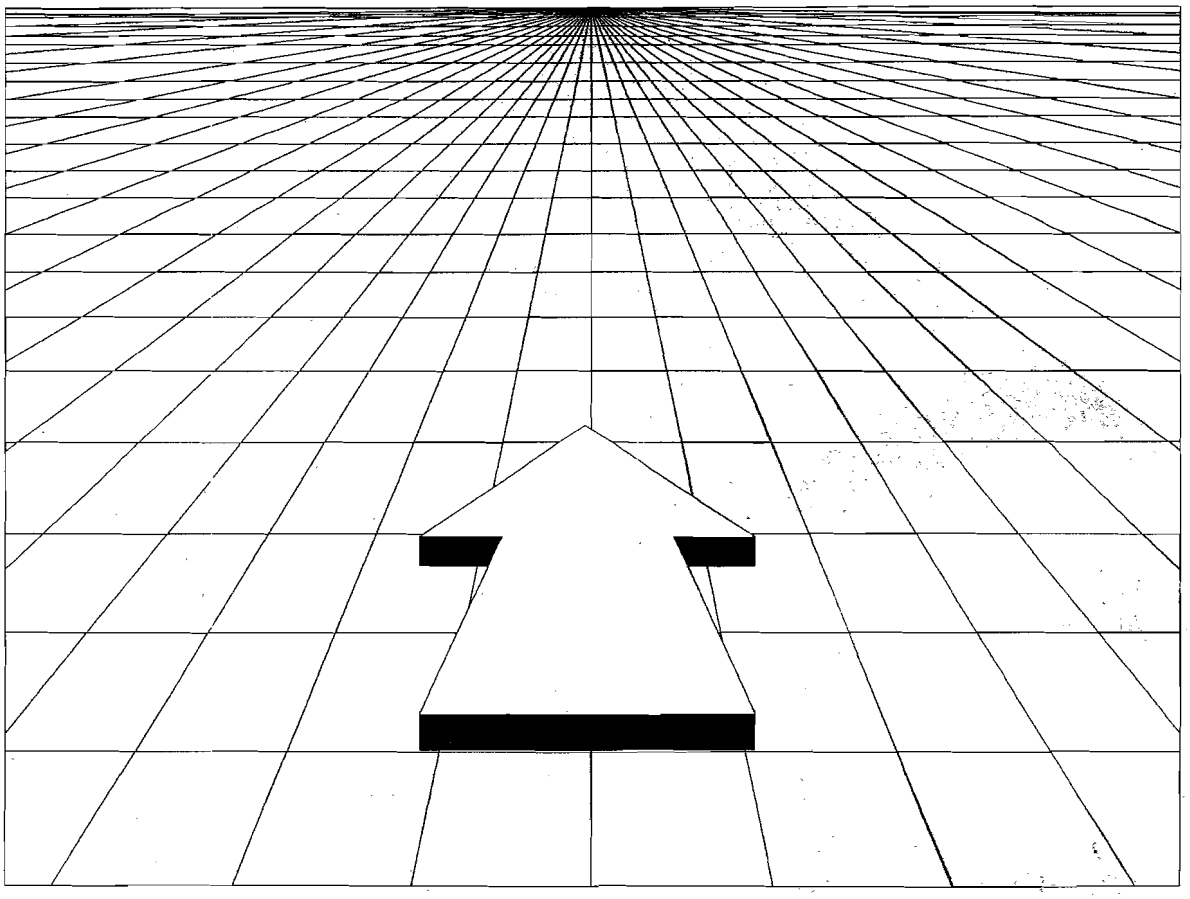
Vektorgrafik på Amiga



Af: Finn Arne Gangstød
Arnfinn Fomess & Carsten Nordenhof

DATASKOLEN
programmers choice

DATASKOLEN ApS. Postboks 62, Nordengen 18, DK-2980 Kokkedal, ☎ +45 49 18 00 77

Introduktion

Velkommen til dette kursus i Vektorprogrammering på Amiga.

Forkundskaber

I dette kursus gennemgås 2- og 3-dimensional grafik og de såkaldte vektorer. Dette kræver noget viden om de matematiske begreber SINUS og COSINUS. Vi har ikke mulighed for at forklare dig alt om det her - det lader vi skolerne om - men det mest grundlæggende vil dog blive forklaret i forbindelse med gennemgangen af dette emne.

Assemblere

En del af programmerne er skrevet til både Devpac2 og K-seka assembleren. De programmer, der er skrevet til K-Seka ligger i et directory for sig.

Disketten

Alle programeksempler ligger på den tilhørende diskette (se oversigten på side 49). Forklaringerne til eksemplerne ligger i sourcekoden.

Der ligger også nogle "readme" filer, som du selvfølgelig bør læse.

Figurer

Bagest i mappen finder du også alle illustrationerne (figurerne). De er nummererede og der henvises til dem i kurset.

Opgaver

Du kan teste din viden ved at løse opgaverne, på side 50-52.

Når du ønsker at få opgaverne rettet, indsender du løsningerne sammen med en frankeret svarkuvert til Dataskolen Aps, Postboks 62, 2980 Kokkedal, hvorefter du vil få dem rettet.

Du er velkommen til at kontakte Dataskolen og få gratis lærervejledning. Du bedes indsende dine spørgsmål sammen med en frankeret svarkuvert, og vi vil så hjælpe dig. Vi har desværre ikke mulighed for at hjælpe dig telefonisk.

Du ønskes rigtig god fornøjelse med kurset.

Nu går vi i gang:

Koordinatsystemet

Når du skal lave vektorgrafik på Amiga er der en del ting du skal have helt styr på. Lad os derfor begynde med at se på, hvordan et 3-dimensionalt koordinatsystem er opbygget.

Hvis du har et ark papir foran dig og du sætter et punkt midt på, kan du bruge dette punkt som referencepunkt når du skal beskrive, hvor eventuelle andre punkter befinder sig. Dette punkt kalder du for ORIGO.

Man bruger X- og Y-koordinater (en tænkt linie vandret på arket og en tilsvarende lodret) for at beskrive, hvor andre punkter befinder sig i forhold til referencepunktet, ORIGO. Hvis du f.eks sætter et punkt 5 cm længere oppe på arket, kan du sige at Y-koordinatet er 5.

Hvis et punkt befinder sig nedenfor ORIGO, får punktets Y-koordinat negative værdier. Et punkt 5 cm nedenfor ORIGO vil altså få koordinatværdien y = -5.

Ligesom afstanden oppe eller nede fra ORIGO kaldes Y-koordinatet, kaldes afstanden til højre og venstre for X-koordinatet. Et punkt 4 cm til højre for ORIGO vil således have X-koordinat +4, og et punkt 4 cm til venstre for ORIGO vil have X-koordinat lig -4.

Læg også mærke til at måleenheden (f.eks cm) ikke tages med når du skriver koordinaternes værdier, og at de altid nævnes i alfabetisk rækkefølge således: (X,Y).

Det vil altså sige, at et punkt tre centimeter til højre og fire centimeter nedenfor ORIGO skrives (3,-4).

Punktet ORIGO har ingen afstand fra sig selv (selvfølgelig) og kan derfor også benævnes (0,0)

Du undrer dig måske over hvorfor vi bruger centimeter og ikke millimeter eller et andet mål for afstanden? Faktisk kan man bruge en hvilken som helst måleenhed, og fordi vi skal bruge samme skala for både X- og Y-værdierne, er måleenheden ikke nødvendig i denne sammenhæng.

Den linie som går lodret op/ned fra ORIGO kaldes for Y-aksen. Tilsvarende kaldes den linie, som går vandret til højre og venstre fra ORIGO for X-aksen.

På **figur 1A** kan du se X-akseh og Y-aksen. Origo er det sted hvor de to akser (linier) krydser hinanden. Pilen til højre har sin spids i et punkt, der ligger 2 oppe ad y-aksen og 3 ude af x-aksen, eller sagt på en anden måde i (3,2).

Når du kun bruger X- og Y-koordinater (som man jo gør på et fladt stykke papir), har du altid et fladt eller såkaldt 2-dimensionalt koordinatsystem.

Hvis du nu ser på **figur 2**, så har du der et eksempel på et sådant koordinatsystem. Du har der et punkt ved navn P med koordinaterne XO og YO. Betegnelserne XO og YO står for to tal, som er afstanden fra ORIGO henholdsvis horisontalt (vandret) og vertikalt (lodret).

Den verden vi går rundt i til daglig er 3-dimensional, man kan sige, at den har dybde eller perspektiv. Når vi skal lave grafik på Amigaen, ville det derfor være dejligt, hvis vi kunne få lidt perspektiv i tegningerne.

Hvis du stiller dig foran et spejl, så er det forholdsvis enkelt at forestille sig, hvordan man kan få dybde i et fladt billede. Højden er let nok, det er Y-aksen; bredden er også til at have med at gøre, det er X-aksen; så mangler du blot afstanden "ind i" spejlet.

Hvis du nu forestiller dig en linie vinkelret på spejlet, så den går henholdsvis lige ind i og lige ud fra spejlet gennem punktet ORIGO, så har vi en linie, hvorfra vi kan måle den tredie dimension, på et ellers fladt stykke glas. Denne linie kaldes Z-aksen og målene kaldes Z-koordinaterne. Se **figur 1B.**

Så selv om skærmen på en monitor er flad og 2-dimensional, kan og skal du bruge 3 dimensioner ved udregningerne når du laver vektorgrafik..

Vektor

En vektor har en bestemt retning og en bestemt længde, men ikke en bestemt position. I **figur 1A** er vektoren (3,2) placeret på to forskellige steder i koordinatsystemet.

Når du har en trekant, kan du bruge koordinaterne for hjørnerne til at udregne to vektorer eller liniestykker.

