# Programmering i maskinkode på *AMIGA*

**A.Forness & N.A.Holten**


## Hæfte 12

**Indhold**



**DATASKOLEN**
Postboks 62
Nordengen 18
2980 Kokkedal

Telefon 49 18 00 77

Postgiro 7 24 23 44

# INTRODUKTION

Du er nu nået til BREV 12 - det sidste brev i kurset. På de følgende sider vil vi beskæftige os med en del **TIPS & TRICKS.** Du vil få at se, hvordan du kan udnytte en del af det du har lært. Vi gennemgår ikke programmerne instruktion for instruktion, men forklarer kun hvordan de enkelte del-rutiner fungerer.

Når du er færdig med dette brev, er du som nævnt færdig med hele kurset. Selvfølgelig findes der meget mere litteratur om AMIGA, som kan være nyttigt at have. Den berømte "hardware manual" fra Commodore er den man ofte anskaffer sig først. Denne bog indeholder alt om det hardware-mæssige om AMIGA, som f.eks alle HARDWARE-registrene (som begynder med: $DFF-- -),, BITPLANES, BLITTER, DISK, serieport, parallelport osv. Læg mærke til at bogen ikke er en lærebog, men en referencebog (opslagsbog) for AMIGA. Endvidere kan du skaffe hele serien med referencebøger fra Commodore. Der er udgivet fire bøger indtil videre:

1. **HARDWARE REFERENCE MANUAL**

2. **INTUITION REFERENCE MANUAL**

3. **EXEC REFERENCE MANUAL**

4. **LIBRARIES & DEVICES REFERENCE MANUAL**

Med disse bøger er du sikret information nok til at løse alle dine programmeringsproblemer - men husk at de er referencebøger (opslagsværker).

Der findes vældig mange lærebøger, som omhandler AMIGA. På dette felt er det vanskeligt for os at komme med nogle anbefalinger, fordi der hele tiden udkommer nye bøger om emnet. Egentlig tror vi ikke at du har behov for flere bøger end dette kursus plus de fire bøger, som er nævnt ovenfor.

# HAM, HIRES og INTERLACE

I dette kapitel vil vi se nærmere på følgende programeksempler (som findes på DISK 2):

**HAM**
**HIRES**
**INTERLACE**

Disse programeksempler har det fælles, at de sætter et skærmbillede op. Lad os først fordybe os i HAM-eksemplet.

**HAM-eksemplet findes på DISK 2 i DIRECTORY "HAM".** Der findes følgende filer:

| | | |
|---|---|---|
| **HAM.S** | = | **SOURCE-koden.** |
| **HAM** | = | **En færdig-assemblet version** |
| **SCREEN** | = | **Filen som indeholder billedet** |

Billedet er 320 * 256 PIXELS. Det har 6 BITPLANES og benytter de første 16 farveregistre. Husk at et HAM-billede næsten altid har 6 BITPLANES (det er muligt at bruge 5). Som bekendt kan et HAM-billede vise 4096 (alle) farver. Lad os se lidt nærmere på hvordan dette fungerer.

Forkortelsen HAM står for. **HOLD AND MODIFY.** Dette kan oversættes med: BEHOLD OG JUSTER. Lad os tage et eksempel:

Forestil dig at første PIXEL (øverst oppe i venstre hjørne) på skærmen er rød. I AMIGA's paletsystem kan der sættes en værdi fra 0 til 15 for hver grundfarve. En rød PIXEL har værdierne: RØD = 15, GRØN = 0 og BLÅ = 0. I HAM-modus vil næste PIXEL (den til højre for den første) og dens farve ofte være afhængig af den forrige PIXEL. Det er altså muligt at sætte PIXEL nummer 2 til orange uden at farveregistrene bliver benyttet. For at lave overgangen fra rød til orange, gøres følgende:

AMIGA lægger farvedataene for den røde PIXEL i et bufferregister. Derefter bliver kun farve-værdien for grøn forandret til f.eks 8. Farveværdierne for rød og blå forbliver uforandrede. Vi har nu fået en ny farvekombination, som ser således ud: RØD = 15, GRØN = 8 og BLÅ = 0. Lad os se hvordan de 6 BITPLANES bliver udnyttet:

BITPLANE 1-4 = Farvestyrke (0-15) eller farveregister.

BITPLANE 5-6 = Justeringsvalg.

