Bresenham's line algorithm

Here is a C program using the allegro library that I pout up and ported to 6502 assembler. This implementation was designed by Janusz Ganczarski.

```
#include "allegro.h"

void init();
void deinit();
void B_Line (int x1, int y1, int x2, int y2);
void PutPixel (int x,int y);

int main() {

    init();

    B_Line (1, 1, 50, 10);

    while (!key[KEY_ESC]) {
        /* put your code here */
    }

    deinit();
    return 0;
}
END_OF_MAIN()

void init() {
    int depth, res;
    allegro_init();
    depth = desktop_color_depth();
    if (depth == 0) depth = 32;
    set_color_depth(depth);
    res = set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0);
    if (res != 0) {
        allegro_message(allegro_error);
        exit(-1);
    }

    install_timer();
    install_keyboard();
    install_mouse();
    /* add other initializations here */
}

void deinit() {
    clear_keybuf();
    /* add other deinitializations here */
}
```

```
void PutPixel (int x,int y)
{
    putpixel(screen,x,y,makecol(255,255,255));
}

//--------------------------------------------------------------------------
-
// rysowanie linii algorytmem Bresenhama
//--------------------------------------------------------------------------
-
void B_Line (int x1, int y1, int x2, int y2)
{
  // zmienne pomocnicze
  int d,dx,dy,ai,bi,xi,yi;

// dy - "dlugosc" y
// dx - "dlugosc" X
// xi , yi - kierunek rysowania w osi x , y
//

  // ustalenie kierunku rysowania oraz kroków zmiennych
  if (x1 < x2)
    {
      xi = 1;
      dx = x2 - x1;
    }
  else
    {
      xi = -1;
      dx = x1 - x2;
    }

  // ustalenie kierunku rysowania oraz kroków zmiennych
 if (y1 < y2)
   {
     yi = 1;
     dy = y2 - y1;
   }
 else
  {
     yi = -1;
     dy = y1 - y2;
  }

 // pierwszy piksel
 PutPixel (x1,y1);

 // oś wiodąca OX
 if (dx > dy)
   {
```

```
      ai = (dy - dx) * 2;
      bi = dy * 2;
      d = bi - dx;

      // warunek != zamiast < umożliwia rysowanie "pod górkę"
      while (x1 != x2)
        {
          // przejście NE
          if (d > 0)
            {
              y1 += yi;
              d += ai;
            }
          // przejście E
          else
            d += bi;
          x1 += xi;
          PutPixel (x1,y1);
        }
    }
  // oś wiodąca OY
  else
    {
      ai = (dx - dy) * 2;
      bi = dx * 2;
      d = bi - dy;

      // warunek != zamiast < umożliwia rysowanie "pod górkę"
      while (y1 != y2)
        {
          // przejście NE
          if (d > 0)
            {
              x1 += xi;
              d += ai;
            }
          // przejście E
          else
            d += bi;
          y1 += yi;
          PutPixel (x1,y1);
        }
    }
}
```

6502 code was written using 6502sim and debugged here too , it's not optimized in any way and entire code uses only accumulator - it was written to represent that C program's B_Line function completely and needs further optimization to be really useful. it also uses 8bit variables only so lenght of line is limited too , and so on. 'debug' call should be exchanged with proper putpixel routine, in simulator 'debug' routine is used to watch how coords of pixels are chagning with line being drawn. after all this code is good for beginners I hope :)

```
; defines for 6502 simulator debugging
io_area = $e000
io_cls  = io_area + 0    ; clear terminal window
io_putc = io_area + 1    ; put char
io_putr = io_area + 2    ; put raw char (doesn't interpret CR/LF)
io_puth = io_area + 3    ; put as hex number
io_getc = io_area + 4    ; get char


    *= $c000
    LDA #10
    STA l_x1
    lda #4
    STA l_y1
    LDA #0
    STA l_x2
