

2nd line FLI - another approach to twisters, x-rotators and waving carpets

by Bitbreaker/Oxyron^Arsenic^Nuance

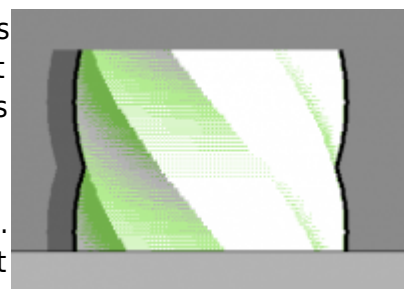
With FLI we can force another \$d018 value per line. However the forced DMA consumes additional 40 cycles, so not much else can be done when displaying FLI. When doing a FLI only every 2nd line we have another 63 cycles (PAL that is) available for doing a lot of fun things during display, like changing sprite registers, background colors, or even do things like updates in data being displayed. The difference to common FLI is, that you don't use \$d018 to flip in different screens for different colors for your bitmap, but use the FLI to change not only the screen but also the charset-pointer. That way you are able to display for e.g. 32 char wide chunks of graphics resembled by 8 half-filled charsets and 4 different screens. That gives us 32 individual chunks per bank. It is possible to extend the number of chunks by using more banks of course, but it is also possible to use other banks for interlacing 2 charsets together to get even smoother gradients. That is what i do in my examples. Feel free to squeeze in even more data by choosing another width and interleaving of screen and charset data.

So a bank could look like:

```
$0000-$03ff charset 1 ;chunk 0-3
$0400-$07ff screen 1
$0800-$0bff charset 2 ;chunk 4-7
$0c00-$0fff screen 2
$1000-$13ff charset 3 ;chunk 8-11
$1400-$17ff screen 3
$1800-$1bff charset 4 ;chunk 12-15
$1c00-$1fff screen 4
$2000-$23ff charset 5 ;chunk 16-19
$2400-$27ff free
$2800-$2bff charset 6 ;chunk 20-23
$2c00-$2fff free
$3000-$33ff charset 7 ;chunk 24-27
$3400-$37ff free
$3800-$3bff charset 8 ;chunk 28-31
$3c00-$3fff free
```

Twister

Imagine the basic shape behind a (untwisted) twister, do 32 animations substeps to make it rotate by one face and save 2 lines of that shape at that very certain step. With those line fragments gained by that process you can now reassemble any twisted form of the original shape, right? Now squeeze those fragments into 8 half-filled charsets, from which each \$0100 bytes resemble a 32 char wide chunk (see mapping above). Also fill the chars repeatedly in y-direction. When we now select charset



0 via \$d018 and also select a screen as source that contains the chars \$00..\$1f repeatedly in every line, we will display the first line fragment, whenever we switch \$d018 for e.g. to \$10.

So that is what we would have on the screen for a single line fragment:



And what it looks like with charset enabled:



Here's some c-code that will spit out the fakeshaded segments of a 8-sided shape as two interlaced charsets with 8 dithersteps:

```
#include <math.h>
#include <stdio.h>
#include <inttypes.h>

#define PI atan2 (0.0, -1.0)
#define RADIUS 64.0
#define DITHERSTEPS 8
#define SHADE (DITHERSTEPS * 3 + 1)
#define DEGREE (2.0 * PI / 360.0)

//our nicely interlaced dither patterns
static const uint8_t dither_patterns[9][8][4] = {
    {
        {0, 0, 0, 0}, {0, 0, 0, 0},
        {0, 0, 0, 0}, {0, 0, 0, 0},
        {0, 0, 0, 0}, {0, 0, 0, 0},
        {0, 0, 0, 0}, {0, 0, 0, 0},
    },
    {
        {1, 0, 1, 0}, {0, 0, 0, 0},
        {0, 0, 0, 0}, {0, 0, 0, 0},
        {0, 0, 0, 0}, {0, 1, 0, 1},
        {0, 0, 0, 0}, {0, 0, 0, 0},
    },
    {
        {1, 0, 1, 0}, {0, 0, 0, 0},
        {0, 1, 0, 1}, {0, 0, 0, 0},
        {0, 0, 0, 0}, {1, 0, 1, 0},
        {0, 0, 0, 0}, {0, 1, 0, 1},
    },
    {
        {1, 1, 1, 1}, {0, 0, 0, 0},
        {0, 1, 0, 1}, {0, 0, 0, 0},
        {0, 0, 0, 0}, {1, 1, 1, 1},
        {0, 0, 0, 0}, {1, 0, 1, 0},
    },
}
```

```

    },
    {
        {1, 1, 1, 1}, {0, 0, 0, 0},
        {0, 0, 0, 0}, {1, 1, 1, 1},
        {1, 1, 1, 1}, {0, 0, 0, 0},
        {0, 0, 0, 0}, {1, 1, 1, 1},
    },
    {
        {1, 1, 1, 1}, {1, 0, 1, 0},
        {1, 1, 1, 1}, {0, 0, 0, 0},
        {0, 1, 0, 1}, {1, 1, 1, 1},
        {0, 0, 0, 0}, {1, 1, 1, 1},
    },
    {
        {1, 1, 1, 1}, {1, 0, 1, 0},
        {1, 1, 1, 1}, {0, 1, 0, 1},
        {1, 0, 1, 0}, {1, 1, 1, 1},
        {0, 1, 0, 1}, {1, 1, 1, 1},
    },
    {
        {1, 1, 1, 1}, {1, 1, 1, 1},
        {1, 1, 1, 1}, {0, 1, 0, 1},
        {1, 1, 1, 1}, {1, 1, 1, 1},
        {1, 0, 1, 0}, {1, 1, 1, 1},
    },
    {
        {1, 1, 1, 1}, {1, 1, 1, 1},
        {1, 1, 1, 1}, {1, 1, 1, 1},
        {1, 1, 1, 1}, {1, 1, 1, 1},
        {1, 1, 1, 1}, {1, 1, 1, 1},
    }
};

void plot_pixel(uint8_t *charset, int x, int line, int luma) {
    int col1;
    int col2;
    int dith;

    int pos;
    int shift;

    int yy;
    int pix_pos;

    uint8_t byte;

    if(luma > 24) luma = 24;
    if(luma < 0) luma = 0;
    if(x >= 128) return;

    col1 = ((luma / DITHERSTEPS)) & 3;