**VALGKOMBINATIONER**

| BIT 5 | BIT 6 | | |
|---|---|---|---|
| 0 | 0 | = | Farven på denne PIXEL bliver hentet direkte fra et farveregister, som er angivet i BITPLANE 1-4. Dette er det samme, som sker i f.eks almindelig LORES. |
| 0 | 1 | = | Her bliver farven fra forrige PIXEL brugt igen, undtagen den BLÅ-værdi, som bliver hentet fra BITPLANEs 1-4. |
| 1 | 0 | = | Også her bruges farven fra sidste PIXEL, og derefter bliver RØD-værdien for farven ændret til den værdi, som BITPLANES 1-4 indeholder. |
| 1 | 1 | = | Dette vil virke akkurat som ovenfor, med den forskel at det er GRØN-styrken i farven, som bliver forandret. |

På denne måde kan AMIGAen vise 4096 farver på en gang. Ulempen ved HAM-modus er, at du kun kan ændre en af de tre grundfarver ad gangen. Du kan altså ikke sætte en rød PIXEL lige ved siden af en grøn. Det er derfor alle HAM-billeder har "slørede" farveovergange.

For at starte HAM-eksemplet, skal du selvfølgelig assemble det først. Derefter indlæser du "SCREEN"-filen, som ligger i samme DIRECTORY. Program-eksemplet behøver ikke nogen nærmere forklaring, da det meget ligner andre eksempler, som opsætter en skærm. Det er vigtigere, at du forstår virkemåden for HAM-modus.

Lad os nu gå direkte over til det næste program-eksempel, som hedder **HIRES.** Forkortelsen står for: **HIGH RESOLUTION.** Eller på dansk: HØJ OPLØSNING. På DISK 2 i **DIRECTORY "HIRES"** findes følgende filer:

| | | |
|---|---|---|
| **HIRES.S** | = | **SOURCE-koden.** |
| **HIRES** | = | **Et kørbart eksempel** |
| **SCREEN** | = | **Billedet** |

Forskellen mellem LORES og HIRES er, at du i den sidstnævnte opløsning får dobbelt så mange PIXELs pr linie. Standardstørrelsen på en LORES-skærm er som bekendt 320 * 256. I HIRES er denne størrelse altså 640 * 256.

Et eksempel på en hires-skærm er WorkBench-skærmen. Den er HIRES, 640 * 256, 2 BITPLANES (eller 4 farver). Begrænsningen for HIRES-modus i forhold til LORES, er at du ikke kan have mere end 16 farver på skærmen (4 BITPLANES), og at du ikke kan benytte HAM-modus. Vi kan også nævne at spil og demo'er sjældent benytter HIRES.

For at sætte HIRES-programmet igang, skal du indlæse filen "SCREEN" (som indeholder billedet). Den ligger i samme DIRECTORY. Læg mærke til at både HAM-billedet ovenfor og dette HIRES-billede er digitaliseret med videokamera (de er absolut ikke tegnet med hånden). Nærmere forklaring af HIRES-programmet skulle være unødvendigt.

Nu er vi kommet til sidste emne i dette kapitel, nemlig **INTERLACE.** Dette program-eksempel finder du i **DIRECTORY "INTERLACE"**, og det indeholder disse filer:

| | | |
|---|---|---|
| **LACE.S** | = | **SOURCE-koden.** |
| **LACE** | = | **En kørbar version.** |
| **SCREEN.IFF** | = | **Billedet i IFF-format.** |
| **SCREEN** | = | **Billedet i "rå-format".** |

INTERLACE-modus gør at du kan fordoble opløsningen vertikalt. Med andre ord: Dobbelt så mange linier på skærmen.

I HIRES + INTERLACE har du en standardopløsning på 640 * 512 PIXELs. Du har sikkert set, at billedet "flimrer" når AMIGAen er i INTERLACE-modus. Det skyldes, at billedet på skærmen kun opdateres 25 gange i sekundet. Almindeligvis opdateres 50 gange i sekundet. Hvorfor tegner AMIGA så kun billedet op 25 gange i sekundet i interlace?

Egentlig tegnes billedet 50 gange, men der tegnes 2 forskellige billeder op hver anden gang. Er det en smule forvirrende? Lad os prøve at uddybe det:

Først tegnes linie 1 af billedet, derefter tegnes linie 3, så nummer 5, og således fortsætter det med hveranden skærmlinie helt ned til linie 355. Derefter starter AMIGA fra toppen igen, og tegner linie 2, linie 4, linie 6 osv. Dette resulterer i, at du ser dobbelt så mange linier på skærmen. INTERLACE-modus kan benyttes i kombination med HAM eller HIRES.

Dette program-eksempel opsætter en INTERLACE + HIRES-skærm - som altså er 640 * 512 PIXELs stor. Billedet har et (1) BITPLANE (2 farver).