    lda #0
    STA l_y2
    jmp do_line
    *= $c100
do_line
; // zmienne pomocnicze
;//  int d,dx,dy,ai,bi,xi,yi;

;U2 = two's compliment
;ZU = normal binary

;var name       type  desc
l_dx = $10      ; ZU - "dlugosc" x (rozpietosc na osi)
l_dy = l_dx + 1 ; ZU - "dlugosc" y
l_xi = l_dx + 2 ; U2 xi,yi - kierunek rysowania w osi x , y
l_yi = l_dx + 3 ; U2
l_ai = l_dx + 4 ; U2 step
l_bi = l_dx + 5 ; U2 step
l_d  = l_dx + 6 ; U2 'error'

;points
l_x1 = l_dx + 7 ; ZU poczatek linii
l_y1 = l_dx + 8 ; ZU
l_x2 = l_dx + 9 ; ZU koniec linii
l_y2 = l_dx +10 ; ZU


; // ustalenie kierunku rysowania oraz kroków zmiennych
;// deciding what direction to draw + steps
;  if (x1 < x2)

    ;if
x1mx2   LDA l_x1
    CMP l_x2
    BCC x1mx2_
```

```
    jmp x1wx2
x1mx2_
    ;then [x1<x2]
    LDA #1
    STA l_xi ; xi = 1;
    LDA l_x2
    SEC
    SBC l_x1
    STA l_dx ; dx = x2 - x1
    jmp y1my2
    ;else [x1=>x2]
x1wx2   lda #255 ; -1 U2
    STA l_xi ; xi = -1;

    LDA l_x1
    SEC
    SBC l_x2
    STA l_dx ; dx = x1 - x2
;// ustalenie kierunku rysowania oraz kroków zmiennych
;// deciding what direction to draw + steps
;if (y1 < y2)

y1my2 ; [y1<y2]

    ;if
    LDA l_y1
    CMP l_y2
    Bcc y1my2_
    jmp y1wy2
y1my2_
    ;then
    LDA #1
    STA l_yi ; yi = 1;
    LDA l_y2
    SEC
    SBC l_y1
    sta l_dy ; dy = y2 - y1
    jmp skok1
    ;else [y1>y2]
y1wy2
    lda #255 ; -1 U2
    STA l_yi ; yi = -1;

    LDA l_y1
    SEC
    SBC l_y2
    STA l_dy ; dy = y1 - x2
skok1

;;;;;
; postaw pierwszy pixel
```

```
; put first pixel

; putpix(x1,y1);

    jsr debug


; spr ktora os jest wiodaca
; check 'primary axis' (longer distance)
    ; if (dx>dy)  // (dy<dx)
    LDA l_dx
    CMP l_dy
    BCC wiod_oy ; wiodaca jest OY (dy>dx)

;// oś wiodąca OX
wiod_ox
    ;ai
    LDA l_dy
    SEC
    SBC l_dx
    CLC
    ROL ;a
    STA l_ai ; ai = (dy-dx)*2
    ;bi
    LDA l_dy
    CLC
    ROL ;a
    STA l_bi ; bi = dy * 2
    ;d
    LDA l_bi
    SEC
    SBC l_dx
    STA l_d ; d = bi - dx
whx1rx2 ;// while (x1 != x2)
    LDA l_x1
    CMP l_x2
    BEQ l_end

;        if (d > 0)
    LDA l_d
    CMP #0
    beq _a
    BPL _else
_a  ; <0
    ; przejscie E

    LDA l_d
    CLC
    ADC l_bi
    sta l_d ; d += bi;
```

```
    jmp _reszta

    ;else [d>0]
_else
;     // przejście NE

    LDA l_y1
    CLC
    ADC l_yi
    sta l_y1 ;y1 += yi;

    LDA l_d
    CLC
    ADC l_ai
    sta l_d ; d += ai;

;end else

_reszta

    LDA l_x1
    CLC
    ADC l_xi
    sta l_x1 ; x1 += xi;

;;// putpix(x1,y1);
    jsr debug
    jmp whx1rx2 ; (petla while)
    jmp l_end

;// os wiodaca OY (dy > dx)
wiod_oy

    ;ai
    LDA l_dx
    SEC
    SBC l_dy
    CLC
    ROL ;a
    STA l_ai ; ai = (dx-dy)*2
    ;bi
    LDA l_dx
    CLC
    ROL ;a
    STA l_bi ; bi = dx * 2
    ;d
    LDA l_bi
    SEC
    SBC l_dy
    STA l_d ; d = bi - dy
why1ry2     ;// while (y1 != y2)
```

```
    LDA l_y1
    CMP l_y2
    BEQ l_end


 ;        if (d > 0)
    ; then
    LDA l_d
    CMP #0
    BEQ _b
    BPL _elsey
_b  ; <0

    LDA l_d
    CLC
    ADC l_bi
    sta l_d ; d += bi;


    jmp _resztay

    ;else [d>0]
;     // przejście NE
_elsey ; >0


    LDA l_x1
    CLC
    ADC l_xi
    sta l_x1 ;x1 += xi;

    LDA l_d
    CLC
    ADC l_ai
    sta l_d ; d += ai;


;end else

_resztay

    LDA l_y1
    CLC
    ADC l_yi
    sta l_y1 ; y1 += yi;

;;// putpix(x1,y1);
    jsr debug
    jmp why1ry2
```

```
; koniec
l_end
    RTS
; PUTPIXEL should be called instead of this debug stuff for 6502sim
debug   ;rts
    LDY #10
    STy io_putc
    LDy #13
    STy io_putc

    LDY l_x1
    STy io_puth

    LDy #' '
    STy io_putc

    LDy l_y1
    STy io_puth
    rts
```