```

```
col2 = (col1 + 1) & 3;

dith = luma % DITHERSTEPS;
pos = (line / 4 * 0x800) + (line & 0x3) * 32 * 8 + ((x * 2) & 0xf8);

pix_pos = x & 3;
shift = (3 - pix_pos) * 2;

for(yy = 0; yy < 4; yy++) {
    /* first frame */
    byte = charset[pos + yy] & (0xff ^ (3 << shift));
    if(dither_patterns[dith][yy * 2 + 0][pix_pos]) {
        byte |= (col2 << shift);
    } else {
        byte |= (col1 << shift);
    }
    charset[pos + yy + 0] = byte;
    charset[pos + yy + 4] = byte;
    /* second frame */
    byte = charset[0x4000 + pos + yy] & (0xff ^ (3 << shift));
    if(dither_patterns[dith][yy * 2 + 1][pix_pos]) {
        byte |= (col2 << shift);
    } else {
        byte |= (col1 << shift);
    }
    charset[0x4000 + pos + yy + 0] = byte;
    charset[0x4000 + pos + yy + 4] = byte;
}
}

void main() {
    int a;
    double deg;
    int x1, x2, x3, x4, x5;
    int z1, z2, z3, z4, z5;
    int x, line;
    int luma_dist;
    int x_dist;
    uint8_t charset[32768] = { 0 };
    FILE* fw;

    int xpos[256];

    int c;
    int pos;
    int b;
    int offset = 4;
    double zmin = cos(0 - 112.5 * DEGREE + 0 * 45 * DEGREE);
    double zmax = 1;
```

```

double scale = xmax - xmin;

line = 0;

//32 steps
for (a = 0; a < 32; a++) {
    deg = DEGREE * 45 * a / (32);
    xpos[line] = sin(deg - 112.5 * DEGREE + 0 * 45 * DEGREE) * RADIUS *
2 + RADIUS * 2 + 8 * offset - 128;

    //max. 5 visible edges on front side (8 sided shape)
    x1 = sin (deg - 112.5 * DEGREE + 0 * 45 * DEGREE) * RADIUS + RADIUS;
    z1 = (cos(deg - 112.5 * DEGREE + 0 * 45 * DEGREE) - xmin) / scale *
SHADE;
    x2 = sin (deg - 112.5 * DEGREE + 1 * 45 * DEGREE) * RADIUS + RADIUS;
    z2 = (cos(deg - 112.5 * DEGREE + 1 * 45 * DEGREE) - xmin) / scale *
SHADE;
    x3 = sin (deg - 112.5 * DEGREE + 2 * 45 * DEGREE) * RADIUS + RADIUS;
    z3 = (cos(deg - 112.5 * DEGREE + 2 * 45 * DEGREE) - xmin) / scale *
SHADE;
    x4 = sin (deg - 112.5 * DEGREE + 3 * 45 * DEGREE) * RADIUS + RADIUS;
    z4 = (cos(deg - 112.5 * DEGREE + 3 * 45 * DEGREE) - xmin) / scale *
SHADE;
    x5 = sin (deg - 112.5 * DEGREE + 4 * 45 * DEGREE) * RADIUS + RADIUS;
    z5 = (cos(deg - 112.5 * DEGREE + 4 * 45 * DEGREE) - xmin) / scale *
SHADE;

    for(x = 0; x < x1; x++) {
        plot_pixel(charset, x, line, 0);
    }
    for(x = x1; x < x2; x++) {
        plot_pixel(charset, x, line, z1 + (z2 - z1) * (x1 - x) / (x2 -
x1));
    }
    for(x = x2; x < x3; x++) {
        plot_pixel(charset, x, line, z2 + (z3 - z2) * (x2 - x) / (x3 -
x2));
    }
    for(x = x3; x < x4; x++) {
        plot_pixel(charset, x, line, z3 + (z4 - z3) * (x3 - x) / (x4 -
x3));
    }
    for(x = x4; x < x5; x++) {
        plot_pixel(charset, x, line, z4 + (z5 - z4) * (x4 - x) / (x5 -
x4));
    }
    for(x = x5; x < RADIUS * 2; x++) {
        plot_pixel(charset, x, line, 0);
    }
    line++;
}

```

```
//generate screens + sprite-pointers
for (a = 0; a < 8; a++) {
    pos = 0x400 + a * 0x800;
    for (b = 0; b < 0x3f0; b++) {
        charset[pos + b] = 0xff;
        charset[0x4000 + pos + b] = 0xff;
    }
    for (b = 0; b < 25; b++) {
        for (c = 0; c < 32; c++) {
            if(a > 4) x = 0xfe;
            else x = c + a * 32;
            charset[pos + offset + b * 40 + c] = x;
            charset[0x4000 + pos + offset + b * 40 + c] = x;
        }
    }
    for (b = 0x3f8; b < 0x400; b+=2) {
        charset[pos + b] = 0xf0;
        charset[0x4000 + pos + b] = 0xf0;
        charset[pos + b + 1] = 0xf1;
        charset[0x4000 + pos + b + 1] = 0xf1;
    }
}

//generate sprite data for left and right cover sprite
for(a = 0; a < 63; a += 3) {
    charset[0x3c00 + a + 0] = charset[0x7c00 + a + 0] = 0xff;
    charset[0x3c00 + a + 1] = charset[0x7c00 + a + 1] = 0xea;
    charset[0x3c00 + a + 2] = charset[0x7c00 + a + 2] = 0xa9;

    charset[0x3c40 + a + 0] = charset[0x7c40 + a + 0] = 0x7f;
    charset[0x3c40 + a + 1] = charset[0x7c40 + a + 1] = 0xff;
    charset[0x3c40 + a + 2] = charset[0x7c40 + a + 2] = 0xff;
}

//write out stuff
fw = fopen("5col.data", "wb");
c = 0x00;
fwrite(&c,1,1,fw);
c = 0x40;
fwrite(&c,1,1,fw);
fwrite(&charset[0],1,32768,fw);
fclose(fw);
}
```

And now comes the 6502 part to reassemble twisted shapes with that data. For that we just draw virtual lines where the y-position is the y-position on the screen and the x-position is the line segment to display. So a cheap bresenham will help out. The steeper our slope is, the straighter the shape appears, the flatter our slope is, the more it will appear twisted. Also we make the twister appear

more colorful by using all 4 charset-colors for the twister's shape only and not wasting one color for the area outside of the twister. For covering the FLI-bug and the area outside of the shape we use expanded sprites for which we update the x and y-positions every second line (yes, no stretcher needed here, can be done even cheaper). For making the outline smoother we set the x-position of the sprites with an accuracy of 1 pixel. Also we use multicolor mode for the sprites. That enables us to use one color for covering the area, another (black) for doing a stylish outline and even one more for casting a fake shadow, that follows the outline of the twister for free. Now this thingy looks pretty colorful! That's really all? Yes! You can now feel free to fade around the twister by changing the charset-, sprite- and border-colors at any fashion.