For at starte programmet, skal du først indlæse filen "SCREEN", som findes i samme DIRECTORY.

# WAVE

"WAVE" er en effekt som ofte benyttes i demo'er. **Det er en bølge-effekt, som får billedet på skærmen til at "bugte sig" i horisontal retning.** I AMIGA findes et hardwareregister, som kaldes SCROLL-registret. Dette register har adresse $DFF102. Opsætningen for dette register ser således ud:

**SCROLL $DFF102 (READ ONLY)**

| **BIT Nr.** | **Funktion** |
|---|---|
| 0-3 | SCROLL-værdi for BITPLANE 1,3 og 5 |
| 4-7 | SCROLL-værdi for BITPLANE 2,4 og 6 |
| 8-15 | Ikke benyttet, sættes til 0. |

Med dette register kan du altså forskyde skærmbilledet 0-15 PIXELs. For at få bølge-effekten bruges COPPERen til at sætte en ny værdi i registret for hver linie. Vi kan altså forskyde de enkelte linier uafhængig af hinanden.

I dette program-eksempel har vi lavet en lille SINUS-tabel, som giver en "naturlig" bølgebevægelse. Programmet opsætter en LORES-skærm med 320 * 256 PIXELs opløsing og 2 BITPLANEs. Bølge-effekten laver vi i området 30 linier fra toppen på skærmen og ned til linie 230. Dataene i SINUS-tabellen bliver kontinuerligt madet ind i COPPERen for at give en konstant bølgebevægelse.

Læg specielt mærke til, at en del af COPPER-listen bliver genereret af selve programmet.

Hvis du vil prøve dette program-eksempel, finder du det i **DIRECTORY "WAVE"** på Kursusdisketten DISK 2. Der findes følgende filer der:

| | | |
|---|---|---|
| **WAVE.S** | = | **SOURCE-koden.** |
| **WAVE** | = | **En kørbar version.** |
| **SCREEN.IFF** | = | **Billedet lagret i IFF-format.** |
| **SCREEN** | = | **Billedet i "rå-format".** |
| **SIN** | = | **SINUS-tabellen.** |

For at starte programmet, skal du som sædvanlig assemble det. Derefter indlæser du "SCREEN"-filen og "SIN"-filen, som ligger i samme DIRECTORY. Du kan jo prøve at ændre bølge-hastigheden. Dette gøres ved at ændre tallet på programlinie 68. God fornøjelse!

# ROTATION AF BILLEDE

**At få et billede til at rotere** synes for mange som en "kedelig" opgave. Egentlig er det ikke så komliceret. I og med at skærmen er flad (2 dimensioner) er det kun et spørgsmål om en sammentrykning. Hvis du trykker billedet sammen og trækker det ud i en SINUS- (cirkel-) bevægelse rundt midt på skærmen, vil det se ud som om man roterer billedet.

Dette kan gøres med COPPERen. Det man egentlig gør når man sammentrykker billedet, er at man springer enkelte linier over. Hvis man hopper over hver anden linie vil billedet kun blive halvt så højt. Du begynder altså med alle linier synlige og billedet oprejst. Derefter springer du over f.eks 1 linie pr. 100 linier. Når du øger dette i en jævn SINUS bevægelse, vil det se ud som om det roterer.

Program-eksemplet ligger på kursusdiskette DISK 2 i **DIRECTORY "ROTATE"** og det indeholder disse filer:

| | | |
|---|---|---|
| **ROT.S** | = | **SOURCE-koden.** |
| **ROT** | = | **Den kørbare fil.** |
| **SCREEN.IFF** | = | **Billedet i IFF.format.** |
| **SCREEN** | = | **Billedet i "rå-format"** |
| **SIN** | = | **SINUS-tabellen.** |

For at kunne køre dette program-eksempel, skal filerne "SCREEN" og "SIN" indlæses først. Hvis du vil ændre på rotations-hastigheden, ændrer du tallet på programlinie 15.

For en god ordens skyld nævner vi, at billedet er LORES, 320 * 256, 3 BITPLANES eller 8 farver.

