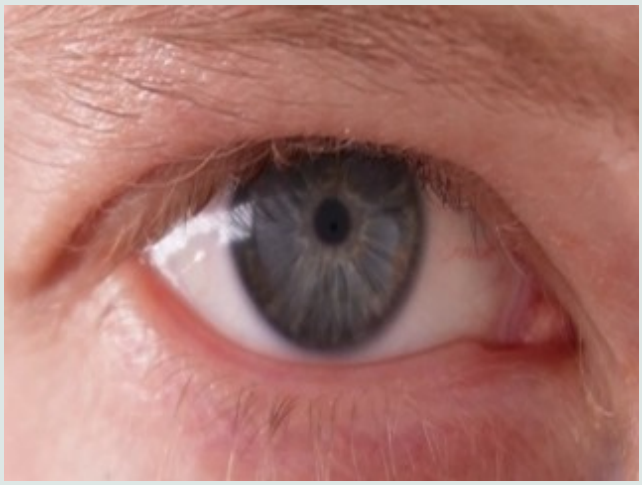
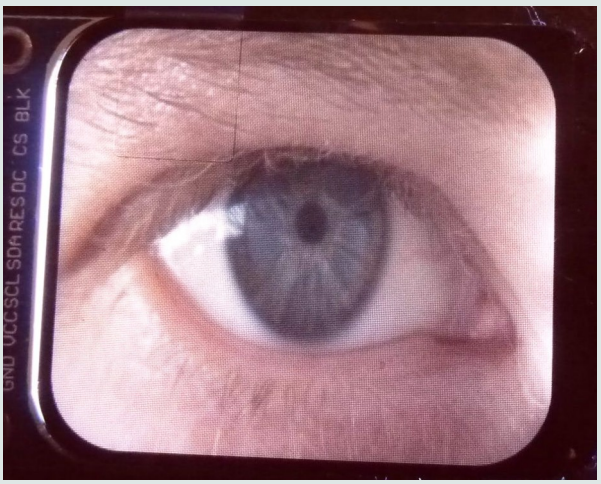
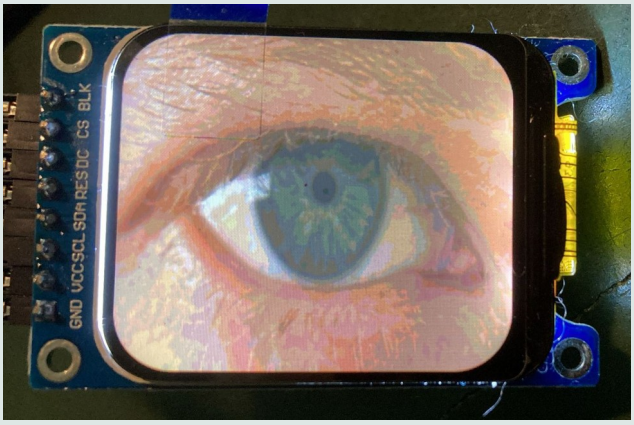
Filmpjes voor de vogel robot (Compressie)



PNG origineel, 176Kb



5-6-5 code op LCD, 144Kb



3-4-3 code op LCD, 42Kb

3-3-2 code op LCD, 27,5Kb

We moeten niet teveel op het kleurverschil letten, maar meer op de details en de vlakken waar de kleur geleidelijk verloopt.

Waar beginnen we:

Het 240x280 LCD heeft een ST7789 driver chip. De hardware lijntjes en SPI-driver zijn geen probleem. Genoeg voorbeelden te vinden in de Egel en de RP2040 voorbeeld files.
De chip activeren is de eerste stap. In het ST7789 datasheet staat geen voorbeeld voor de initialisatie van de chip, na flink online zoeken vond ik een C-library met initialisatie er in.

En wel deze:

```
// Format: cmd length (including cmd byte),
// post delay in units of 5 ms, then cmd payload
// Note the delays have been shortened a little
static const uint8_t st7789_init_seq[] = {
  1, 20, 0x01,         // Software reset
  1, 10, 0x11,         // Exit sleep mode
  2, 2, 0x3a, 0x55,    // Set colour mode to 16 bit
  2, 0, 0x36, 0x00,    // Set MADCTL: row then column, refresh is bottom to top
// CASET: column addresses
  5, 0, 0x2a, 0x00, 0x00, SCREEN_WIDTH >> 8, SCREEN_WIDTH & 0xff,
// RASET: row addresses
  5, 0, 0x2b, 0x00, 0x00, SCREEN_HEIGHT >> 8, SCREEN_HEIGHT & 0xff,
  1, 2, 0x21,          // Inversion on, then 10 ms delay (supposedly a hack?)
  1, 2, 0x13,          // Normal display on, then 10 ms delay
  1, 2, 0x29,          // Main screen turn on, then wait 500 ms
  0                    // Terminate list
};
```