Here's the code:

```
        *= $3000

y1      = $40
y2      = $41
x1      = $42
x2      = $43
err     = $44
clk     = $46
dy     = $47
pos     = $48
step    = $49
fade    = $52
offset  = $51

        jmp start
irq1
        dec $d019
        ldy #$00
loop0
        ;do fli
sta18   lda #$00
        sta $d018
sta11   lda #$1b
        sta $d011

        ;prepare next values
        lda tab11,y
        sta stall+1
        lda tab18,y
        sta stal8+1

        ;wait for right moment
        bit $ea

        ;set x-positions of cover sprite 1
        lda xpos,y
        ;even too lazy to add an offset to the table, as we have enough
        ;cycles available
        adc #$66
```

```
    sta $d000
    adc #$c0
    eor #$ff

    ;wait a bit, so that is cocky, right? :-)
    nop
    nop

    ;advance y-position of sprites
    ldx ypos,y
    stx $d001
    stx $d003

    ;set x-positions of cover sprite 2
    iny
    sta $d002

    ;enough cycles left to enjoy the luxury of a loop
    cpy #100
    bcc loop0

coll  ;all lines done, display something sane
    lda #$01
    sta $d020
    lda #$f0
    sta $d018
    lda #$50
    sta $d011

    ;interlace between both banks
    lda $dd00
    and #$03
    eor #$02
    sta $dd00

    ;even do a $d016 shift
    lda $d016
    eor #$01
    sta $d016

    ;our fancy rasterline
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
```



```
col2    lda #$01
        sta $d020

        inc clk

        ;update tables and colors
        jsr update
        jsr colors
col3    lda #$01
        sta $d020
        ;jsr $1003

        ;return from irq
        pla
        tay
        pla
        tax
        pla
        rti

start   sei
        ;sync and turn off screen
        lda $d011
        bpl *-3
        lda $d011
        bmi *-3
        lda #$0b
        sta $d011
        lda $d011
        bpl *-3
        lda $d011
        bmi *-3

        ;set up $d011 table
        ldx #$00
loop2   txa
        asl
        ora #$01
        and #$07
        ora #$10
        sta tab11,x
        inx
        bne loop2

        ;set up colors
        lda #$09
        ldx #$00
loop3   sta $d800,x
```

```
    sta $d900,x
    sta $da00,x
    sta $db00,x
    dex
    bne loop3

;init values
    lda #$00
    sta x1
    lda #$00
    sta y1
    lda #$00
    sta x2
    lda #99
    sta y2

    ldx #$d0
    stx clk

    ldx #$00
    stx pos
    stx step
    stx fade
    stx pos
    stx offset

;copy 2nd bank
    lda #$34
    sta $01
    ldx #$3f
    ldy #$00
-
src    lda $8000,y
dst    sta $c000,y
    dey
    bne -
    inc src+2
    inc dst+2
    dex
    bne -
    inc $01

;fade to white
-
    ldx #$04
    jsr wait
    lda fadec,y
    sta $d020
```

```
    iny
    cpy #$07
    bne -

    ;create display tables for the first time
    jsr update

    ;vsync
    lda $d011
    bpl *-3
    lda $d011
    bmi *-3

    ;copy last bytes now to not destroy any still active irq-pointers @
$fffe
    ldy #$00
    -
    lda $bf00,y
    sta $ff00,y
    dey
    bne -

    ;now use irq @ vector $0314, but we could also just use the vector
@ $fffe/f as long as we have no needed data there
    sei
    lda #$37
    sta $01
    lda #$7f
    sta $dc0d
    lda $dc0d
    lda #$0b
    sta $d011
    lda #$30
    sta $d012
    lda #<irq1
    sta $0314
    lda #>irq1
    sta $0315
    lda #$01
    sta $d01a

    ;setup sprites and colors and things
    lda #$01
    sta $d025
    sta $d026
    sta $d027
    sta $d028
    sta $d021
    sta $d022
    sta $d023
    sta $d020
```

```
    lda #$03
    sta $d015
    lda #$03
    sta $d017
    sta $d01d
    sta $d01c
    lda #$32
    sta $d001
    sta $d003
    lda #$18
    sta $d000
    lda #$28
    sta $d002
    lda #$02
    sta $d010
    ldx #$f0
    stx $7ff8
    stx $7ff9
    lda #$02
    sta $dd00
    lda #$18
    sta $d016
    cli
```

```
    jmp *
```

fadec

```
    !byte $00,$09,$08,$0a,$0f,$07,$01
```

wait

```
    lda $d011
    bmi *-3
    lda $d011
    bpl *-3
    lda #$30
    cmp $d012
    bne *-3
    dex
    bne wait
    rts
```

update

```
    inc offset
    inc offset
    lda offset
    cmp #64
    bcc *+6
    lda #$00
    sta offset
```

```
    lda clk
    and #$01
    bne step00

    ;decide what to update (upper x pos, lower x pos, move xpos to
left/right)
    lda step
    cmp #$00
    beq step0
    cmp #$01
    beq step1
    cmp #$02
    beq step2
    cmp #$03
    beq step3
    lda #$00
    sta step
    jmp update
step0
    inc x2
    lda x2
    cmp #99
    bne step00
    inc step
step00
    jmp drawline
step1
    inc x1
    dec x2
    lda x2
    bne step10
    inc step
step10
    jmp drawline
step2
    dec x1
    inc x2
    lda x2
    cmp #99
    bne step20
    inc step
step20
    jmp drawline
step3
    dec x2
    lda x2
    bne step30
    inc step
step30
```

```
        ;jmp drawline

drawline
        ;setup bresenham (dx/dy, inx/dex)
        lda y2
        sta toy+1
        sec
        sbc y1
        sta ty2+1
        lsr
        sta err

        ldx #$e8
        lda x2
        sec
        sbc x1
        bcs ov2
        eor #$ff
        adc #$01
        ldx #$ca

ov2
        stx incx2
        sta tx2+1

        lda x1
        clc
        adc offset
        tax

        ;bresenham to calc slope
        ldy y1

loopy
        lda mytab18,x
        sta tab18,y
        lda myxpos,x
        sta xpos,y
        lda err
        sec

tx2
        sbc #$00
        bcs +

ty2
        adc #$00
incx2
        inx
+
        sta err
        iny

toy
        cpy #$00
        bne loopy
        rts
```