# DEMO

I dette kapitel ser vi lidt nærmere på, hvordan man sætter en DEMO op. På DISK 2 i **DIRECTORY "DEMO"** ligger disse filer:

| | | |
|---|---|---|
| **DEMO1.S** | = | **SOURCE-kode. Opsætter et HAM-billede.** |
| **DEMO2.S** | = | **SOURCE-kode. Lægger en ekstra skærm til.** |
| **DEMO3.S** | = | **SOURCE-kode. Inkluderer SCROLLen.** |
| **DEMO4.S** | = | **SOURCE-koden. Den færdige DEMO med SPRITEs.** |
| **DEMO4** | = | **En kørbar version af den færdige DEMO.** |
| **SCREEN.IFF** | = | **HAM-billedet i IFF-format.** |
| **/SCREEN** | = | **Billedet i "råformat".** |
| **FONT.IFF** | = | **Fonten i IFF.format.** |
| **FONT** | = | **Fonten færdig-konverteret.** |
| **SPR.IFF** | = | **SPRITEne i IFF-format.** |
| **SPR** | = | **SPRITEne i færdig "rå-format".** |
| **MOVETAB=** | | **En tabel, som indeholder bevægelsesdata for SPRITEne.** |

Programeksemplerne DEMO1, DEMO2, DEMO3 og DEMO4 viser skridt for skridt, hvordan man lægger nye effekter til i en DEMO.

# DEMO1

Her begynder vi med at opsætte et HAM-billede, som er 320 * 230 PIXELs stort. Vi har sløjfet noget af højden, således at vi får plads til en **SCROLL-tekst nederst på skærmen.**

**Linie 1-2:** Slukker INTERRUPTS, BITPLANE-DMA, COPPER-DMA OG SPRITE-DMA.

**Linie 4-10:** Lægger 16 farver ind i COPPER-listen. Vi lader farverne blive opdateret af COPPERen, fordi vi skal ændre nogle af farverne længere nede på skærmen.

**Linie 12-23:** Lægger adresserne på de enkelte BITPLANEs ind i COPPERen.

**Linie 25-28:** Sætter adressen på COPPER-listen og starter BITPLANE-DMA og COPPER-DMA.

**Linie 31-39:** Venter til du trykker på mus-knappen. Derefter hentes WorkBench-skærmen tilbage og programmet afsluttes.

**Linie 42-91:** Her ligger vores COPPER-liste deklareret.

**Linie 94:** Deklarerer skærm-hukommelsen.

# DEMO2

**Næste skridt er at opsætte en ekstra skærm nederst.** Det er her SCROLL-teksten skal flyde. Denne skærmbid er 320 * 32 PIXELs i et BITPLANE, og har 4 BYTEs i MODULO. Vi forklarer kun de nye rutiner:

**Linie 25-30:** Her lægges adressen på den nye skærm ind i COPPER-listen.

**Linie 99-112:** Dette har vi lagt til COPPER-listen for at programmet skal vise den nye skærm. Vi ændrer her MODULO til 4 og angiver 1 BITPLANEs LORES-skærm.

**Linie 115:** Her har vi skaffet plads til den nye skærm. Størrelsen er altså; 44 * 32 = 1408 BYTEs.

# DEMO3

**Vi fortsætter med at lægge en FONT ind og sætter SCROLL-teksten i bevægelse.** Her følger en forklaring til de nye programlinier:

**Linie 38-42:** Venter på at elektronstrålen er nået til linie 100.

**Linie 44:** Hopper til SCROLL-rutinen som opdaterer SCROLL-teksten.

**Linie 56-122:** Her ligger rutinen som bevæger SCROLL-teksten. En sådan rutine skulle være kendt stof.

**Linie 192-250:** Deklarerer konverterings-tabellen for SCROLL-rutinen. Læg mærke til at vi bruger både X- og Y-position, for at finde ud af, hvor tegnet i FONTen befinder sig.

**Linie 253:** Her har vi deklareret plads til FONTen - som i dette program er et billede på 320 * 128 PIXELs. Et tegn er 24 * 32 PIXELs, og der ligger 13 tegn på en linie. Vi har altså 13 * 4 = 52 forskellige bogstaver og tegn i FONTen.

**Linie 262:** Her ligger teksten deklareret. De lovlige tegn er:

**A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 ! ( ) . , : ; = + / - * < > '**

**Læg mærke til at teksten skal slutte med 255.**

# DEMO4

Her lægger vi sidste hånd på værket ved at lave en **"SPRITE-SNAKE"-rutine.** Denne består af 8 SPRITES. Her følger forklaringen:

**Linie 32-43:** Her lægges adresserne på SPRITEne ind i COPPER-listen.

**Linie 59-64:** Denne rutine venter til elektronstrålen er nået til linie 280.

**Linie 66:** Hopper til rutinen som flytter SPRITEne på skærmen.

**Linie 76-129:** Her ligger rutinen som bevæger "SPRITESNAKEN" på skærmen.

**Linie 200-216:** Her har vi lagt SPRITE-pointerne til i COPPER-listen.

**Linie 229-240:** Her har vi deklareret SPRITE-farverne.

**Linie 359:** Deklarerer plads til "MOVETAB" - som indeholder bevægelses-data til SPRITEne.

**Linie 362:** Her har vi sat plads af til vores SPRITE-data. Hver SPRITE er 16 PIXELs høj og bruger 72 BYTEs.

Vi har nu lavet en enkel DEMO. For at køre dette program (DEMO4), skal du indlæse disse filer: SCREEN, FONT, SPR og MOVETAB. Hvis du vil lave din egen FONT, skal du gøre følgende:

Start et tegneprogram op (f.eks DeluxePaint) i LORES 320 * 256 med et BITPLANE (2 farver). Første bogstav skal være øverst oppe i venstre hjørne, og størrelsen på et bogstav må være 24 PIXELs bred og 32 PIXELs høj. Du får da plads til 13 tegn pr. linie. Næste linie begynder 32 PIXELs nede på skærmen, og det kan være op til 4 linier. Her kommer placeringen af tegnene:

```
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 ! ( )
. , : ; = + / - * < > '
```

Når du har tegnet FONTen færdig, skal den konverteres. Start IFF.konverteren op og hent FONTen ind. Vælg "No cmap", og derefter lagres FONTen med "frame+save". Du skal klippe fra position (0,0) til (320,128). FONTen er nu klar til brug. Held og Lykke!

# DISK

I dette kapitel skal vi se lidt nærmere på, hvordan man "håndterer" disketter. I **DIRECTORY "DISK"** finder du følgende fil:

**DISK.S** = **SOURCE-kode.**

Dette program er inddelt i forskellige rutiner:

**DRIVEON**
**DRIVEOFF**
**WRITEDISK**
**READDISK**
**TRACK**
**TRACK00**
**CODEMFM**
**DECODEMFM**

Med disse rutiner kan du lave en såkaldt "trackloader". Rutinerne **DRIVEON** og **DRIVEOFF** bruges til at starte og stoppe et angivet drive.

**WRITEDISK**-rutinen skriver MFM-data til disk. MFM-data er det data-format, som bl.a AMIGAs disk-system benytter. Hvis du f.eks har 1000 BYTEs med data, som skal lagres på disk, skal disse først laves om til MFM.format. Længden på MFM.dataene bliver det dobbelte af de sædvanlige data, nemlig 2000 BYTEs. Rutinen **READDISK** bruges til at indlæse MFM-data fra disken igen.

**TRACK-rutinen** benyttes til at flytte læse/skrivehovederne til et bestemt spor, mens TRACK00-rutinen skal udføres i begyndelsen af programmet for at nulstille læse/skrivehovederne. På denne måde giver vi maskinen besked på, hvor spor 0 (nul) er. Denne position bliver så brugt som referencepunkt for læse/skrivehovederne, således at de øvrige spor kan findes.

Rutinen **CODEMFM** benyttes for at lave almindelige data om til MFM-format, mens **DECODEMFM**-rutinen gør det modsatte - altså laver MFM-data om til almindelige data.

AMIGAs diskettestation har 80 spor (TRACKs) på hver side, tilsammen 160 spor. Et sådant spor kan teoretisk set lagre 12500 BYTEs MFM-data eller 6250 BYTEs almindelige (egentlige) data. I praksis kan du lagre 6000 BYTEs almindelige data på et spor. Den total lagringskapacitet bliver da 6000 * 160 = 960.000 BYTEs = 937,5 kB.

Fremgangsmåden for at skrive data til et spor på en disk er: Først startes diskettestationen med rutinen DRIVEON. Du angiver i register D0, hvilket drive du vil bruge:

```
moveq #?,D0 D0 = 0-3 (0=df0:, osv.)
bsr   driveon
```

Derefter hopper du til rutinen TRACK00. Denne rutine behøver intet parameter.

```
bsr    track00
```

Nu lægger du dine data ind i "diskbuf". Der kan højst lægges 6000 BYTEs ind i bufferen.