Hvis du ser på **figur 3**, bliver vektor 1 lig med (X3-X1, Y3-Y1, Z3-Z1), og vektor 2 bliver lig (X2-X1, Y2-Y1, Z3-Z1).

> En Vektor er et liniestykke. Den bliver defineret ved tre tal eller koordinater, og de tre tal står for, hvor langt linjen går i X-retningen, Y-retningen og Z-retningen

Hvis du tager eksemplet med papirarket og ønsker at lave et 3-dimensionalt system ud af det, skal du forestille dig et punkt, som svæver over arket. Afstanden over arket kaldes da Z-koordinatet.

Punktet har en negativ Z-koordinat når det er nærmere os end ORIGO, og en positiv Z-koordinat når det er længere væk fra os end ORIGO (altså "inde i spejlet" eller "under papiret").

Når du nu skal beskrive en genstand (f.eks. en terning), opgiver du koordinaterne for hjørnerne og derefter trækker du linier mellem en del af hjørnepunkterne for at vise terningens omrids.

Radianer

Hvis du udover "bare" at kunne tegne en genstand også vil kunne rotere den, så skal du bruge en matematisk formel for at finde ud af de nye koordinater for hjørnerne efter at du har vendt genstanden.

Det første du skal se på i forbindelse med en sådan drejning af en genstand er, hvordan en rotation beskrives.

De fleste ved sikkert hvordan en vinkel beskrives eller måles i grader (°), men i matematikken er det ikke almindeligt at bruge grader - man bruger i stedet såkaldte RADIANER.

Det er lettest at forklare hvad en RADIAN er med et eksempel:

Forestil dig, at du har et ur af den "gammeldags" type med visere. Du placerer et punkt "klokken tre" og flytter det op til "klokken tolv" langs urskivens kant, det vil sige: uden at forandre afstanden ind til centrum. Nu har du drejet dette punkt 90 grader langs urskivens kant.

Og her kommer det vigtige: Radianen er forholdet mellem den LÆNGDE som du har flyttet punktet og AFSTANDEN ind til centrum (som er cirklens radius).

Cirklens omkreds kan udregnes som $2*\pi*radius$ (π er som bekendt cirka 3,141592654 .... - eller forkortet: 3,14), så en fjerdedel (90 grader) bliver $(2*\pi*radius)$ delt med 4.

Vinklen målt i RADIANER bliver $(2*\pi*radius)/(4*radius)= \pi/2$.

Det tal som beskriver vinklen har ingen betegnelse som meter, kilogram, grader eller lignende. Vinklen som svarer til 90° skrives bare 1,571 ($\pi/2$).

I nogle tilfælde tages bogstaverne RAD med bagefter tallet for at markere at det drejer sig om en vinkel angivet i RADIANER, f.eks. 1,571 RAD (som altså svarer til 90 grader).

Denne formel omregner grader til RAD:	$RAD = (2\pi * antal\ grader)/360$
og denne omregner RAD til grader:	$Grader = (360 * RAD)/(2 * \pi)$

Sinus og Cosinus

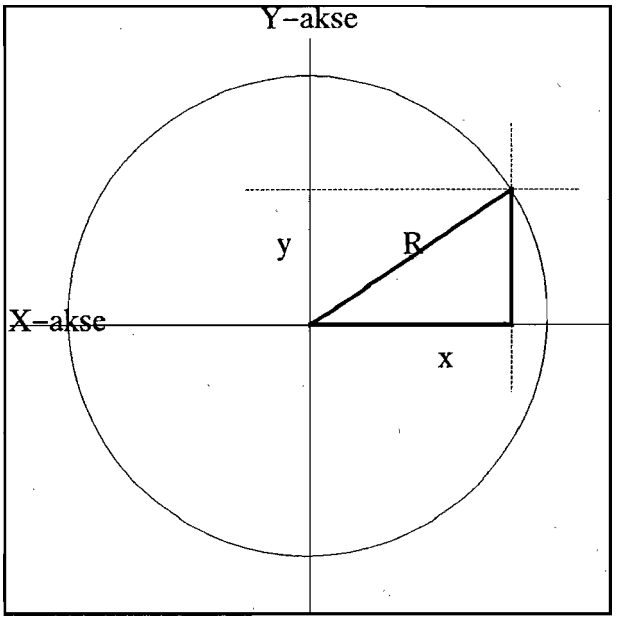
Under udregningerne af en rotation skal du også bruge nogle funktioner med betegnelserne SINUS og COSINUS. Og hvad er så det for noget?

Forestil dig et punkt i en bestemt afstand fra ORIGO, og kald afstanden fra ORIGO til dette punkt for R. SINUS er så lig med Y-koordinatet delt med R.

SINUS til en vinkel er altså forholdet mellem Y-koordinatet og cirkelbuens radius.

Den måles fra positionen "klokken tre" og i retning mod uret.

Når punktet ligger i retning "klokken tre" (eller klokken ni) bliver Y-koordinatet nul - og dermed bliver SINUS-værdien også lig med nul. For O/R (nul divideret med R) vil altid blive 0, uanset R's størrelse



SINUS og COSINUS

Altså: SINUS til en vinkel på 0 RAD (det er 0°)
eller π RAD (180°) er altid nul.

Hvis du tager SINUS for en vinkel i retning "klokken to" (π/6 RAD) vil du få en SINUS¬ værdi på 0,5. Et punkt i retning "klokken to" har altså et Y-koordinat, som er halvdelen af længden ind til centrum.

SINUS til en vinkel på π/2 RAD (90°) er 1, fordi y og R så er lige store.

COSINUS-funktionen er nøjagtig omvendt af SINUS-funktionen så her divideres X-koordinatet med Radius. Derfor er COSINUS til 0 og π RAD lig med 1 og COSINUS til π/2 RAD lig med 0.

SINUS og COSINUS forkortes til henholdsvis SIN og COS.

Hvis du ser på **figur 2** igen, så ser du, at SIN(a) (den værdi du får ud af SINUS-funktionen for vinklen "a") er lig Y0 delt med R, og at COS(a) er lig X0 delt med R. Koordinaterne kan ikke være større end afstanden ind til centrum, så værdierne du får ud af SIN() og COS() skal være mindre end eller lig 1.

Når koordinaterne er negative får du negative værdier ud for SIN() og COS(). De tal du får må altså ligge mellem -1 og 1. Radien betragtes som EN længdeenhed, uanset den er 0,5 mm eller "halvtreds" lysår lang.

Rotation

En rotation går (kort fortalt) ud på at fortælle Amiga, hvilke koordinater, der knytter sig til en bestemt figur og hvilken vinkel (hvor mange grader) figuren skal drejes. Derefter beregnes de nye koordinater, skærmen opdateres og rotationen er fuldført. Det lyder da forholdsvis enkelt - ikke sandt?

Vi begynder med at se på hvordan punkter i et to-dimensionalt system kan roteres rundt om centrum (ORIGO).

Vi siger, at du har et plan - eller en flade - i et system med kun X- og Y-koordinater (altså ingen dybde). Når punkter skal bevæges rundt om et centrum - eller et punkt - i dette plan, kan du bruge lidt forskellige formler alt efter hvad du ønsker at opnå. Du kan få en rotation med uret eller mod uret, og du kan spejlvende figuren horisontalt eller vertikalt. Prøv at se lidt på denne formel:

```
X1=COS(a0)*X0  + SIN(a0)*Y0
Y1=COS(a0)*Y0  -  SIN(a0)*X0
```

X1 og Y1 er de nye koordinater som punktet får efter det er drejet rundt om centrum. Den vinkel punkterne skal drejes er kaldt for a0, og de koordinater punktet har, før du drejer det rundt, kaldes henholdsvis X0 og Y0 **(figur 4).** Denne formel giver en rotation mod uret når vinklen a0 (som du opgiver) er positiv. Vi siger, at den har positiv omløbsretning.

F.eks: Et punkt i positionen "klokken to" bliver, når du har en vinkel større end nul, drejet rundt mod uret. Hvis vinklen er lig jt/3 RAD, bliver punktet lagt i retning klokken 12, hvis vinkelen er π/2 RAD lægges punktet i retning klokken elleve osv.

Spejl vending

Spejlvending rundt om en af akserne opnår du ved at ændre fortegnet. Formlen for en spejlvending rundt om X-aksen, får du ved at sætte et minustegn ind på den første linie således at formlen kommer til at se sådan ud:

```
X1= -  (COS(a0)*X0  +  SIN(a0)*Y0)
Y1=      COS(a0)*Y0  -  SIN(a0)*X0
```

Når du spejlvender en geristand, gøres det i urets retning. Punkterne bliver altså nu drejet en vinkel a0 MED uret.

Det var jo til at forstå, men hvilken nytte har du så af formlen, når du skal udregne de tre dimensioner?

Ved udregninger med tre dimensioner har du naturligvis tre forskellige planer eller flader et punkt kan drejes rundt i. Du har for det første XY-planet, som bruges i to-dimensionale udregninger.

Hvis du forestiller dig en urskive som står oprejst, og som du ser lige forfra, bliver dette XY-planet.

Derefter har du YZ-planet, som går indover/udover og opover/nedover. Dette svarer til en urskive, som står oprejst og som ses fra siden.

Det sidste plan er XZ-planet, og det bliver som en urskive liggende fladt ned og set fra siden.

I den formel der vistes her ovenfor, kan du sætte værdier ind for to koordinater ad gangen. Ved udregninger for tre plan, bruger du ganske enkelt formlen tre gange.