colors

```
;all the color fadings
```

```
lda clk
cmp #$e0
bcs *+3
rts
```

```
and #$03
bne ++
```

```
lda fade
cmp #$28
bne +
lda #$00
sta fade
beq *+2
```

```
+
```

```
inc fade
tax
lda fade1,x
sta $d021
lda fade2,x
sta $d022
lda fade3,x
sta $d023
lda fade00,x
sta $d025
sta col1+1
lda fade0b,x
sta $d027
sta $d028
lda fade0c,x
sta col3+1
sta $d026
lda fade0f,x
sta col2+1
```

```
++
```

```
rts
```

```
;fading tables
```

fade0f

```
!byte $01,$01,$01,$01
!byte $01,$01,$01,$0f
!byte $0f,$0f,$0f,$0f
!byte $0f,$0f,$0f,$0f
!byte $0f,$0f,$0f,$0f
!byte $0f,$0f,$0f,$0f
!byte $0f,$0f,$0f,$0f
!byte $0f,$0f,$0f,$0f
!byte $0f,$0f,$0f,$0f
!byte $0f,$01,$01,$01
!byte $01,$01,$01,$01
```

fade00

```
!byte $01,$01,$01,$01
!byte $0f,$0c,$0b,$00
!byte $00,$00,$00,$00
!byte $00,$00,$00,$00
!byte $00,$00,$00,$00
!byte $00,$00,$00,$00
!byte $00,$00,$00,$00
!byte $00,$00,$00,$00
!byte $00,$00,$00,$00
!byte $00,$0b,$0c,$0f
!byte $01,$01,$01,$01
```

fade0c

```
!byte $01,$01,$01,$01
!byte $01,$01,$0f,$0c
!byte $0c,$0c,$0c,$0c
!byte $0c,$0c,$0c,$0c
!byte $0c,$0c,$0c,$0c
!byte $0c,$0c,$0c,$0c
!byte $0c,$0c,$0c,$0c
!byte $0c,$0c,$0c,$0c
!byte $0c,$0c,$0c,$0c
!byte $0c,$0f,$01,$01
!byte $01,$01,$01,$01
```

fade0b

```
!byte $01,$01,$01,$01
!byte $01,$0f,$0c,$0b
!byte $0b,$0b,$0b,$0b
!byte $0b,$0b,$0b,$0b
!byte $0b,$0b,$0b,$0b
!byte $0b,$0b,$0b,$0b
!byte $0b,$0b,$0b,$0b
!byte $0b,$0b,$0b,$0b
!byte $0b,$0b,$0b,$0b
!byte $0b,$0c,$0f,$01
!byte $01,$01,$01,$01
```

fade1

```
!byte $01,$01,$01,$01
!byte $01,$0d,$0f,$05
!byte $05,$0f,$0d,$01
!byte $01,$07,$0f,$0a
!byte $0a,$0f,$07,$01
!byte $01,$0d,$03,$0e
!byte $0e,$03,$0d,$01
!byte $01,$07,$0f,$0a
!byte $0a,$0f,$07,$01
!byte $01,$01,$01,$01
```

fade2

```
!byte $01,$01,$01,$01
!byte $01,$01,$0d,$0f
!byte $0f,$0d,$01,$01
```



```
!byte $01,$01,$07,$0f
!byte $0f,$07,$01,$01
!byte $01,$01,$0d,$03
!byte $03,$0d,$01,$01
!byte $01,$01,$07,$0f
!byte $0f,$07,$01,$01
!byte $01,$01,$01,$01
```

fade3

```
!byte $01,$01,$01,$01
!byte $01,$01,$01,$0d
!byte $0d,$01,$01,$01
!byte $01,$01,$01,$07
!byte $07,$01,$01,$01
!byte $01,$01,$01,$0d
!byte $0d,$01,$01,$01
!byte $01,$01,$01,$07
!byte $07,$01,$01,$01
!byte $01,$01,$01,$01
```

```
;sprite y-positions
```

ypos

```
!byte $32,$32,$32,$32
!byte $32,$32,$32,$32
!byte $32,$32,$32,$32
!byte $32,$32,$32,$32
!byte $32,$32,$32,$32
!byte $32
```

```
!byte $5c,$5c,$5c,$5c
!byte $5c,$5c,$5c,$5c
!byte $5c,$5c,$5c,$5c
!byte $5c,$5c,$5c,$5c
!byte $5c,$5c,$5c,$5c
!byte $5c
```

```
!byte $86,$86,$86,$86
!byte $86,$86,$86,$86
!byte $86,$86,$86,$86
!byte $86,$86,$86,$86
!byte $86,$86,$86,$86
!byte $86
```

```
!byte $b0,$b0,$b0,$b0
!byte $b0,$b0,$b0,$b0
!byte $b0,$b0,$b0,$b0
!byte $b0,$b0,$b0,$b0
!byte $b0,$b0,$b0,$b0
!byte $b0
```

```
!byte $da,$da,$da,$da
!byte $da,$da,$da,$da
```

```
!byte $da,$da,$da,$da
!byte $da,$da,$da,$da
!byte $da,$da,$da,$da
!byte $da
```