In noForth wordt dat:

```
: LCD-INIT      ( -- )      \ LCD to default mode
    tft-init                \ Hardware I/O initialisation
    reset       dm 150 ms   \ Hardware LCD reset
    01 cmd      dm 150 ms   \ Software LCD reset
    11 cmd      dm 150 ms   \ Exit sleep mode
    55 3A cmd1  dm 010 ms   \ 16-bit color mode
    00 36 cmd1              \ Refresh is bottom to top
    fullscreen              \ F0 00 00 00  2A  Column parameters
                            \ 2C 01 07 00  2B  Row parameters
    21 cmd      dm 10 ms    \ Inverse screen display
    13 cmd      dm 10 ms    \ Normal display on
    29 cmd      dm 150 ms ; \ Main screen on
```

Ik gebruik alleen het volledige scherm:

```
: FULLSCREEN    ( -- )
    2A {cmd  0. 2>lcd   F0. 2>lcd  spi}   \ 00 240
    2B {cmd  7. 2>lcd  2C 1 2>lcd  spi} ; \ 07 300
```

Het scherm schoonmaken:

```
: &FILL     ( c +n -- ) fullscreen  >r  b-b  r>  2C {cmd  fill) spi} ;
: &PAGE     ( -- )      0  dm 72000 &fill ;
```

De data voor een pixel:

Dat is een 16-bits woord met RGB-data, er passen er ongeveer 72.000 op een LCD schermpje.

Een plaatje neemt dan 72.000 * 2 = 144.000 bytes in beslag.
Nu wil ik de filmpjes (serie plaatjes) opslaan in het Flash van de RP2040. Dan zou het handig zijn als de plaatjes wat kleiner zijn.

Omdat we alles zelf maken, kan een (de)compressie algoritme daar ook nog wel bij. Er zijn talloze voorbeelden van de te vinden, maar de meesten zijn nogal complex en drukken zwaar op de CPU en het geheugen.

De eerste stap:

We beperken het aantal kleuren, Albert Nijhof kwam met een 8-bits voorbeeld (256 kleuren) 3-3-2 en wel zo:

De byte erna bevat het aantal opvolgende pixels met dezelfde kleur. Deze vorm van compressie heet run-length codering (het verkort noteren van herhalende tinten). Samen levert dat een flinke winst op: 27.5k. Dat is een compressie factor 5. Het geeft wel een mooi effect, maar er verdwijnen wel erg veel tinten.

Omdat de herhalende patronen zelden langer zijn dan 63 pixels, kies ik de volgende 10-bits code (1024 kleuren) 3-4-3.

De compressie wordt wat minder omdat er meer detaillering in het beeld is door het grotere aantal af te beelden tinten. Het voorbeeld plaatje wordt dan 41.9k, dat is een compressie factor 3.5.

Een gecomprimeerd filmpje maken:

1. Een film splitsen tot PNG plaatjes van 240 x 300 pixels of een veelvoud daarvan
2. PNG plaatjes converteren naar 5-6-5 codering (240 x 300)
3. De 5-6-5 code plaatjes bewerken naar 3-4-3 met run-lengte compressie. Zolang de pixels identiek zijn wordt de run-lengte **R** verhoogd
4. De reeks plaatjes wegschrijven met ervoor het aantal pixels in run-lengte code, zodat het een afspeelbaar filmpje wordt
5. Film data inlezen in noForth en opslaan in het Flash geheugen

De codering van een 3-4-3 pixel in WIN32Forth:

De 5-6-5 code wordt omgezet naar 3-4-3 met 6-bits run-lengte.

```
: ROOD5>3   ( 31 -- 7 )     2/ 2/ ;     \ van max 31 naar max 7
: GROEN6>4  ( 63 -- 15 )    2/ 2/ ;     \ van max 63 naar max 15
: BLAUW5>3  ( 31 -- 7 )     2/ 2/ ;     \ van max 31 naar max 7
: COMPR ( 16b -- 16b )
    dup #11 rshift $1F and rood5>3   #13 lshift
    over  5 rshift $3F and groen6>4    9 lshift or
    swap           $1F and blauw5>3    6 lshift or ;
```

De run-lengte codering gaat zo:

```
#144000 0 ?do
    buffer i + w@ compr      \ Lees volgende 5-6-5 pixel
    px over <> if            \ Anders dan vorige?
        rep!  dup to px  fh, \ Noteer run-lengte en nieuwe pixel
    else  1 +to rep  then    \ Verhoog run-lengte teller
2 +loop
```

Het afspelen gaat als volgt:

1. Lengte van de film ophalen (aantal plaatjes)
2. Plaatje uitlezen, decoderen en afbeelden
3. Tot alle plaatjes gedaan zijn

De decodering van een 3-4-3 pixel in noForth t:

```
decimal
: C-COMMAS ( -- ) begin  0. bl-word count >number nip nip
                  0= while  c,  repeat drop ;

create ROOD3>5  c-commas 0 5 10 15 19 23 27 31 |
create GROEN4>6 c-commas 0 5 10 15 19 23 27 31 35 39 43 47 51 55 59 63 |
create BLAUW3>5 c-commas 0 5 10 15 19 23 27 31 |
:  DECOMPR ( 16b -- color +n )
    >r r@ 13 rshift  7 and rood3>5  + c@ 11 lshift
    r@     9 rshift 15 and groen4>6 + c@  5 lshift  or
    r@     6 rshift  7 and blauw3>5 + c@  or
    r> 63 and ; ( Number of the same pixels )
```