Først regner du X1 og Y1 ud med formlen, derefter bruger du den Y-værdi du fik ud sammen med den originale Z-koordinat for at regne koordinaterne ud efter drejning i YZ planet:

```
Z1=COS(a1)*Z0 + SIN(a1)*Y1
Y2=COS(a1)*Y1- SIN(a1)*Z0
```

Nu har du fået den endelige værdi for Y, men du mangler endnu en udregning for at få de endelige værdier for X og Z. Denne udregning giver rotationen i XZ-planet:

```
Z2=COS(a2)*Z1 + SIN(a2)*X1
X2=COS(a2)*X1 - SIN(a2)*Z1
```

Rotationen er bestemt af én vinkel for hvert af planerne. Her har de fået navnene a0, a1 og a2. Punktets nye position er angivet ved tallene X2, Y2 og Z2. Nu har du fået roteret punkterne i tre dimensioner, men problemerne med 3-dimensional grafik er ikke ovre endnu!

Når du skal tegne genstanden på skærmen, kan du ikke bruge Z-koordinatet, for skærmen er jo flad. Du gør det så på den måde, at du kun bruger X- og Y-koordinaterne.

For at det hele skal se naturligt ud, skal de dele af genstanden som er langt væk (og dermed har stor Z-koordinat) se mindre ud (vi skal have et perspektiv over genstanden). Dette kan opnås ved at X- og Y-koordinaterne ganges med et tal, som bliver mindre jo større Z bliver. Senere i kurset er der en rutine "rotate.s", der roterer et objekt om alle 3 akser.

Polygonrutine

Når du skal tegne fyldte vektorer, skal du bruge en rutine, som kan optegne flader i forskellige farver. En sådan rutine kaldes ofte for en POLYGON-rutine *(polygon=mangekant)*. Der findes indbyggede rutiner i Amigaen, som tager sig af det, men for at opnå maksimal hastighed kan det være bedre at lave sin egen rutine.

En del polygonrutiner har den begrænsning, at de kun kan optegne trekanter. For at få det størst mulige anvendelsesområde, vil vi i dette kursus lave en rutine, som kan tegne polygoner med lige så mange hjørner, du vil have. Det giver flere fordele, bl.a. er det hurtigere at tegne dem. Det medfører dog nogle få begrænsninger i polygonernes udseende. Denne problemstilling vender vi tilbage til senere i kurset.

I dette kursus har vi valgt at lave en ganske enkel rutine til at tegne polygoner med, og den har derfor ikke den maksimale hastighed. Fordelen er imidlertid, at den bliver lettere at forstå, den kan benyttes til næsten hvad som helst, og den er forholdsvis let at modificere til eget brug. Risikoen for alvorlige fejl bliver således også mindre.

Blitteren

For at kunne lave en polygonrutine, er det vigtigt at være klar over, hvordan blitteren arbejder, når den udfylder flader. Det første, som kan virke usædvanligt, er at blitteren arbejder fra bunden mod toppen, og fra højre mod venstre. Bit 1 i BLTCON1 *(DESCending mode)* skal derfor sættes til 1. Der er tre andre bits, som også er interessante, når det drejer sig om udfyldning:

Bit 4: **EFE** *(Exclusive Fill Enable),*
Bit 3: **IFE** *(Inclusive Fill Enable)*
Bit 2: **FCI** *(Fill Carry Input).*

Hver af disse behøver en mere indgående forklaring:

Blitteren har to modes for udfyldning, og de vælges ved at sætte *enten* **EFE** *eller* **IFE.** **FCI**-bit'en skal også sættes fornuftigt **(Se figur 5**).

Rent praktisk fungerer det sådan:

Gå pixel for pixel mod venstre. Hver gang du kommer til en pixel, som er sat, veksler du imellem at fylde og ikke at fylde. Forskellen mellem **IFE** og **EFE** er, at sidstnævnte også sletter den pixel, som slukker for udfyldningen.

Hvis **FCI** er sat, vil blitteren sætte den første pixel (og fortsætte med dette indtil den kommer til en 1'er). Hvis du derimod slukker **FCI (FCI=0),** vil blitteren ikke begynde udfyldningen før den kommer til den første 1'er.

Blitteren arbejder således:

Den laver sin egen kopi af **FCI**-bit'en - lad os kalde den **FCI'**. Når **EFE** er sat, inverteres **FCI'** for hver 1**.** I dette mode sættes **FCI'**-bit'en ind over både 0**-** og 1-bits. Eftersom **FCI'**-bit'en også erstatter Terne, vil l'eren som slukker for udfyldningen blive slettet. Når **IFE** er sat, vil den derimod bare indsætte **FCI'** for 0'erne. Den inverterer som sædvanlig **FCI'** for hver 1**.**

Der er altså mange muligheder, når man skal tegne polygoner. Vi har valgt at koncentrere os om **EFE**-udfyldning med **FCI=0. FCI**-valget er helt tilfældigt. Som du sikkert har set, vil ændring af **FCI** i **EFE**-mode give en negativ. Det spiller ingen rolle for blitteren, den kan negere hvad som helst uden at bruge tid på det.

Fordelen ved at bruge **EFE**-mode er helt tydelig - det er den enkleste og hurtigste måde at få skarpe kanter på. Hvis du tænker over det, er skarpe kanter umuligt i **IFE**-mode, da bredden af det, som fyldes nødvendigvis bliver mindst 2. Du behøver jo én pixel for at begynde udfyldningen, og én for at stoppe den.

Da blitteren i **EFE** mode sletter pixelen længst til venstre, skal der kompenseres for det ved at flytte alle linierne på den venstre side af polygonen én pixel til venstre. Hermed undgås at linierne tegnes oveni hinanden i toppen og bunden. Dermed er vi klar til at gå videre med næste skridt i polygonrutinen.

Her dukker der desværre nye problemer op. For det første kræver udfyldningen, at der er to pixels pr horisontale linie, og kun to. Hvis linien er mere horisontal end vertikal, vil den af og til være mere end 2 pixels pr horisontal linie **(figur 6A).** Det vil medføre at udfyldningen udføres helt forkert. Men dette problem kan selvfølgelig løses:

☞ Bit 1 i **BLTCON1** *(SINGle bit per horizontal line)* sætter blitteren i et speciel mode som kun giver en pixel pr horisontal linie.

Udfyldningen fungerer nu udmærket igen **(figur 6B).**

Linietegningsrutine

Det første der skal laves, er en rutine, som tager sig af linietegning. Før vi begynder, så lad os se, hvad der skal til, for at få blitteren til at tegne linier:

Blitteren opdeler sin "verden" i **8 oktanter (figur 7).** Det er gjort således, for at gøre det så enkelt som muligt for blitteren *(og mere arbejde for processoren...).* Det første, der skal gøres, er derfor at finde ud af, i hvilken oktant linien hører hjemme.

Forestil dig at linien går fra **(x1,y1)** til **(x2,y2) (figur 8).** Placer så centret for oktant-figuren i **(x1,y1).** Du skal så finde ud af, i hvilken oktant **(x2,y2)** havner, dvs i hvilken retning linien går. Hvis linien er nøjagtig på skillelinien mellem to oktanter, er det ligegyldigt hvilken af dem du vælger.

Før du i det hele taget kan begynde at tegne linier, skal blitteren sættes i linietegnings¬ -mode. Der er en del registre, som skal have bestemt indhold i denne mode, og det letteste er derfor at lave din egen rutine, som initialiserer blitteren, før du begynder at tegne linier. Den kan så udføres før den første linie tegnes, og så undgår man at tænke mere på det. Vi har lavet sådan en rutine, og kaldt den "initline".

init_line sætter:

- bltafwm til $ffff
- bltalwm til $ffff — Blitteren vil have disse tre registre på denne måde i linie-mode.
- bltadat til $8000
- bltbdat til $ffff: — Bestemmer teksturen på linien. $FFFF giver en ubrudt linie.
- a5 til $dff000: — Vi bruger A5 som baseregister når vi adresserer hardware-registrene.
- bltcmod til #ln_mod
- bltdmod til #ln_mod: — bltcmod og bltdmod indeholder bredden på det bitplane der tegnes i. Vi har valgt at lægge bredden ind i en forhåndsdefineret variabel "lnmod", således at det skal være lettest mulig at ændre. Det bliver derudover mere overskueligt.

Nu mangler vi bare at fylde de andre registre med noget meningsfyldt. Vi har valgt at bruge registrene d0-d3 til at repræsentere koordinaterne. Linien tegnes fra (d0,d1) til (d2,d3). For enkelhedens skyld kalder vi abs(d2-d0) for X og abs(d3-d1) for Y. X angiver altså hvor bred linien er, mens Y angiver højden.

Efter at have fundet ud af, hvilken oktant linien går til, ved vi om x er større eller mindre end y, og hvilket fortegn de har. Det er godt, for blitteren har ingen forståelse for, at Y kan være større end X. Dette forekommer i 4 af de 8 oktanter, og i disse skifter X og Y betydning. Vi skal derfor bytte om på X og Y i disse oktanter. Vi laver så et lille trick: Hvis Y skulle vise sig at være større end X, bytter vi bare om på de to registre, som repræsenterer y og x i udregningen. Efter at dette er gjort, kan resten af kalkulationerne udføres.

Blitter-registrene skal sættes på denne måde:

bltamod	**= 4y-4x**	
bltbmod	**= 4y**	
bltapth	**= 2y-x**	
bltcon0[bitl5-12]	**= x1 & $f**	
bltcon0[bitll-0]	**= $B5a:**	$B=use A, C & D. $5a=Minterm ->XORe linien ind.
bltconl[bitl5-12]	**= 0:**	Start bit for linie-tekstur er uinteressant.
bltconl[bit4-2]	**= Oktant**	
bltconl.SING[bitl]	**= 1:**	Dette sætter blitteren i det specielle mode som kun tegner én pixel per horisontale linie.
bltconl.SIGN[bit6]:		1 hvis linien er mindst dobbelt så lang i den ene retning som i den anden, ellers 0.
bltsize[bit5-0]	**= 2:**	Skal være på denne måde ved linietegning.
bltsize[bitl5-6]	**= X+1:**	Dette modsvarer antal pixels totalt i linien, eller linie-længden om du vil.
bltcpt.l	=	Adresse på første pixel:
bltdpt.l	=	Adresse på første pixel: (x1/16)*2 + y1*ln_mod + bitpl.adresse

Adressen på den første pixel udregnes på samme måde, som hvis vi skulle udregne adressen på koordinatet (d0,dl). Det er imidlertid vigtigt at huske, at blitteren ***skal*** have ligetals-adresser! X-koordinatet skal derfor først deles med 16 for at finde antallet af words, og så gange med to igen for at lave dem om til bytes.