;corresponding \$d018 values for each fragment

mytab18

```
!byte $10,$30,$50,$70
!byte $12,$32,$52,$72
!byte $14,$34,$54,$74
!byte $16,$36,$56,$76
!byte $18,$38,$58,$78
!byte $1a,$3a,$5a,$7a
!byte $1c,$3c,$5c,$7c
!byte $1e,$3e,$5e,$7e
```

```
!byte $10,$30,$50,$70
!byte $12,$32,$52,$72
!byte $14,$34,$54,$74
!byte $16,$36,$56,$76
!byte $18,$38,$58,$78
!byte $1a,$3a,$5a,$7a
!byte $1c,$3c,$5c,$7c
!byte $1e,$3e,$5e,$7e
```

```
!byte $10,$30,$50,$70
!byte $12,$32,$52,$72
!byte $14,$34,$54,$74
!byte $16,$36,$56,$76
!byte $18,$38,$58,$78
!byte $1a,$3a,$5a,$7a
!byte $1c,$3c,$5c,$7c
!byte $1e,$3e,$5e,$7e
```

```
!byte $10,$30,$50,$70
!byte $12,$32,$52,$72
!byte $14,$34,$54,$74
!byte $16,$36,$56,$76
!byte $18,$38,$58,$78
!byte $1a,$3a,$5a,$7a
!byte $1c,$3c,$5c,$7c
!byte $1e,$3e,$5e,$7e
```

```
!byte $10,$30,$50,$70
!byte $12,$32,$52,$72
!byte $14,$34,$54,$74
!byte $16,$36,$56,$76
!byte $18,$38,$58,$78
!byte $1a,$3a,$5a,$7a
```

```
!byte $1c,$3c,$5c,$7c
!byte $1e,$3e,$5e,$7e
```

```
!byte $10,$30,$50,$70
!byte $12,$32,$52,$72
!byte $14,$34,$54,$74
!byte $16,$36,$56,$76
!byte $18,$38,$58,$78
!byte $1a,$3a,$5a,$7a
!byte $1c,$3c,$5c,$7c
!byte $1e,$3e,$5e,$7e
```

```
!byte $10,$30,$50,$70
!byte $12,$32,$52,$72
!byte $14,$34,$54,$74
!byte $16,$36,$56,$76
!byte $18,$38,$58,$78
!byte $1a,$3a,$5a,$7a
!byte $1c,$3c,$5c,$7c
!byte $1e,$3e,$5e,$7e
```

```
!byte $10,$30,$50,$70
!byte $12,$32,$52,$72
!byte $14,$34,$54,$74
!byte $16,$36,$56,$76
!byte $18,$38,$58,$78
!byte $1a,$3a,$5a,$7a
!byte $1c,$3c,$5c,$7c
!byte $1e,$3e,$5e,$7e
```

;xpos table for sprite (TODO: should also be generated)

myxpos

```
!byte $aa,$a9,$a8,$a7,$a6,$a5
!byte $a4,$a4,$a3,$a2,$a2,$a1
!byte $a1,$a1,$a1,$a1,$a0,$a1
!byte $a1,$a1,$a1,$a1,$a2,$a2
!byte $a3,$a4,$a4,$a5,$a6,$a7
!byte $a8,$a9
```

```
!byte $aa,$a9,$a8,$a7,$a6,$a5
!byte $a4,$a4,$a3,$a2,$a2,$a1
!byte $a1,$a1,$a1,$a1,$a0,$a1
!byte $a1,$a1,$a1,$a1,$a2,$a2
!byte $a3,$a4,$a4,$a5,$a6,$a7
!byte $a8,$a9
```

```
!byte $aa,$a9,$a8,$a7,$a6,$a5
!byte $a4,$a4,$a3,$a2,$a2,$a1
!byte $a1,$a1,$a1,$a1,$a0,$a1
!byte $a1,$a1,$a1,$a1,$a2,$a2
!byte $a3,$a4,$a4,$a5,$a6,$a7
```

```
!byte $a8,$a9

!byte $aa,$a9,$a8,$a7,$a6,$a5
!byte $a4,$a4,$a3,$a2,$a2,$a1
!byte $a1,$a1,$a1,$a1,$a0,$a1
!byte $a1,$a1,$a1,$a1,$a2,$a2
!byte $a3,$a4,$a4,$a5,$a6,$a7
!byte $a8,$a9

!byte $aa,$a9,$a8,$a7,$a6,$a5
!byte $a4,$a4,$a3,$a2,$a2,$a1
!byte $a1,$a1,$a1,$a1,$a0,$a1
!byte $a1,$a1,$a1,$a1,$a2,$a2
!byte $a3,$a4,$a4,$a5,$a6,$a7
!byte $a8,$a9

!byte $aa,$a9,$a8,$a7,$a6,$a5
!byte $a4,$a4,$a3,$a2,$a2,$a1
!byte $a1,$a1,$a1,$a1,$a0,$a1
!byte $a1,$a1,$a1,$a1,$a2,$a2
!byte $a3,$a4,$a4,$a5,$a6,$a7
!byte $a8,$a9

!byte $aa,$a9,$a8,$a7,$a6,$a5
!byte $a4,$a4,$a3,$a2,$a2,$a1
!byte $a1,$a1,$a1,$a1,$a0,$a1
!byte $a1,$a1,$a1,$a1,$a2,$a2
!byte $a3,$a4,$a4,$a5,$a6,$a7
!byte $a8,$a9

!byte $aa,$a9,$a8,$a7,$a6,$a5
!byte $a4,$a4,$a3,$a2,$a2,$a1
!byte $a1,$a1,$a1,$a1,$a0,$a1
!byte $a1,$a1,$a1,$a1,$a2,$a2
!byte $a3,$a4,$a4,$a5,$a6,$a7
!byte $a8,$a9

;the final tables to be displayed
*= $3d00
xpos
*= $3e00
tab18
*= $3f00
tab11

;include generated data

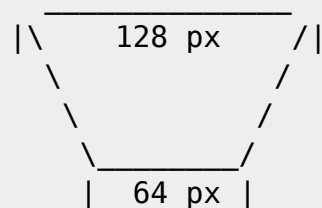
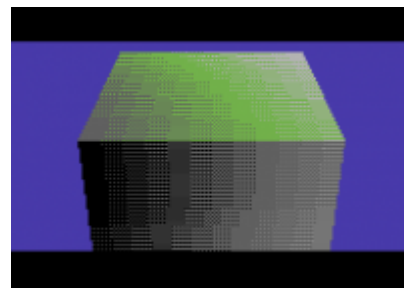
* = $4000
!bin "5col.data", $4000, $0002
```

```
* = $8000
!bin "5col.data", $4000, $4002
```

Now try this with ECM-mode and hires, you'll need to optimize the chunks and reuse chars from the charsets to save space as we can only use the first \$200 bytes of each charset.