De drie tabellen worden door **DECOMPR** gebruikt voor de conversie van 3-4-3 naar de 5-6-5 LCD-code en de run-lengte (zelfde pixels).

```
code DECOMPR ( 16b -- color +n ) \ W = pointer to tables (35 cycles)
    ( 0 )  c-commas 0 5 10 15 19 23 27 31 |
    ( 8 )  c-commas 0 5 10 15 19 23 27 31 35 39 43 47 51 55 59 63 |
    ( 24 ) c-commas 0 5 10 15 19 23 27 31 |
code>  \ Convert RED, GREEN & BLUE
    day tos dm 13 lsrs.mv,  \ 1 Get 3-bit red color to DAY (7)
    sun 7 # movs,           \ 1 Mask other bits
    day sun ands,           \ 1
    day w adds,             \ 1 Calc. color table
    day  day ) ldrb,        \ 2 Convert red color
    hop day dm 11 lsls.mv,  \ 1 To correct position
    day tos dm 09 lsrs.mv,  \ 1 Get 4-bit green color to DAY (9)
    sun hx 0F # movs,       \ 1 Mask other bits
    day sun ands,           \ 1
    day w adds,             \ 1 Calc. color table
    day 8 # adds,           \ 1 To green table
    day  day ) ldrb,        \ 2 Convert green color
    moon day dm 5 lsls.mv,  \ 1 To green table
    hop moon orrs,          \ 1 Add to red color
    day tos dm 06 lsrs.mv,  \ 1 Get 3-bit blue color to DAY (8)
    sun 7 # movs,           \ 1 Mask other bits
    day sun ands,           \ 1
    day w adds,             \ 1 Calc. color table
    day dm 24 # adds,       \ 1 To blue table
    day  day ) ldrb,        \ 2 Convert blue color
    hop day orrs,           \ 1 Add to other colors
    hop  sp -) str,         \ 3 Result on the stack (11)
    hop hx 3F movs,         \ 1 Leave +n (number of the same bits)
    tos hop ands,           \ 1
    next,                   \ 6
end-code
```

```
: .ROW      ( color +n -- ) \ Print pixel row
    for  dup b-b 2>lcd  next  drop ;


100 value SPEED
: .FRAME    ( a1 -- a2 )    \ Print next picture frame
    fullscreen  hx 2C {cmd
    h@+ 2/ 0 ?do  h@+ decompr .row  loop  spi}  speed ms ;


: FORW      ( a -- )  h@+ 0 ?do  .frame  loop  drop ; \ Play movie ‘a’
```

Een filmpje ziet er na compressie zo uit:

```
\ oog-open.f
frames
0005
A3F2
F98C F97D F77F F748 F944 F742 F943 F755
F541 F742 F541 F741 D741 F74B F541 D544
D502 D541 F746 D742 F749 F941 F984 F741
D541 D741 D541 D502 D541 F743 D543 F541
F742 D741 F741 D741 F984 F941 F989 F97C
F77F F744 F942 F741 F945 F75D D541 D741
F74B F541 F742 D542 D502 D541 F746 D742
F74A F984 F741 D541 D741 D541 D503 F743
F541 D543 F541 F744 F982 F948 F983 F97D
F75D F541 F749 D543 F743 F542 F741 D742
F751 F947 F751 F941 F748 D741 F741 D543
D741 F748 D742 F746 D543 F746 D741 F74B
F981 FB82 F981 F741 D541 F541 D541 D503
F541 F741 F541 D544 F541 D541 F742 D741
F982 F947 F984 F97D F75D D547 F541 F742
D543 F545 D541 D742 F753 F945 F746 F942
F748 F942 F749 D741 D543 D741 F748 D541
D741 F746 D741 D541 F753 F941 FB82 F941
D542 F541 D541 D501 D541 F542 F741 D544
F542 D541 F742 D741 F949 F984 F97D F758
F545 D543 D504 F542 D506 D544 D742 F753
F943 F741 F944 F742 F944 F753 D543 F748
D741 D541 F74D F941 F74E F941 F981 FB81
F941 D543 D502 F541 F743 D541 D501 D546
F741 D541 F982 F947 F984 F97B F75A F546
Etc.

END
```

Er valt altijd te sleutelen:

A) Bij het coderen kan afronding toegepast worden
B) De vertaaltabellen kunnen anders ingericht
C) Het kleurverschil verkort coderen
D) Veel voorkomende pixels verkort afbeelden
E) Kunnen jullie nog wat verzinnen?

Tenslotte resten er nog een paar voorbeelden:

- `oog-links  try`
- `oog-rechts decom`
- `oog-dicht  dm test`
- `oog-dicht1 forw  oog-dicht1 backw  many`
- `oog1`