Y-koordinatet ganges med antal bytes pr linie (#ln_mod). Hvis vi lægger de to tal sammen, kan vi se hvor langt fra det øverste venstre hjørne pixelen befinder sig. Hvis vi så lægger startadressen, på det bitplane vi tegner i, til, når vi frem til den endelige adresse.

Nu skulle alt være klart til selve rutinen.

Vi har kaldt rutinen **"draw_line"**

Før den kaldes op skal argumenterne lægges ind således:

d0	**=**	**første x-koordinat**
d1	**=**	**første y-koordinat**
d2	**=**	**sidste x-koordinat**
d3	**=**	**sidste y-koordinat**
a3	**=**	**adresse på den første byte i bitplanet der skal tegnes i**
ln_mod	**=**	**antal bytes per horisontale linie i bitplanet.**

Der skal også defineres en makro "LnModMul", som kan gange tal med ln_mod. Årsagen til, at vi ikke bruger en enkel "mulu #ln_mod,Dn", er at det kan udføres hurtigere på andre måder.

De næste fem instruktioner udfører i virkeligheden to opgaver:

De udregner værdien, som skal ind i bltcon0, samtidig med, at de udregner det første X-koordinats bidrag til adressen. Det fungerer således:

☞ **bltcon0 skal gives værdien $xb5a, hvor x er de fire laveste bit's af d0. For at udregne x-koordinatets adressebidrag, skal der først deles med 16 (= shif te 4 trin til højre), og så gange med 2 igen.**

move.w	**#$b5a0,d4**	d4=$****.b5a0 (*=ukendt, punktummet er der kun for at skelne imellem øvre og nedre halvdel af registret.)
swapd4		d4=$b5a0.****
move.w	**d0,d4**	d4=$b5a0.xxxx (xxxx=X-koordinatet)
ror.l	**#4,d4**	d4=$xb5a.0xxx Det er denne instruktion "som slår to fluer med et smæk". Samtidig med at den roterer de fire laveste bit's af x-koordinatet ind i det øverste word af d4, deles x-koordinatet med 16.
add.w	**d4,d4**	Dette er det eneste, som mangler - at gange den nederste halvdel af d4 med to. d4.w=(x/16)*2, mens d4.w efter en "swap d4" kan flyttes direkte ind i bltcon0.

Vi venter så længe som muligt med at flytte noget ind i blitter-registrene - i håb om at den er færdig med den forrige blit-operation, før vi er færdige med alle udregningerne. Hvis der flyttes data ind i blitter-registrene før den er færdig, vil blitteren lave alle mulige slags fejl, og det kan i værste fald få maskinen til at gå ned. Så hvis du har for travlt, medfører det bare alvorlige fejl i grafikken.

Nu er det på tide at udregne Y-koordinatets del af adressen. Det gøres sådan:

move.w	**d1,d7**	
LnModMul	**d7,d6**	Udregner Y-koordinatets del af adressen.

Læg mærke til, at indholdet af dl gemmes, da det skal bruges senere. LnModMul kan være noget så enkelt som "mulu #ln_mod,\1", men når du har besluttet dig for en ln_mod, kan det være bedre at lave en rutine, som ganger med ln_mod. F.eks, vil "lsl #6,\1" være en meget hurtigere måde at gange med 64 på. D6 er her tænkt som et register, som LnModMul kan bruge til mellemregninger.

```
        add.w   d4,d7           D7 = Afstand fra øverste venstre hjørne.
        lea.l   0(a3,d7.w),a0   A0 = Komplet adresse på første prik.
        sub.w   d0,d2           d2 = d2-d0 (=x2-x1 = bredde på linien)
        bge     xpos            Hopper til xpos hvis x2 >= x1

xneg                            Her havner du kun hvis x2 < x1, dvs linien går fra højre
                                mod venstre
        neg.w   d2              Blitteren kan ikke lide negative bredder, så vi negerer den og
                                får en positiv i stedet.
        sub.w   d1,d3           d3 = d3-d1 = y2-y1 = højden på linien
        bge     xneg_ypos       Hopper til xneg_ypos hvis y2 >= y1

xneg_yneg                       Her havner du kun hvis x2<x1 og y2<y1, dvs hvis linien går
                                opad mod venstre.
        neg.w   d3              Blitteren kan heller ikke lide negative højder, så den negeres
                                og bliver et positivt tal.
        cmp.w   d2,d3           Bredden og højden sammenlignes.
        bge.s   octant3         Hvis højden er større end bredden er vi i oktant 3.
octant7                         Oktant 7: Linien går opad mod venstre, og den er bredere
                                end den er høj, dvs den går mere mod venstre end opad.
        moveq   #(7<<2)+3,d6    Tallet, som skal ind i bltcon1, flyttes ind i d6, således at vi
                                kan vente til bagefter med at lægge det ind. Tallet (7<<2)+3
                                (=%00011111) slukker for linietegningsmode i det specielle
                                mode som udfyldning med blitteren kræver. Dette tal vælger
                                også oktant 7.
xgty                            Denne makro udføres nu, efter vi har fundet ud af hvilken
                                oktant linien hører hjemme i, startadressen på det første
                                punkt, og om bredden eller højden er størst. XGTY er den
                                makro, som bruges når linien er bredere end den er høj,
                                således at herefter er x = bredden og y = højden.

                                OBS! Denne makro afsluttes med en RTS, sådan at den
                                afslutter linietegningsrutinen! (se i øvrigt forklaring side 15)

octant3                         Linien går op mod venstre, og den går mere opad end til
                                venstre (x2<x1,y2<y1)
        moveq   #(3<<2)+3,d6    d6 = bltcon1 for oktant 3
ygt                             Denne makro modsvarer XGTY, men den bruges når højden
                                af linien er større end bredden. Forskellen er at herefter
                                bliver X = højden, mens Y = bredden. Denne makro slutter
                                også med RTS, sådan at den afslutter linietegningsrutinen.

xneg_ypos                       Her havner du hvis x2<x1
        cmp.w   d2,d3           Bredden og højden sammenlignes
        bge.s   octant2         Højden >= Bredden : Oktant 2

octant5                         Linien går nedad mod venstre, og mere til venstre end nedad
        moveq   #(5<<2)+3,d6    d6 = bltcon1 for oktant 5
```

```
      xgty                        Makro som gør resten af arbejdet når bredden er større end
                                  højden.

octant2                           Linien går nedad mod venstre, og mere nedad end til venstre
      moveq#  (2<<2)+3,d6         d6 = bltcon1 for oktant 2

ygtx                              Makro som gør resten af arbejdet når højden er større end
                                  bredden, (se i øvrigt forklaring side 15)

xpos                              Her havner du hvis linien går lige op, lige ned eller mod
                                  højre.
      sub.w   dl,d3               d3 = d3-d1 = y2-y1 = højden på linien
      bge     xpos_ypos           y2=>y1 : Hop til xpos_ypos
xpos_yneg                         x2>x1 og y2<y1, dvs linien går opad mod højre
      neg.w   d3                  Laver højden på linien om til et positivt tal
      cmp.w   d2,d3               Sammenligner bredden (d2) og højden (d3)
      bge.s   octant1             Hvis højden er større end bredden ender linien i oktant 1

octant6                           Linien går opad mod højre, og mere mod højre end opad.
      moveq   #(6<<2)+3,d6        d6 = bltcon1 for oktant 6
xgty                              Udfører resten af jobbet når bredden er større end højden.

octant1                           Linien går opad mod højre, og mere opad end til højre.
      moveq   #(l«2)+3,d6         d6 = bltcon1 for oktant 1

ygtx                              Gør resten af jobbet når højden er større end bredden

xpos_ypos                         Linien går nedad mod højre
      cmp.w   d2,d3               Sammenligner bredden (d2) og højden (d3)
      bge.s   octant0             Højden > bredden : Oktant 0

octant4                           Linien går ned mod højre, og mere mod højre end nedad.
      moveq   #(4<<2)+3,d6        d6 = bltcon1 for oktant 4
xgty                              Fuldfører linietegningen når bredden er større end højden


octant0                           Linien går nedad mod højre, og mere nedad end til højre
      moveq   #(0<<2)+3,d6        d6 = bltcon1 for oktant 0
ygtx                              Fuldfører linien når højden er større end bredden

line_end                          Slut på linietegningsrutinen.
```

Her følger definitionerne på makroerne **xgty** og **ygtx.** Disse skal stå foran "drawline"-rutinen i programmet, men vi har taget dem med hér til sidst for at få dem i den rækkefølge, som maskinen udfører dem.