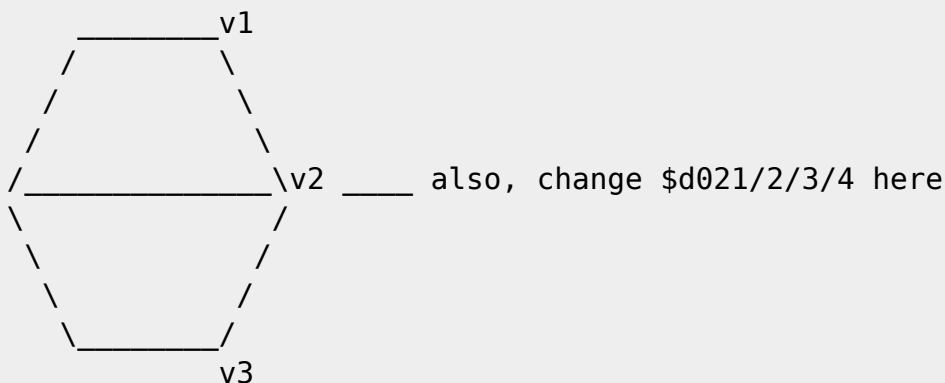
X-rotating cube

Now, having all that knowledge from doing a twister with 2nd line FLI, one can do of course also other shapes with that. How's about a x-rotating cube? So let's take the following quad:



This resembles a face that would go from the maximum width you wish the cube to have, to the minimum width the cube should have (usually 2*32 pixel less than the maximum, so each line gets one pixel smaller on both sides). Now rasterize it the same way as you would do for the twister, so you end up with 32 different lines ranging from 128 pixel (multicolor) to 64 pixel width.

Cube frontview



Depending on the slope, you can now again reassemble the lines to form 2 perspective faces watched from the front. So all we need to calculate are the slopes v1→v2 and v2→v3, build our display tables from that and we have a nice rotating cube. The rotated values for v1/v2/v3 can be easily calculated by a lookup into a sine-table. To make the shape of the cube more obvious we change the colors for the charset whenever the last line for each face is displayed and we end up with a cube that has

different colors on each face. Now we can get fancy again and add an y-offset to all slopes to make the cube stomp and add some fading to fake some moving light. Only a good fake is a good make!

And here comes some code:

```
!cpu 6510
    *= $3000

y1      = $40
y2      = $41
x1      = $42
x2      = $43
err     = $44
clk     = $46
dy      = $47
pos     = $48
stpos  = $49
offs   = $4a
col    = $4b
shift  = $4c
dest   = $50
fcnt_l = $52
fcnt_h = $53
c_offs = $60

    jmp start

irq1
    dec $d019
    lda #$06
    ldx $d012
    inx
    cpx $d012
    bne *-3
    ldy #$0a
    dey
    bne *-1
    inx
    cpx $d012
    nop
    beq time1
    nop
    bit $ea

time1
    ldy #$09
    dey
    bne *-1
    nop
    nop
    inx
```

```

        cpx $d012
        nop
        beq time2
        bit $ea
time2
        ldy #$0a
        dey
        bne *-1
        inx
        cpx $d012
        bne time3
time3
        ldy #$00
        bit $ea
        ldx #$18
        stx $d011
        sta $d020

loop0
poi18  lda $1000,y
        sta $d018
        lda tab11,y
        sta $d011
        stx $ea           ; -> sets FLI-bug color

split  cpy #$00
        bcc nosplit
colu2  lda #$06
        sta $d022
colu3  lda #$06
        sta $d023
colu1  lda #$06
        sta $d021
        jmp in1
nosplit
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        lda $1000
in1    lda $1000
        lda $1000
        lda $1000
        lda $1000
        lda $1000
```

```
        lda $1000
        lda $1000
        bit $ea
        nop
        nop

        iny
        cpy #100
        bcc loop0

        lda #$f0
        sta $d018
        lda #$00
        sta $d020
        lda #$50
        sta $d011
        lda $dd00
        eor #$02
        sta $dd00
colo2   lda #$06
        sta $d022
colo3   lda #$06
        sta $d023
colo1   lda #$06
        sta $d021

        ;jsr $1003

        inc clk
        lda clk
        and #$01
        bne +
        jsr update
        jsr cols
        inc fcnt_l
        bne +
        inc fcnt_h
+
        pla
        tay
        pla
        tax
        pla
        rti

start
        ;copy 2nd bank
        sei
```



```

        lda #$34
        sta $01
        ldx #$40
        ldy #$00
-
sr01    lda bank2,y
tg01    sta $c000,y
        dey
        bne -
        inc sr01+2
        inc tg01+2
        dex
        bne -

        lda #$37
        sta $01
        cli

        ;generate values for $d011
        ldx #$00
loop2   txa
        sec ;a = a * 2 + 1
        rol
        and #$07
        ora #$18
        sta tab11,x
        inx
        bne loop2

        ;populate tables for the first time
        jsr update

        ;set colram
        lda #$0e
        ldx #$00
loop3   sta $d800,x
        sta $d900,x
        sta $da00,x
        sta $db00,x
        dex
        bne loop3

        ;init variables
        ldx #$00
txy     stx fcnt_l
        stx fcnt_h
        stx clk
        stx pos

```

```
    stx stpos
    stx col
    stx shift
    stx c_offs
loopdl
    ;create all slopes needed
    stx pos
    jsr doline
    ldx pos
    inx
    cpx #$10
    bne loopdl
    ldx #$00
    stx pos

    ;vsync
    lda $d011
    bpl *-3
    lda $d011
    bmi *-3

    lda #$02
    sta $dd00
    lda #$18
    sta $d016

    ;set colors for the first time
    jsr cols

    ;set up irq
    sei
    lda #$37
    sta $01
    lda #$7f
    sta $dc0d
    lda #$1b
    sta $d011
    lda #$2c
    sta $d012
    lda #<irq1
    sta $0314
    lda #>irq1
    sta $0315
    lda #$01
    sta $d019
    sta $d01a
    cli

    ;fade in
```

```

        ;wait 4 frames
-
        lda fcnt_l
        and #$03
        bne -
        lda fcnt_l
        ;wait until next frame starts
--
        cmp fcnt_l
        beq --
        inc c_offs
        lda c_offs
        cmp #$08
        bne -
        jmp *

cols
        ;set up colors
        ldy shift
        lda col           ;which colors to choose (face 1..4)
        clc
        adc spos,y       ;add x-movement depending on counter shift
        tay

        ;colors of first face shown
        lda coltab+0,y   ;fetch index into fadings tab
        adc c_offs      ;add colorfade offset
        tax              ;use as index
        lda fadings,x   ;fetch corresponding color
        sta colu3+1     ;set color
        lda coltab+1,y  ;same for all other colors
        adc c_offs
        tax
        lda fadings,x
        sta colu2+1
        lda coltab+2,y
        adc c_offs
        tax
        lda fadings,x
        sta colu1+1

        ;add offset to shift and wrap around in case -> we advance to the
        colors of next face
        tya
        clc
        adc #$08
        and #$1f
        tay

        ;same for other face shown
        lda coltab+0,y