Derefter hopper du til rutinen CODEMFM. Denne rutine vil lave dataene i "diskbuf" om til MFM-format. Denne rutine behøver heller ikke noget parameter.

```
bsr    codemfm
```

Nu vælger du det spor, som dataene skal skrive til. Dette gøres med TRACK-rutinen. Du angiver i D0 hvilket spor du vil flytte til.

```
moveq  #?,D0 D0 = 0-159 (spor nr.)
bsr    track
```

Til sidst startes selve skrivningen med rutinen WRITEDISK.

```
bsr   writedisk
```

Hvis du skal skrive til flere spor på samme tid, bruger du rutinerne CODEMFM, TRACK og WRITEDISK. Du behøver altså ikke hoppe til DRIVEON eller TRACK00. Hvis du derimod er færdig med skrivningen, stopper du diskettestationen med DRIVEOFF. Du angiver i D0, hvilket drive som skal stoppes:

```
moveq  #?,D0 D0 = 0-3 (0=dfo:, osv)
bsr    driveoff
```

Hvis du vil læse et spor fra disk, er fremgangsmåden således:

Start diskettestationen op med DRIVEON på samme måde som under skrivning til disk.

Hop til TRACK00 for at nulstille hovederne.

Angiv hvilket spor, som skal læses i D0 og hop til TRACK-rutinen.

Hop til READDISK-rutinen for at få indlæst MFM-dataene.

Derefter laves MFM-dataene om med DECODEMFM-rutinen. Denne rutine behøver heller intet parameter i lighed med CODEMFM-rutinen.

Dataene ligger nu klar til "afhentning" i "diskbuf". Hvis du skal læse flere spor på samme tid, repeterer du bare rutinerne TRACK, READDISK og DECODEMFM. Når du er færdig med at læse, stopper du drivet med DRIVEOFF.

Du er nu klar til at prøve dig frem med "trackloading". Vi vil bare påpege en meget vigtig ting før du går igang:

**BRUG IKKE DISKETTER, HVORPÅ DER LIGGER AMIGA-PROGRAMMER. HVIS DU BRUGER DISSE RUTINER TIL AT SKRIVE (LÆSE VIRKER UDMÆRKET) TIL EN AMIGA-FORMATERET DISKETTE, HVOR DER ALLEREDE LIGGER PROGRAMMER, VIL DU HØJST SANDSYNLIGT MISTE DINE DATA !**

# STAR1

**Dette kapitel behandler de berømte "stjernehimler"**. Dette program-eksempel animerer 256 "stjerner" (PIXELs). Effekten bliver som om du bevæger dig ud i verdensrummet i en forfærdelig fart. "Stjernerne" strømmer altså ud fra midten af skærmen og ser ud som om de kommer imod dig.

Effekten er ganske enkel at programmere. Du har på forhånd lavet en tabel (STARS) - som er en række vilkårlige tal, der angiver den startposition og udgangsvinkel, hver enkelt PIXEL skal have. Der benyttes også en SINUS/COSINUS-tabel (SIN) for at udregne kursen på PIXELene.

"Stjernerne" skal jo have lidt hastighed inde på midten af skærmen for at se naturlige ud, for derefter at øge ud mod kanten af skærmen. Til dette har vi lavet en "accelerations"-tabel (ACC).

Til sidst har vi en rutine, som stille roterer hele "stjernehoben". Dette gøres ved at dreje udgangsvinklen på alle "stjernerne".

Program-eksemplet ligger på DISK 02 i **DIRECTORY "STAR"**. Det indeholder disse filer:

| | | |
|---|---|---|
| **STAR.S** | = | **SOURCE-koden.** |
| **STAR** | = | **En kørbar version.** |
| **SIN** | = | **SINUS/COSINUS-tabellen.** |
| **ACC** | = | **Accelerations-tabellen.** |
| **STARS** | = | **Positions-tabel.** |

For at starte program-eksemplet skal filerne: SIN, ACC og STARS indlæses først.

Du kan jo som et eksperiment prøve at ændre rotations-hastigheden og retningen. Dette gøres ved at ændre tallet på programlinie 59. Hvis du vil ændre på retningen, kan du bruge "subq.w" i stedet for "addq.w".

Have fun - drive yourself crazy!

# LINEDRAW

Vi skal nu se lidt nærmere på linietegning på AMIGA. På DISK 2 i **DIRECTORY "LINEDRAW"** findes disse filer:

**LINEDRAW.S = SOURCE-kode for linietegning.**
**LINETEST.S = SOURCE-kode for testprogram.**
**LINETEST = En kørbar fil af testprogrammet.**

Lad os begynde med **LINEDRAW-rutinen.** Vi går ikke nærmere ind på, hvordan denne fungerer, men nøjes med at fortælle, hvordan den bruges.

Lad os sige, at du har sat en LORES, 320 * 256 PIXELs og med et BITPLANE skærm op. Bredden på skærmen i BYTEs bliver 40. Du sætter da "swid" på programlinie 1 til værdien 40.

Næste skridt er at hoppe til "initlinedraw"-rutinen, som sætter BLITTERen op og lægger adresser ind i adresse-register A0-A2.