XGTY MACRO

Denne macro kaldes op med:

- Adressen på den første prik i linie A0
- Bredden af linien i d2
- Højden af linien i d3
- Den del af bltcon1 som bestemmer retningen af linien i D6

(Eftersom bredden er større end højden bliver x=D2 og y=D3)

YGTX MACRO

Denne macro kaldes op med:

- Adressen på den første prik i linie A0
- Bredden af linien i d3
- Højden af linien i d2
- Den del af bltcon1 som bestemmer retningen af linien i D6

(Eftersom bredden er større end højden bliver x=D3 og y=D2)

"ygtx"-makroen er næsten helt identisk med xgty, bortset fra at d2 og d3 er byttet om. Det er pga. af blitterens "idé" om, at x altid skal være mindst lige stor som y, og det er ikke tilfældet i oktanterne 0,1,2 og 3. Derfor er blitteren "snedig", og bytter betydningen af x og y i disse oktanter. Dermed bliver x alligevel større end y.

add.w	**d3,d3**	d3 = 2y
move.w	**d3,d0**	d0 = 2y
sub.w	**d2,d0**	d0 = 2y-x
bgt.s	**xgty_nslack\@**	Hopper til "xgty_nslack" hvis x<2y, dvs linien er ikke helt dobbelt så bred som den er høj.

Dette ser måske helt uinteressant ud, men det er vigtigt for blitteren. En sådan linie vil nemlig bestå af segmenter, som er enten én eller to pixels brede, og det første segment skal altid være kun én pixel bred. Dette gøres ved hjælp af et 0 i bit 6 i bltcon1.

bset	**#6,d6**	Sætter bit 6 D6, dvs bit 6 i bltcon1. Linien er altså mindst dobbelt så bred som den er høj, og hvert segment i linien skal derfor være mindst 2 pixels bredt. Hvis denne bit er sat, giver du besked til blitteren om, at det første segment skal være mindst 2 pixels bredt.
xgty_nslack\@		Her er bit 6 i D6 sat som den skal, da linien ovenfor ellers blev sprunget over, hvis linien ikke er mindst dobbelt så bred som den er høj.

ifeq	**line_waitblit+1**	Dette er en kommando direkte til assembleren.

De følgende linier skal kun medtages i programmet, hvis "linewaitblit" er sat til -1

btst.b	**#6,dmaconr(a5)**	Checker bit 6 i dmaconr, dvs. "blitter finished"-flaget.
bne.s	**xgty_nslack\@**	Springer op igen til "xgty_nslack", hvis blitteren ikke er færdig med sit forrige job endnu.
endc		Slut på IFEQ-blokken.

Normalt skal line_waitblit være indstillet på -1. Der er kun én mulighed for at undgå denne test, og det er at lægge alle optegningsrutinerne ind i et interrupt, således at de først kaldes op når blitteren er færdig med sin forrige operation. Det har vi dog ikke gjort i dette program, men se alligevel sidst i kurset under afsnittet "Mulige forbedringer".

move.w	**d0,bltaptl(a5)**	bltapt1 = 2y-x
sub.w	**d2,d0**	d0 = 2y-2x
add.w	**d0,d0**	d0 = 4y-4x
move.w	**d0,bltamod(a5)**	bltamod = 4y-4x
add.w	**d3,d3**	d3 = 4y
move.w	**d3,bltbmod(a5)**	bltbmod = 4y
addq.w	**#1,d2**	d2 = x+1, antal pixels på linien, dvs længden som blitteren er interesseret i.
lsl.w	**#6,d2**	Shifter længden på linien 6 skridt mod venstre, sådan at den kommer i den rigtige position inden den skal lægges ind i bltsize
addq.w	**#2,d2**	Bit 1 skal også være sat i bltsize for at blitteren skal være tilfreds.
swap	**d4**	Henter værdien som skal ind i bltcon 0 og som vi udregnede i begyndelsen af linietegningsrutinen. D4 indeholder information om, hvilken bit (fra 0 til 15) linien begynder i, i adressen a0, og om hvordan linien skal tegnes.
move.w	**d4,bltcon0(a5)**	bltcon0 = $xb5a (x er de 4 laveste bits af x-koordinatet)
move.w	**d6,bltconl(a5)**	bltcon1 = oktantvalg + information om linien er dobbelt så bred som den er høj eller ej
move.l	**a0,bltcpth(a5)**	Adressen på liniens første pixels lægges ind i bltcpth
move.l	**a0,bltdpth(a5)**	og i bltdpth
move.w	**d2,bltsize(a5)**	Til sidst lægges længden på linien i bltsize. Dette starter blitteren, og skal derfor udføres til allersidst!
rts		Returnerer fra linietegningsrutinen!
endm		Slut på xgty-makroen.

Afgrænsning af polygonet

Nu er linietegningsrutinen færdig. Nu kan den bruges til at tegne polygoner, men først skal vi gennemtænke, hvad det egentlig er, vi vil. Vi skal tegne en linie rundt om hele polygonet, og disse linier skal være af den type, som blitteren kræver for at kunne udfylde rigtigt, dvs kun én pixel for hver horisontal linie. Det har linietegningsrutinen allerede taget sig af, så den sag er i orden. Vi skal også flytte alle stregerne på den venstre side af objektet én pixel til venstre, da vi bruger EFE-udfyldningen. Det sidste problem er blot at lave hjørnerne rigtige.

Hvis vi optegner polygonerne med det vi har indtil nu, bliver det kun top- og bundhjørnerne, som bliver korrekte **(figur 9).** Vi skal altså forhindre, at en linie slutter på samme linie, som en anden begynder på. Den enkleste måde, at klare det på, er at tegne en linie, hvor vi fjerner den første prik. Vi har allerede udregnet adressen på denne prik, så det skulle ikke volde nogle særlige problemer.

Det spiller også en vigtig rolle, i hvilken retning linierne tegnes. Blitteren sætter den første pixel på hver ny linie den kommer til, og en linie som tegnes opad mod højre kommer længere mod venstre end en, som tegnes nedad mod venstre **(figur 10).** Dette kan udnyttes ved at tegne alle fladerne så store som mulige, således at der ikke kommer grimme huller imellem dem. For at opnå dette, skal linierne tegnes imod toppen og bunden som vist på **figur 11.**

Det er kun to af linierne rundt om polygonet, som skal tegnes normalt, dvs uden at den første prik fjernes. Det er en af hver af de linier, som går ud fra henholdsvis punkt A og B. Læg mærke til, at punkt A altid er det punkt, som er længst mod venstre, uanset om det er i toppen eller bunden. Tilsvarende er B altid det punkt, som er længst mod højre.

Linietegningsrutinen i dette kursus er lavet så den tegner linier uden den første prik.

Nu har vi planlagt og fundet ud af hvilken metode, der skal benyttes til at tegne linierne rundt om polygonet. Nu mangler "bare" udfyldning, kopiering ind i bitplanerne og "klipning".

Kopiering

Rutinen, som tegner hovedobjekterne, skal også indeholde information om, hvor objektet blev tegnet. Når objektet skal flyttes, skal grafikken, som lå bag ved tegnes igen. Man skal derfor vide nøjagtigt, hvilket område som skal tegnes igen. Du kan selvfølgelig lagre en kopi af hele skærmen bag ved vektorobjekterne og kopiere den tilbage igen mellem hver gang vektorobjekterne tegnes. Dette tager imidlertid ret lang tid. Det er meget bedre at finde den mindste firkant, som er stor nok til at dække hele objektet, Og så nøjes med at kopiere grafikken i denne firkant.

På den måde har vi lavet en vektorrutine, som gør det nemmere at vise vektorobjekter over baggrundsgrafik. Du sparer en også en del hukommelse og lidt tid, hvis du kan nøjes med ensfarvet baggrund. Det eneste du så skal gøre er at slette området, hvor objektet blev tegnet.

Klipning

En vigtig del af en polygonrutine, er "klipning" eller rettere sagt beskæring af kanterne. Hvis man undgår at tegne et polygon, der har et eller flere hjørner uden for skærmen, vil det umiddelbart se godt nok ud. Hvis derimod en del af polygonet havner udenfor skærmen, og det mange, så må du benytte dig af den teknik, der kaldes "klipning" eller "clipping".

Hvis man tegner hele polygonet op - og ikke ønsker at det helt eller delvist havner -udenfor skærmen, skal man være opmærksom på følgende:

- ☞ Det, der kommer udenfor til venstre kommer ind igen fra højre.
- ☞ Derudover vil data, som kommer udenfor top- og bundkanten overskrive andre vigtige data i hukommelsen.

Det er derfor nødvendigt at finde en løsning på problemet! Kunsten, kun at optegne den del af polygonet, som vises på skærmen, kaldes klipning. Der er flere måder at gøre det på, men vi har valgt at benytte den enkleste, selvom den ikke er den hurtigste i alle tilfælde.

Polygonet optegnes i et midlertidigt bitplane, *temp-plane,* som er 512 x 512 pixels stort. Det skal kunne indeholde selv de største polygoner. For at få bedst mulig plads til polygonerne, tegnes i det øverste venstre hjørne af *temp-plane.* Polygonerne kan så være ca. 500 pixels høje og brede førend polygonrutinen nægter at optegne dem. Det er dog nemt at undgå sådanne størrelser.

Hvis man får brug for endnu større polygoner, kan det midlertidige bitplane udvides til 1024 x 1024 pixels. Dette optager imidlertid 128 kB chipmem mod de 512 x 512 pixels, som "kun" optager 32 kB.

Før polygonet tegnes flyttes det så langt mod venstre og opad som muligt for at få plads til et så stort polygon som muligt. For at forenkle både klipning og kopiering til de endelige bitplanes, flyttes polygonet et eller flere hele words til venstre. Det vil sige, at ***k*16*** trækkes fra x-koordinatet, hvor ***k*** er et heltal.

Før du begynder at tegne et polygon, skal du på forhånd have fundet ud af, hvilket punkt som er længst til venstre og hvilket der er øverst. X-koordinatet, som er længst til venstre kaldes for *minx* og y-koordinatet, som er øverst for *miny.*

For at kunne flytte polygonet derhen man ønsker, trækkes **(minx & 16)** fra alle x-koordinateme og **miny** fra alle y-koordinaterne. Dette betyder, at det øverste hjørne havner helt i toppen af det midlertidige bitplane, mens det venstre hjørne placeres imellem **x=0 og x=15,** dvs i *wordet* længst til venstre.

