore be incremented by 2*(dy-dx).

At this point, one small modification needs to be made. The registers that store the accumulator and increments can only hold even numbers. If an odd number is written to one of those registers, the value will be truncated to the next smaller even number, which leads to that an incorrect value is stored in the register. In order to handle this, we will multiply both sides of the inequality by two, which will not change the inequality. We then have:

$$(4*dy - 2*dx) + 4*(x'*dy - y'*dx) \geq 0$$

The correct values are:

- acc = 4*dy - 2*dx
- inc_maj = 4*dy
- inc_majmin = 4*(dy - dx)

Note that these values were calculated for the example where the line is drawn in octant 1. However, if another octant is used, the expressions look much the same, but dx should be replaced by the absolute value in the major direction and dy by the absolute value in the minor direction.

Sub-pixel correctness

It is also possible to use the blitter line drawing to draw sub-pixel correct lines, that is, lines that start and end at coordinates that may not be integers (not at pixel centers), but can be in between pixels, and the drawing of between pixels takes this into account. The required setup and calculation of the initial accumulator and increments are different, but after that there is no difference to the blitter.

Registers

The registers that the values should be written into are shown in Table 3.

Value	Register	Comment
$8000	BLTADAT	
$FFFF	BLTBDAT	This is a texture which can be changed but we assume we want a solid line
$FFFF	BLTAFWM	
$FFFF	BLTALWM	
inc_majmin	BLTAMOD	
inc_maj	BLTBMOD	
ymod	BLTCMOD	
ymod	BLTDMOD	
acc	BLTAPTR	Initial accumulator. This is a signed long word (4 bytes)
p	BLTCPTR	Pointer to word containing first pixel
p	BLTDPTR	
shift	BLTCON0 bits 15-12	x1 modulo 16. This is the number of times $8000 should be shifted to reach the first pixel in the word pointed to by p
11	BLTCON0 bits 11-8	
xor	BLTCON0 LF bit 7-0	0x42 if exclusive or mode, or 0xf2 if inclusive or
octant	BLTCON1 bits 4-2	The value written is the code from the right-most column in Figure 1, and not the octant number itself
acc < 0	BLTCON1 bit 6	The SIGNFLAG bit should be set if the initial accumulator is negative
onedot	BLTCON1 bit 1	
1	BLTCON1 bit 0	
0	BLTCON1 other bits	All bits in BLTCON1 that wasn't mentioned above should be zero
count	BLTSIZE bits 15-6	The number of iterations of the loop that the blitter shall perform
2	BLTSIZE bits 5-0	BLTSIZE should be written last as it starts the line drawing operation