Nu er du klar til at tegne. Du angiver første position på skærmen i D0 og D1 (X og Y), og anden position i D2 og D3 (X og Y). Derefter hopper du til "linedraw", og der bliver trukket en linie mellem de to angivne positioner.

Husk at, hvis du bruger BLITTERen til noget andet end linietegning, skal du hoppe til "initlinedraw"-rutinen før du kan tegne næste linie. Dette gælder også, hvis du har benyttet A0, A1 eller A2 til noget andet. Hastigheden på linierutinen er ca. 94 µS (mikrosekunder) for en "gennemsnitlig lang linie" (mere unøjagtigt end sådan kan det vel ikke siges...).

Den næste program-rutine - som hedder **"LINETEST"** - sætter en skærm op og lader dig tegne linier ved at bevæge musen. Du kan også se i SOURCE-koden for dette program, hvordan man benytter sig af - og kalder - "linedraw"-rutinen. Til sidst skal vi blot nævne, at hvis du prøver at tegne en linie fra og til samme position (punkt), så vil du ikke få noget resultat.

# VECTOR

I dette kapitel skal vi se lidt nærmere på **VECTOR-animation.** Ordet VECTOR betyder egentlig "linie imellem to punkter". På kursusdisketten i directory **"VECTOR"** findes følgende filer:

| | | |
|---|---|---|
| **VEC1.S** | = | **SOURCE-kode for ekempel 1.** |
| **VEC1** | = | **En kørbar fil.** |
| **VEC2.S** | = | **SOURCE-kode for eksempel 2.** |
| **VEC2** | = | **Den kørbare fil til eksempel 2.** |
| **SIN** | = | **SINUS/COSINUS-tabel.** |

Hvis vi skal komme ind på matematikken i disse program-eksempler, ville det fylde en hel bog. Vi koncentrerer os derfor kun om, hvordan man bruger programmerne.

Når du skal definere et objekt, opgiver du positionen for hvert punkt i objektet. Som eksempel har en kube 8 punkter, et i hvert hjørne. Det som gør det hele så interessant er, at du angiver både X-, Y- og Z-position for hvert punkt. Du laver altså tredimensionale (ofte forkortet til 3D) objekter. Når objektet skal vises på skærmen, skal det omregnes til et todimensionalt (2D) billede. For at beholde dybdevirkningen laver programmet også lidt perspektiv.

På programlinie 229 defineres punkterne for objektet. Det har følgende opsætning:

```
vectors:
dc.w   X,Y,Z (punkt 1)
dc.w   X,Y,Z (punkt 2)
...osv
```

Husk at position 0 er i rotationspunktet, sådan at du kan angive både positive og negative værdier. Værdierne kan variere fra -10000 til + 10000. Du kan godt bruge større værdier, men der er da en vis fare for at objektet stikker udenfor skærmen. Når du har defineret alle punkterne, angiver du antal punkter i "pnr" på programlinie 2.

Nu er det din tur til at bestemme mellem hvilke punkter der skal trækkes linier. Dette angives på programlinie 239 således:

Hvis du skal trække linier mellem f.eks punkt 1 til 2, 2 til 3 og 3 til 1, gør du sådan:

```
lines:
dc.w   0,1,1,2,2,0
```

Derefter sætter du antal linier (i dette tilfælde 3) i "lnr" på linie 1 i programmet.

**Program-eksemplet ligger i filen "SIN".** Det er også muligt at forandre rotations-hastigheden i alle tre akser. Det gøres ved at forandre værdierne i programmet på linierne 31, 32 og 33. Disse værdier bestemmer henholdvis X-, Y-, og Z- rotationen.

Du vil sikkert lægge mærke til at du ikke kan definere uendelig komplicerede objekter. Objekter med ikke ret meget mere end 20 punkter og 50 linier kan være vanskelige at få til at fungere rigtigt. Du kan f.eks opleve at du mister noget af objektet af og til - eller at det begynder at flimre.

# DIVERSE

I dette kapitel skal vi beskæftige os med - som overskriften lyder - diverse. På kursusdisketten i **DIRECTORY "DIVERSE"**, findes følgende filer:

**READMOUSE.S**
**GETWB.S**
**INITSCREEN.S**
**SQR.S**

Lad os starte med **READMOUSE**-rutinen. Denne rutine benyttes til at aflæse musens position. Den er næsten selvforklarende, men vi gennemgår lige brugen alligevel.

Når du skal lægge denne rutine ind i dit program, skal du hoppe til rutinen "initmouse" øverst i dit program. Dette gør at mustællerne bliver nulstillet. Derefter hopper du til rutinen for hver skærmopdatering.