Når polygonet skal kopieres, skal de samme tal lægges til igen. Her får vi imidlertid brug for klipningen. Hvis den venstre kant af polygonet er til venstre for skærmkanten, skal du undlade at kopiere den del ind. Tilsvarende skal du heller ikke tage de dele med, som er over toppen, til højre for den højre kant og under bunden. Dermed undgås at dele af polygonet havner udenfor skærmen.

Klipningen er illustreret i **figur 12.** Der ser du, hvor meget de forskellige kanter skal klippes, for at det skal se godt ud. I eksemplet i **figur 12** er polygonet så stort, at det går udenfor skærmen på alle 4 sider - noget man sjældent kommer ud for. Det er imidlertid nemmest at se, hvordan det fungerer med denne illustration.

Vertikal klipning foregår ved, at hver enkel pixel "klippes", mens klipning horisontalt foregår med hele *word,* dvs 16 pixels. På venstre side skal du altså trække polygonets venstre x-koordinat fra skærmens x_koordinat i det midlertidige bitplan. Derefter lægges 15 til og resultatet divideres med 16 ((min x -x4 + 15)>>4). Grunden til at du skal lægge 15 til er, at du på den måde sikrer, at du beskærer billedet med et helt word - også selvom det kun er 1 pixel for stort til skærmen.

I visse situationer slipper man også for at udfylde den del af polygonet, der kommer under og over skærmen. Det samme gælder for den del som kommer ud over den venstre kant. Dog **skal** den højre kant udfyldes, fordi blitteren arbejder fra højre mod venstre. Den skal bruge den første streg i polygonet for at kunne "aktivere" udfyldningen. Derimod stopper den af sig selv, når den kommer til venstre skærmkant.

Hvis man ikke begynder helt fra højre med udfyldningen, kan man være sikker på at polygonet **ikke** bliver udfyldt. Det er derfor nødvendigt at udfylde en større del af polygonet end det, som er skraveret i **figur 12.**

For at illustrere, hvordan en klipnings-rutine virker i assembler er der lagret et eksempel **"poly.s"** på disketten. Den indeholder både kildekoden, udfyldningen, klipningen og kopieringen af polygonet på skærmen.

Rotationsrutine

Du er nu så langt at du kan rotere et objekt om alle 3 akser. Rotationen rundt om akserne er defineret ved disse vinkler **(figur 14):**

rx : Vinklen der roteres rundt om x-aksen
ry : Vinklen der roteres rundt om y-aksen
rz : Vinklen der roteres rundt om z-aksen

(nx,ny,nz) er koordinatet før rotationen

(X,Y,Z) er koordinatet efter rotation

Rotation rundt om x-aksen:

Z=nz * cos(rx)-ny * sin(rx)
Y=ny * cos(rx)+nz* sin(rx)

Rotation rundt om y-aksen:

X =nx*cos(ry)-z*sin(ry)
nz=z * cos(ry)+nx * sin(ry)

Rotation rundt om z-aksen:

nx=x*cos(rz)-y*sin(rz)
ny=y*cos(rz)+x*sin(rz)

Positiv rx giver rotation mod urets retning set fra højre
Positiv ry giver rotation mod urets retning set ovenfra
Positiv rz giver rotation mod urets retning set forfra

En komplet rotationsrutine i assembler, som benytter disse formler finder du i "rotate.s"

Som du ser, skal vi bruge de trigonometriske funktioner Sin og Cos for at kunne udregne rotationen. Disse funktioner kan selvfølgelig udregnes, men det vil forsinke vektorrutinen alt for meget. For at spare tid, udregner vi derfor en del sinus- og cosinusværdier på forhånd, og lægger dem ind i en tabel. (Det kan f.eks gøres i et basicprogram).

Man skal gennemtænke disse tabeller grundigt før man begynder at lave dem. Det første problem man støder på, er at Sin og Cos returnerer værdier mellem -1 og 1. F.eks. er Sin(1.3rad)=0.963558... Mikroprocessoren kan kun behandle heltal, og så er der kun tre muligheder, -1, 0 og 1. Det er selvfølgelig alt for lidt. Vi løser det med at gange med et så stort tal som muligt, men alligevel passe på at tallene ikke er for store til at passe ind i et **word.** Et passende stort tal er 16384 (2^{14}). Vi får så værdier, som ligger mellem -16384 og 16384. Det giver en præcision, der er mere end god nok, til at bevægelserne på skærmen skal blive jævne. Vi tager lidt mere fat på dette emne i kapitlet "Måling af vinkler".

Måling af vinkler

I matematikken er det, som nævnt før, almindeligt at måle vinkler i enten grader eller radianer. I radianer ligger vinkelmålene mellem 0 og 2π(=6.283185...). Det giver imidlertid alt for dårlig opløsning og det går heller ikke op i et helt tal. Hvis du måler i grader løser du disse problemer, og du har da 360 forskellige vinkler til rådighed. Det kan for så vidt være nok til at kunne lave brugbare og jævne bevægelser, men tallet 360 passer ikke specielt godt til datamaskiner. Hvis du benytter grader, skal gradtallet ligge mellem 0 og 359. Forestil dig, at du for at rotere et objekt, skal lægge et fast tal til eller trække det fra på vinklen hver gang. Hvis vinklen bliver mindre end 0, skal du lægge 360 til, og hvis den bliver større end 359 skal du trække 360 fra. Dette bliver alt for møjsommeligt.

Så er det meget bedre at vælge et nyt vinkelmålesystem, således at tallene kan justeres med en enkelt AND-instruktion. Et gunstigt system er 1024 "datagrader". Eftersom hvert element i tabellen er et word, vil det sige at hele tabellen bliver 2048 bytes (=2 kB) lang. Det er så kort, at det ikke har nogen særlig betydning for ledig hukommelse, samtidig med at det er langt nok til at kunne give helt jævne bevægelser.

For at udføre behandlingen hurtigst muligt, accepteres kun datagrader i lige tal, som ligger mellem 0 og 2046, dvs almindelige datagrader multipliceret med 2. Det gøres for at kunne adressere tabellen direkte med datagraderne. For at kunne justere en datagrad i et data-register, skrives derfor bare "AND #2046,Dn", hvor Dn er det aktuelle dataregister. På den måde er du sikker på at få lige tal, som ligger mellem 0 og 2046.

Når du skal have fat i en bestemt *sin*- eller *cos*-værdi, skal du bruge et adresseregister, som peger på begyndelsen af tabellen - vi kalder det An. Du skal også bruge et dataregister, som indeholder datagraderne - vi kalder det Dn. Svaret kommer ud i et andet dataregister, Dm.

Det hele gøres således: **"MOVE.W 0(An,Dn.w),Dm".**

Et konkret eksempel: A3 peger på starten af Cosinus-tabellen. Du vil finde cosinus til datagraderne, som ligger i D0, og du vil have den lagt i D1. Det bliver da sådan:

"MOVE.W 0(A3,D0.w),D1".

Cosinus- og sinus-tabeller

Der ligger både en sinus- og cosinustabel på disketten. Nedenfor vises den interessante del af et basicprogram, som kan bruges til at udregne cosinustabellen:

```
FOR t = TO 1023
 v = t/6.283186                    (Laver datagrader om til radianer)
x% = INT(COS(v) * 16384 + .5)      (x% er den værdi som skal lægges ind i tabellen)
NEXT
```

Denne formel bruges til at omregne radianer R til datagrader D:

$$D=INTEGER(\frac{R\cdot 1024}{2\Pi}+0{,}5)$$

Vinklen 0.7 radianer giver 114 datagrader, som svarer til 228 ligetalsdatagrader, når den indsættes i denne formel.

Vi kan så regne tilbage igen og se hvilken radian-vinkel 114 grader svarer til:

For at omregne datagrader til radianer:

$$R=\frac{D\cdot 2\Pi}{1024}$$

Hvis der indsættes 114 i denne formel fås 0.6994952...

Når basicprogrammet udregnet tabelværdien for 114 datagrader, bruger det altså vinklen 0.6994952 radianer.

Tabelværdien bliver så:

INT(cos(0.6994952)* 16384+0.5) = INT(0.7651672..*16384+0.5) = INT(12537.0004...)=12537.

Det vil altså sige når:

du vil finde Cos(0.7rad) = Cos(40.11grader) = Cos(114datagr) = Cos(2281igedatagr)

Cos(0.7rad)=0.76484...

Cos(2281igedatagr)=12537/16384=0.76520.., en unøjagtighed på kun 0,047%. Det er så lidt, at det ikke har nogen betydning i praksis.

Perspektiv

Med det tredimensionale koordinatsystem kan vi give alle punkter en koordinat, uanset hvor de befinder sig. Problemet med 3-dimensional grafik er, at den skal vises på en 2-dimensional monitor. Øjet skal derfor "snydes" til at se en ekstra dimension. Det gøres ved at lægge perspektiv på.

Perspektiv i sig selv er langt fra nogen stor gåde, og **figur 15** skulle give et klart billede af, hvordan det udregnes. Det er faktisk ikke andet end at dividere alle X- og Y-koordinater med afstanden fra øjet (=Z-værdi efter rotation) før de optegnes.

Eksempel på Rotation og perspektiv

Nu har du lært nok til at kunne lave en lille demonstration med prikker, som bevæger sig i 3-dimensionale mønstre. I demoen "dots1" ser du et eksempel på dette. Sourcefilen hedder "dots1.s".