```

```
    adc c_offs
    tax
    lda fadings,x
    sta colo3+1
    lda coltab+1,y
    adc c_offs
    tax
    lda fadings,x
    sta colo2+1
    lda coltab+2,y
    adc c_offs
    tax
    lda fadings,x
    sta colo1+1

    ;advance counter
    inc shift
    lda shift
    cmp #$3c
    bne *+6
    lda #$00
    sta shift
    rts
```

spos

```
!byte $00,$00,$00,$00,$00,$00
!byte $00,$00,$00,$00,$00,$00
!byte $00,$00,$00,$00,$00,$00
!byte $00,$00,$00,$00,$00,$00
!byte $00,$00,$00,$00,$00,$00
!byte $01,$01,$02,$03,$04,$04
!byte $04,$05,$05,$05,$05,$05
!byte $05,$05,$05,$05,$05,$05
!byte $05,$05,$05,$04,$04,$04
!byte $03,$02,$01,$01,$00,$00
```

update

```
    ;walk through slopes
    inc pos
    lda pos
    and #$0f
    sta pos
    cmp #$0f
    bne +
    sta stpos
    jmp ++
```

+

```
    cmp #$00
    bne ++
```

```
    ;advance to colors of next face
    lda col
    clc
    adc #$08
    and #$1f
    sta col
++
    ;calc destination pointer
    jsr calcdest
    ;set up start of $d018 table
    lda dest
    sta poi18+1
    lda dest+1
    sta poi18+2

    ;and pick fitting splittab entry
    ldx pos
    lda splittab,x
    sta split+1
    rts

calcdest
    ;dest = pos * $80 + slopes
    ;-> dest = (pos * $100 + (2 * slopes) / 2)
    lda pos
    clc
    adc #>(slopes) * 2
    lsr
    sta dest+1
    arr #$00    ;moves carry to MSB of lowbyte, aka lda #$00 + ror
    sta dest
    rts

doline
    ;calc location to store slope data
    jsr calcdest

    ;calculate offset for stomping/marching
    lda sine+$08,x
    asl
    sta offs
    lda #$7f
    sec
    sbc offs
    sta offs

    ;slope for face 1
    lda sine+$28,x
    asl
    clc
```

```
        adc offs
        sta y1
        lda sine+$38,x
        asl
        clc
        adc offs
        sta y2
        lda sine+$18,x
        lsr
        sta x1
        lda sine+$28,x
        lsr
        sta x2

        ;blank all area above face 1
        ldy #$00
        lda #$9c
cl1     sta (dest),y
        iny
        cpy y2
        bne cl1

        jsr drawline

        ldx pos

        ;slope for face 2
        lda sine+$38,x
        asl
        clc
        adc offs
        sta y1
        sta splittab,x
        lda sine+$08,x
        asl
        clc
        adc offs
        sta y2
        lda sine+$28,x
        lsr
        sta x1
        lda sine+$38,x
        lsr
        sta x2

        jsr drawline
        rts
```

```

drawline
    ;calc dy/dx and decide if we increment or decrement x
    lda y2
    sta toy+1
    sec
    sbc y1
    sta dy

    ldx #$e8
    lda x2
    sta tox+1
    sec
    sbc x1
    bcs ov2
    eor #$ff
    adc #$01
    ldx #$ca
ov2
    stx incx1
    stx incx2

    ldx x1
    ldy y1

    cmp dy
    bcc steep

    ;flat slope
    sta tx1+1
    lsr
    sta err
    lda dy
    sta ty1+1
loopx
    lda mytab,x
    sta (dest),y

    lda err
    sec
ty1
    sbc #$00
    bcs ov3
tx1
    adc #$00
incyl
    iny
ov3
    sta err
incx1
    inx
tox
    cpx #$00
    bne loopx
    rts

    ;we have a steep slope

```

```
steep
    sta tx2+1
    lda dy
    sta ty2+1
    lsr
    sta err

loopy
    lda mytab,x
    sta (dest),y
    lda err
    sec

tx2
    sbc #$00
    bcs ov4

ty2
    adc #$00
incx2
    inx
ov4
    sta err

incy2
    iny
toy
    cpy #$00
    bne loopy
    rts

splittab
    !byte $00,$00,$00,$00
    !byte $00,$00,$00,$00
    !byte $00,$00,$00,$00
    !byte $00,$00,$00,$00

coltab
    !byte $40,$e0,$30,$70
    !byte $30,$e0,$40,$60

    !byte $a0,$f0,$70,$10
    !byte $70,$f0,$a0,$80

    !byte $b0,$c0,$f0,$10
    !byte $f0,$c0,$b0,$00

    !byte $50,$f0,$d0,$10
    !byte $d0,$f0,$50,$b0

    ;sine, two times, so we don't have to cope with wrap arounds

sine
    !byte $20,$23,$26,$29,$2c,$2f
    !byte $31,$34,$36,$38,$3a,$3c
    !byte $3d,$3e,$3f,$3f,$3f,$3f
    !byte $3f,$3e,$3d,$3c,$3a,$38
    !byte $36,$34,$31,$2f,$2c,$29
    !byte $26,$23,$20,$1c,$19,$16
```



```
!byte $13,$10,$0e,$0b,$09,$07
!byte $05,$03,$02,$01,$00,$00
!byte $00,$00,$00,$01,$02,$03
!byte $05,$07,$09,$0b,$0e,$10
!byte $13,$16,$19,$1c
!byte $20,$23,$26,$29,$2c,$2f
!byte $31,$34,$36,$38,$3a,$3c
!byte $3d,$3e,$3f,$3f,$3f,$3f
!byte $3f,$3e,$3d,$3c,$3a,$38
!byte $36,$34,$31,$2f,$2c,$29
!byte $26,$23,$20,$1c,$19,$16
!byte $13,$10,$0e,$0b,$09,$07
!byte $05,$03,$02,$01,$00,$00
!byte $00,$00,$00,$01,$02,$03
!byte $05,$07,$09,$0b,$0e,$10
!byte $13,$16,$19,$1c
```