Når du vil aflæse positionen, kan du gøre følgende:

```
lea.l   mousexy,a1
move.w (a1),d1        (X-pos)
move.w 2(a1),d2       (Y-pos)
```

Det er også muligt at sætte minimum- og maksimum-værdierne øverst i rutinen (rm_xmin, rm_xmax,rm_ymin,rm_ymax).

Den næste rutine vi skal se på er **GETWB.** Denne rutine henter information om WorkBench-skærmen (adresse, højde osv), og returnerer adressen til denne opsætning i A0:

| **OFFSET** | **LÆNGDE** | **INDHOLD** |
|---|---|---|
| 0 | WORD | Skærmbredde i BYTEs |
| 2 | WORD | Skærmhøjde i PIXELs (linier) |
| 4 | WORD | Antal BITPLANEs |
| 6 | WORD | Ikke benyttet |
| 8 | LONG | Adresse BITPLANE 1 |
| 12 | LONG | Adresse BITPLANE 2 |
| 16 | LONG | Adresse BITPLANE 3 |
| 20 | LONG | Adresse BITPLANE 4 |

**Eksempel:**

```
move.w          4(a0),d0        (D0 = Antal BITPLANEs)
...
move.l          8(a0),d1        (d1 = Adresse på BITPLANE 1)
...
```

Denne rutine skulle være let nok. Det kan være nyttigt at få fat på adressen til WorkBench-skærmen, når du ikke "orker" at opsætte din egen skærm for at teste et eller andet.

Næste programeksempel, som hedder "**INITSCREEN**" opsætter en LORES, 320 * 256 PIXELs skærm i et BITPLANE. Du kan bruge dette program som udgangspunkt når du skal lave/teste noget nyt. Eksemplet behøver ikke nogen speciel forklaring.

Det sidste program i dette kapitel er mere til test end til nytte: **"SQR"-rutinen.** Denne rutine udregner kvadratroden af et tal, som angives i D0. Svaret bliver også returneret i D0. Læg mærke til, at denne rutine kun giver heltal som svar.

# EPILOG

Dette var det sidste brev i vores kursus i, hvordan man programmerer AMIGA maskinkode. Som du sikkert har opdaget, er alt ikke beskrevet, men det er forsøgt fra vores side at få så meget med, at du "finder den røde tråd". Derfra burde det ikke være forbundet med så store vanskeligheder at fortsætte med at dygtiggøre sig på egen hånd.

Vi anbefaler vores kursusdeltagere at skaffe sig de såkaldte "AMIGA-bibler" - som vi nævnte i introduktionen til dette brev. Det er relativt dyre bøger, men du vil klare dig længe med en af disse fire: HARDWARE REFERENCE MANUAL. Det var den vi selv brugte da vi tog vores første vaklende skridt ind i den ukendte "vildmark" som blev kaldt AMIGA. Den burde jo således også være et godt hjælpemiddel for dig.

Skulle nogen have vanskeligheder med at skaffe disse bøger, så kan du kontakte os på 49 18 00 77, og vi vil så prøve at hjælpe dig med at få fat i dem.

Når du nu har gennemgået dette kursus (og kan naturligvis alt som står heri udenad), så er du jo sikkert interesseret i at få et bevis for at du har gennemgået dette kursus. Og som vi nævnte for lang tid siden, kan du selvfølgelig få det. Hvis du skriver til os efter at have gennemgået dette brev, så vil du få tilsendt en lille prøve (uden ekstraomkostninger) hvor du vil få testet dine kundskaber.

Efter at have løst opgaverne i den, returnerer du den til os, og vi vil sende dig et diplom som et "bevis" på dine programmeringskundskaber. Vi har skrevet ordet BEVIS i anførselstegn. Årsagen er den, at vi jo ikke kan kontrollere om det er dig alene - som til en rigtig eksamen - der løser opgaverne. Prøven giver dig derudover også en indikation af, hvor dine stærke/svage sider er.

Hvis du en dag søger et job indenfor databranchen, så er dette diplom og eventuelt en DEMO en fin måde at præsentere dig selv og dine kundskaber på.

Til sidst vil vi benytte anledningen til at takke dig for at have deltaget i dette kursus. Hold dine kundskaber ved lige, og når vi udgiver flere dybdegående/special (brev)kurser indenfor maskinkodeprogrammering, så ses vi måske igen?

Vi ønsker dig held og lykke fremover.

Med venlig hilsen
DATASKOLEN

Carsten Nordenhof