I "dots1" har vi lavet en prik-figur, som kan styres med et joystick. Den demonstrerer rotation og perspektiv **(figur 16).** Læg mærke til, at vi i tillæg til Z-værdien også lægger en fast "tænkt" afstand fra øjet til skærmen til, sådan at selv negative z-værdier kan vises på skærmen. Disse skal man forestille sig ligger mellem skærmen og den som ser på, altså i luften udenfor skærmen. (Se evt. afsnittet "Koordinatsystemet").

Fyldte flader

Prikker er én ting, men det er noget helt andet når vi skal bruge fyldte flader. Her støder vi på en del nye problemer. Lad os først tage et objekt, som alle har stiftet bekendtskab med - terningen. Tag en terning i hånden, og vend og drej den. Lige meget hvor meget du anstrenger dig, kan du aldrig se mere end 3 af de 6 sider. I nogle positioner ser du måske ikke mere end 1 eller 2 sider.

Disse erfaringer er gode at have i baghovedet, når vi taler om datagrafik. De sider, som du ikke kan se, er det jo heller ikke nødvendigt at optegne! På den måde undgås at tegne ca halvdelen af alle polygonerne i hvert objekt. Vi skal bare finde en enkel og let måde at finde ud af hvilke sider, som ikke er synlige. En utrolig enkel måde at gøre det på, er "clockwise"-metoden. Den indebærer, at alle polygonerne skal have koordinater, som går i urets retning, når de er synlige **(figur 17).**

Se på dette eksempel som illustration: Tag en diskette eller et stykke papir på omtrent samme størrelse. Vælg en side som du kalder forsiden. På den skriver du "1" i øvre venstre hjørne, "2" i øvre højre hjørne, "3" i nedre højre hjørne og "4" i nedre venstre hjørne. Vend så disketten/papiret og skriv de samme tal, som står i det tilsvarende hjørne på den anden side. Nu vil du se, at ligegyldigt hvordan du end vender og drejer arket/disketten, vil

tallene altid være i urets retning så længe du ser forsiden, og de vil altid være imod urets retning når du ser på bagsiden.

Konvekse objekter

DEFINITION AF KONVEKST OBJEKT

Enhver ret linie fra et punkt på et objekt til et andet punkt på objektet, skal enten gå langs med overfladen eller igennem objektet.

Hvis der findes mindst én linie, som bryder denne regel, er objektet ikke konvekst.

Eksempler på konvekse objekter:

- KUBE
- PYRAMIDE
- FODBOLD

Vend tilbage til terningen igen. Læg mærke til, at enten er en side usynlig, eller også ser du hele siden. Det sker aldrig at en side bliver delvist dækket af en af de andre sider. Hvis du skal tegne sådan et objekt, er det derfor bare at finde ud af hvilke af siderne, som er synlige, og så tegne dem i **vilkårlig rækkefølge.** Sådanne objekter kaldes **KONVEKSE** objekter, dvs objekter, som aldrig kan "skygge" for sig selv. På disketten findes et program, som viser en kube med 6 forskellige farver. Det hedder "kube1.s" som kildekode, og det ligger også i eksekverbar form som "kube1". I sourcekoden "kube1.s" ligger også forklaring på programlinierne.

Inkonvekse objekter

Hvis alle objekter havde været konvekse, så havde det ikke vært noget problem at lave vektorrutiner med det vi har lært indtil nu. Imidlertid er de allerfleste objekter inkonvekse. Et eksempel på det er et almindeligt telefonrør. Kunsten er at opdele et inkonvekst objekt i flere konvekse underobjekter **(figur 18A og 18B).** Så kan hvert underobjekt tegnes som et enkelt konvekst objekt, det vil sige, at kun de synlige sider optegnes. Det eneste problem er hvilken rækkefølge disse underobjekter skal tegnes i. Det ser vi på i næste kapitel.

Sorteringsalgoritme

Vi bruger en lille avanceret algoritme, som har flere fordele: Den er hurtig og den fungerer uden fejl, når visse forudsætninger er opfyldt. Der er dog én ulempe: Man skal tænke sig grundigt om, når man designer objekter. Vi har derfor lavet nogle opgaver, som du kan øve dig på og som netop omhandler dette, til sidst i kapitlet: "Opgaver".

Sortering af underobjekter

Metoden forklares bedst ved hjælp af nogle eksempler: Se på **figur 19.** Den består af to klodser; I og II. Læg mærke til at når flade A er synlig, så er klods II altid foran klods I. Det er dette fænomen, som vi udnytter i vores optegnings-algoritme. Når klods II skal være foran klods I, skal klods I optegnes først, da klods II skal optegnes ovenpå klods I for at kunne ses.

'Når derimod flade A er usynlig, vil klods I altid være foran klods II. Så skal klods II altså tegnes først, da klods I så skal tegnes ovenpå klods II. Vi kalder dette for at *"sortere"* underobjekterne, fordi vi sorterer (eller ordner) dem i den rækkefølge, de skal tegnes i.

Denne algoritme er meget hurtig, fordi vi allerede har checket om flade A er synlig eller ej. Det gøres som et led i optegningen, da vi vil vide hvilke flader, det er nødvendigt at tegne. Denne sorteringsalgoritme kaldes for Z-sortering.

Teorien, der ligger bag, er følgende: Hvis du forestiller dig, at en af fladerne udvides til uendelig størrelse, så vil alle underobjekter, der befinder sig på den ene side af fladen kunne "sorteres" ved hjælp af denne algoritme. Underobjekter, der skærer flader vil derimod ikke kunne sorteres.

Det er måske lettere at forstå med et eksempel. Prøv at se **figur 20:**

Hvis du udvider flade A til en uendelig størrelse, så vil både objekt II og III være placeret fuldstændigt på den højre side af denne. Fladen vil derimod dele objekt IV i to, så objekt IV kan dermed ikke "sorteres" ved hjælp af flade A.

Vi har imidlertid ikke opbrugt alle vore muligheder endnu. Klods I har 5 andre flader, og her kan flade E udvides, således at objekt IV være fuldstændig på undersiden af denne. Vi har nu fået et "sorteringsgrundlag" for alle underobjekterne i forhold til objekt I. Fremgangsmåden er den samme for de øvrige 3 underobjekter. Som du ser, er det allerede blevet lidt indviklet - selv med så få underobjekter - så mere komplicerede objekter kan blive endnu vanskeligere. Det er derfor vigtigt, at koncentrere sig om opgaven og at du forstår denne teori. Gennemgå den eventuelt flere gange og husk *at øvelse gør mester!*

Vi har derfor lavet nogle objekter bagest i hæftet, som du kan øve dig på. Det er præcis denne del af at designe objekter, der er den mest tidskrævende, og det er også den del, det er nemmest at lave fejl i. Brug derfor en del tid på at øve dig. Når du behersker kunsten, kan du lave store flotte objekter, som - helt uden bugs - kan bevæges i alle mulige retninger.

Hovedobjekter

Der er en del ting, som er vigtige at være klar over, når du skal "udtænke" hovedobjekterne:

☞ Du skal lave en koordinattabel for hvert hovedobjekt. Koordinattabellen skal indeholde X, Y og Z koordinater for alle hjørner, som er brugt i hovedobjektet. Antallet af koordinater *fratrukket 1* skal også med i hovedobjektbeskrivelsen. Hvis f.eks et hovedobjekt har brug for 37 koordinater, skriver du tallet 36 i hovedobjektbeskrivelsen.

I hovedobjektbeskrivelsen opgiver du adressen på koordinattabellen. Derudover opgiver du adressen på en "optegnings-koordinattabel" og en "udregnings-tabel". Når alle koordinaterne er roteret, lægges de nye koordinater ind i udregningstabellen. Derefter lægges der perspektiv på, skærm-offset lægges til således at midtpunktet bliver midt på skærmen, og koordinaterne lægges ind i optegningstabellen.

Der ligger de som (x,y)-koordinater, z-koordinatet er altså faldet ud. Det er disse koordinater, der bruges direkte ved optegningen af skærmen - deraf navnet.

☞ **Udregningstabellen**

Du skal afsætte (antal koordinater * 6) bytes til udregningstabellen.

☞ **Optegningstabellen**

Du skal afsætte (antal koordinater * 4) bytes til optegningstabellen.

☞ **Hovedobjekterne skal opdeles i flere underobjekter**

☞ **Hvert af disse underobjekter** SKAL **være konvekse, ellers fungerer optegningen ikke ordentligt.**

☞ **Hvert af underobjekterne skal tildeles et nummer, og det første underobjekt får nummeret 0. Det næste får nummer 1, det næste 2 osv...**

☞ **Der er ingen grænser for, hvor mange underobjekter du kan have med, men pas på at det ikke bliver alt for kompliceret! Det er en fordel at inddele hoved-objekterne i så få underobjekter som muligt.**

☞ **Det er en stor fordel, at det første underobjekt kan bruges til at sortere** ALLE **de øvrige underobjekter i hovedobjektet**

Hvis du ikke kan sortere alle de andre underobjekter i forhold til det første underobjekt, er der to andre måder at løse problemet på:

1. Den ene er, at tillægge ekstra usynlige flader på det underobjekt, der ser mest lovende ud. Disse flader kaldes *"fantom-flader"*. Fantom-flader laves ved at sætte farvenummeret på både forsiden og bagsiden til - 1 . (Se i øvrigt afsnittet om, hvordan man laver underobjekter).

2. Den anden metode består i, at lave et ekstra *"fantom-underobjekt"*. Dette "fantom-underobjekt" skal have den egenskab, at alle sidefladerne er usynlige, dvs. både forsidefarven og bagsidefarven er sat til - 1 , på alle fladerne i underobjektet. Hvis du laver dette kløgtigt, kan du komme til at sortere alle de øvrige underobjekter ved hjælp af det.