;color fadings tabs for each specific color

fadings

```
!byte $06,$00,$00,$00,$00,$00,$00,$00
!byte $00,$00,$00,$00,$00,$00,$00,$06

!byte $06,$0b,$0c,$0f,$01,$01,$01,$01
!byte $01,$01,$01,$01,$0f,$0c,$0b,$06

!byte $06,$02,$02,$02,$02,$02,$02,$02
!byte $02,$02,$02,$02,$02,$02,$02,$06

!byte $06,$06,$04,$0e,$03,$03,$03,$03
!byte $03,$03,$03,$03,$0e,$04,$06,$06

!byte $06,$06,$04,$04,$04,$04,$04,$04
!byte $04,$04,$04,$04,$04,$04,$06,$06

!byte $06,$0b,$0c,$05,$05,$05,$05,$05
!byte $05,$05,$05,$05,$05,$0c,$0b,$06

!byte $06,$06,$06,$06,$06,$06,$06,$06
!byte $06,$06,$06,$06,$06,$06,$06,$06

!byte $06,$09,$08,$0a,$0f,$07,$07,$07
!byte $07,$07,$07,$0f,$0a,$08,$09,$06

!byte $06,$09,$08,$08,$08,$08,$08,$08
!byte $08,$08,$08,$08,$08,$08,$09,$06

!byte $06,$09,$09,$09,$09,$09,$09,$09
!byte $09,$09,$09,$09,$09,$09,$09,$06

!byte $06,$09,$08,$0a,$0a,$0a,$0a,$0a
!byte $0a,$0a,$0a,$0a,$0a,$08,$09,$06
```

```
!byte $06,$0b,$0b,$0b,$0b,$0b,$0b,$0b
!byte $0b,$0b,$0b,$0b,$0b,$0b,$0b,$06
```

```
!byte $06,$0b,$0c,$0c,$0c,$0c,$0c,$0c
!byte $0c,$0c,$0c,$0c,$0c,$0c,$0b,$06
```

```
!byte $06,$06,$0c,$05,$03,$0d,$0d,$0d
!byte $0d,$0d,$0d,$03,$05,$0c,$06,$06
```

```
!byte $06,$06,$04,$0e,$0e,$0e,$0e,$0e
!byte $0e,$0e,$0e,$0e,$0e,$04,$06,$06
```

```
!byte $06,$0b,$0c,$0f,$0f,$0f,$0f,$0f
!byte $0f,$0f,$0f,$0f,$0f,$0c,$0b,$06
```

;d018 values depending for each fragment

mytab

```
!byte $10,$30,$50,$70
!byte $12,$32,$52,$72
!byte $14,$34,$54,$74
!byte $16,$36,$56,$76
!byte $18,$38,$58,$78
!byte $1a,$3a,$5a,$7a
!byte $1c,$3c,$5c,$7c
!byte $1e,$3e,$5e,$7e
```

```
!align 255,0
```

tab11

```
* = $3800
```

slopes

```
* = $4000
```

```
!bin "stomp.data", $4000, 2
```

```
* = $7400
```

```
!fill $0100, $ff
```

```
* = $8000
```

bank2

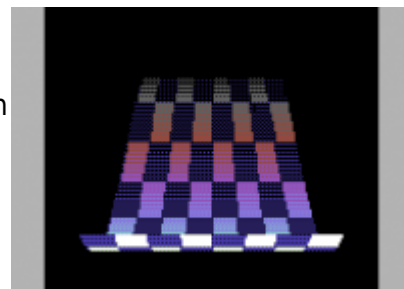
```
!bin "stomp.data", $3ff8, $4002
```

```
* = $b400
```

```
!fill $0100, $ff
```

Waving chessboard carpet

At this kind of effect we concentrate on updating the multicolor registers each second line, while avoiding ugly gray dots. Also we use the free cycles during display to clear the z-table that is generated each frame. Here, the displayed chunk is determined by the z-position of each line. Calculations start with the deepest z-position of the carpet, the y-position is thereby fetched from a sine-table. Now if a closer line is at the same y-position it will automatically cover lines below, as the value in the final table is simply overwritten. Each chunk has also certain colors bonded to it. Also the bitmap is created that way, that we can get a continuous gradient by continuously interleaving 2 colors. This way the gradient \$01 \$0d \$03 \$0e \$04 \$02 \$0b \$00 fits into the whole chessboard. The dark areas are just faded from \$06 to \$00 with the remaining 2 colors. The FLI bug can be successfully colored black by placing a `ldy #$xx` (opcode \$a0) after the `sta $d011`.



The chessboard-texture to be rasterized:



Timing is nifty in the inner loop to avoid gray dots, same goes for timing when entering the loop.

```

;...
;code for stable irq
;...

ldy ztab
;takes 3 instead of 2 cycles a ldx #$00 would need. This fixes
timing as only a single cycle
;needs to be wasted what can't be done with a single mnemonic
ldx zero
loop0
;now it is time to switch colors in the offscreen area to avoid
gray dots
sta $d022
stx $d021

lda mytab,y
sta $d018
.x lda tab11
;possibility to blank first frames, to avoid glitches
first ora #$40
sta $d011

;-> flibug col = 0
col3 ldy #$00
sty $d023
;-> first line is black, always, else unwanted colors appear in
first line

```

```
        ;then $d023 is $06 for all other lines
        lda #$06
        sta col3+1

        nop
        nop
        ldy #$ff
        ldx .x+1
        bit $ea

        ;clear ztab so that update finds a free table when being called,
        saves cycles outside of irq
        sty ztab,x
        inx
        ;store index for d011-tab as x will be destroyed in 3, 2, 1, ...
        stx .x+1
        ;compare beforehand, carry will stay untouched until branch
        cpx #100

        ldy ztab,x
        ;preload values for $d021/22 and waste some cycles for perfect
        timing (destroys x)
        ldx tabd021,y
        nop
        nop
        nop
        nop
        lda tabd022,y
        bcc loop0
```

Yet no full code-examples here, but as you now know about the ideas behind, it should be possible for you to write your own routines for that, right? 😊

From:
<https://codebase64.org/> - **Codebase 64 wiki**

Permanent link:
https://codebase64.org/doku.php?id=base:twisters_x-rotators_and_waving_carpets

Last update: **2015-04-17 04:34**