☞ **Hvis du vil vise flere hovedobjekter på skærmen er det forholdsvist enkelt. Du skal blot optegne de hovedobjekter, som er længst væk, først, og de nærmeste til sidst.**

Underobjekter

Her kommer en oversigt over, hvad du skal være opmærksom på, når du laver underobjekter:

☞ **Alle underobjekter skal være konvekse.**

Det er dog en sandhed med visse modifikationer. I **figur 21** har vi illustreret 2 forskellige, fuldstændig lovlige underobjekter. Som du kan se, behøver underobjekter ikke at være lukkede. Og så kan man jo strengt taget ikke sige, at de er konvekse, men udtrykket får alligevel betydning.

Hvis du udvider fladerne i objektet i **figur 21B,** får du samme objekt som i **figur 21A** - og det er konvekst. Altså, hvis du har et konvekst objekt, så kan du "krympe" en eller flere af sidefladerne, og stadig kalde det for konvekst.

☞ **Hver sideflade kan have hver sin farve, hvor farven også kan være forskellig på for- og bagsiden.**

Hvor mange farver du har til rådighed, afhænger - som du måske ved - af antallet af Bitplanes.

1 Bitplane: 2 farver (0-1)
2 Bitplane: 4 farver (0-3)
3 Bitplane: 8 farver (0-7)
4 Bitplane: 16 farver (0-15)
5 Bitplane: 32 farver (0-31)
6 Bitplane: Ekstra halfbrite: 64 farver (0-63)

Bemærk at farvenumrene begynder med 0, og ikke 1.

Husk, at hvis du bruger baggrundsgrafik, skal farverne deles imellem vektorobjekterne og baggrundsgrafikken!

Hvis du opgiver et for højt farvenummer, vil farvenummeret automatisk blive justeret ned på et acceptabelt niveau. Det er fordi farverne vælges fra ligeså mange af de nederste bits, som der er Bitplanes:

- Ved 1 Bitplane tages der kun hensyn til den nederste bit i farvenummeret
- Ved 2 Bitplane tages der kun hensyn til de 2 nederste bits i farvenummeret
- Ved 3 Bitplane tages der kun hensyn til de 3 nederste bits.....osv

- BEGRÆNSNING: Farvenummeret skal være positivt, dvs mindre end 32768 ($8000)

☞ **Hold and Modify-mode er i de fleste tilfælde ikke specielt godt egnet til vektorgrafik.**

☞ **For- og/eller bagsiden kan gøres usynlig ved at give dem farvenummeret -1.**

☞ **I lukkede underobjekter vil "bagsiden" eller "indersiden" på fladerne aldrig være synlige.** Hvis du vil øge hastigheden af optegningen betydeligt, skal du sætte bagsidefarven til -1 på lukkede underobjekter.

☞ **I åbne underobjekter skal du huske på, at sætte farven for bagsiden af fladerne til noget andet end -1.** Det ser dumt ud, hvis en flade lige pludseligt forsvinder ud i den blå luft. Oftest ser det godt ud med en lidt mørkere farve på indersiden end på ydersiden.

☞ **Koordinaterne i underobjektet er fælles for hele hovedobjektet.** Du skal altså lave en koordinattabel for hvert hovedobjekt - ikke for hvert underobjekt. Det er fordi mange af underobjekteme hænger sammen og hvis hvert underobjekt skulle have sin egen koordinattabel, vil mange af koordinaterne komme med i flere af tabellerne.

☞ **Koordinaterne opgives ved indeks til koordinattabellen.** Første koordinat har nummeret 0, andet koordinat har nummeret 1 osv.

Koordinatnumrene **skal ganges med** 4, før det lægges ind i underobjektet da hvert koordinat altid optager 4 bytes:

1 word for X-koordinat og 1 word for Y-koordinat. Vi gør det på den måde for at kunne adressere dem direkte med (An,Dn.w)-adresseringsmetoden, hvor An peger på koordinat 0 og Dn er koordinat-indekset, efter at være ganget med 4.

☞ **For <u>hver</u> af sidefladerne i underobjekterne skal du udtænke underobjektssorteringen.** Du skal altså finde ud af, hvor mange af de øvrige objekter, som ligger HELT på forsiden (ydersiden) af hver enkel flade, hvis fladen udvides til at blive uendelig stor. Dette antal *fratrukket 1* skal ind i underobjektbeskrivelsen. Derudover skal adressen på en tabel over adresserne på de aktuelle underobjekter opgives. Hvis der 3 underobjekter, som er helt på forsiden af en flade, opgives nummeret 2. Hvis der ikke er nogen, opgives -1. Hvis der er opgivet -1, er værdien af tabeladressen ligegyldig, men den skal alligevel være med! Det skyldes at længden på alle underobjektstrukturer skal være ens.

Det lyder værre end det faktisk er. Se eventuelt i det tidligere eksempel "poly.s", hvor vi gennemgår design af et komplet objekt. Der ligger en komplet programlistning på disketten.

Beskrivelse af hoved- og underobjekter ved hjælp af strukturer

Nu er vi kommet til, hvordan hovedobjekterne og underobjekterne skal beskrives ved hjælp af strukturer.

☞ **Strukturer er egentlig lister, som kan indeholde forskellige typer af værdier, mens**

☞ **Tabeller bruges, hvis det er en liste med elementer af samme type.**

Vi benytter her en speciel måde at skrive dem op på:

For hver linie er der angivet en størrelse, f.eks byte, word, longword..., og hvor mange der skal være af dem.

"B:" Betyder 1 byte
"<2>B:" Betyder 2 bytes
"<3>B:" Betyder 3 bytes osv...
"W:" Betyder 1 word
"<2>W:" Betyder 2 words osv...
"L:" Betyder 1 longword
"<2>L:" Betyder 2 longwords osv.
"<n>B:" Betyder n antal bytes

Hvert element er tildelt et navn, således at man kan adressere de enkelte elementer ved deres navn. Så slipper man for at huske på offsettet hele tiden. Derudover står der foran hvert led i en struktur, hvor mange bytes fra begyndelsen, som dette element befinder sig. Det er det vi kalder for "offset". Skrivemåden er sådan:

navn:(offset) <Antal>Størrelse: Forklaring

Hvis navnet starter med et udråbstegn, betyder det, at denne variabel kun benyttes internt og at du bør undgå at skrive til den. Du kan imidlertid læse værdierne, eftersom de af og til kan være interessante. Navnene skrives uden udråbstegn i programmerne.

Hvis forklaringen er efterfulgt af en stjerne (*) betyder det, at det er en adresse på en ny struktur/tabel, som forklares nedenfor.

Hovedobjektstrukturen:

!objStatus(0)	W: Status word.
!objMinX(2)	W: Mindste X-koordinat ved optegning.
!objMinY(4)	W: Mindste Y-koordinat ved optegning.
!objMaxX(6)	W: Største X-koordinat ved optegning.
!objMaxY(8)	W: Største Y-koordinat ved optegning.
objX:(10)	W: Hovedobjektets X-koordinat (positivt mod højre)
objY:(12)	W: Hovedobjektets Y-koordinat (positivt nedad)
objZ:(14)	W: Hovedobjektets Z-koordinat (positivt ind i skærmen)
objRX(16)	W: Rotation rundt om X-aksen
objRY(18)	W: Rotation rundt om Y-aksen
objRZ(20)	W: Rotation rundt om Z-aksen
objNoCrd(22)	W: Antal koordinater totalt i koordinattabellen minus 1.
objCrd(24)	L: Adresse på koordinattabellen*
objClCrd(28)	L: Adresse på udregnings-koordinattabellen*
objPlCrd(32)	L: Adresse på optegnings-koordinattabellen*
objSubs(36)	L: Adresse på underobjekttabellen. * Der skal altid være mindst et underobjekt!
objNoSubs(40)	W: Antal underobjekter - 1 .

De værdier som skal sættes, når du designer objektet, er:

NoCrd, Crd, ClCrd, PlCrd, Subs og NoSubs (dvs de 6 nederste).

Værdierne, som sættes i programmet, og som skaber bevægelse er:

X, Y, Z, RX, RY og RZ.

Værdier, som du **aldrig** skal sætte, er de 5 øverste:

Status, MinX, MinY, MaxX og MaxY.

De bruges af optegningsrutinerne til at lagre midlertidige resultater etc, og hvis du ændrer på dem, kan det få kedelige konsekvenser.

Koordinattabellen er ganske enkelt opbygget. Den består simpelthen af koordinaterne, der er organiseret på denne måde:

x0, y0, z0, x1, y1, x1, x2, y2, x2 osv, hvor alle del-koordinaterne er words. Hvert punkt beskrives af x,y og z-koordinatet, og det kræver derfor 6 bytes per koordinat. Antal elementer i denne tabel er lig med objNoCrd+1. Længden bliver altså (objNoCrd+1)*6 bytes.

Koordinaterne skal sættes på forhånd. Det er dem, som bestemmer objektets udseende.

Udregnings-koordinattabellen er opbygget på nøjagtig samme måde som koordinattabellen, og er akkurat lige så stor. Koordinaterne i denne tabel behøver ikke at blive sat til nogen bestemt værdi, eftersom optegningsrutinerne selv indsætter værdier i denne tabel.

Optegnings-koordinattabellen er derimod lidt speciel. Eftersom skærmen er 2-dimensional, behøver vi kun X- og Y-koordinater for at beskrive punkterne på skærmen. Efter at der lægges perspektiv på reduceres de 3 dimensioner derfor i udregnings-koordinattabellen til 2 i optegnings-koordinattabellen. Den er opbygget på følgende måde:

x0, y0, x1, y1, x2, y2, osv....

Også her er hver enkelt del-koordinat et word, så et komplet optegnings-koordinat optager 1 longword (=4 bytes). Det er heller ikke nødvendigt at indlægge værdier på forhånd i denne tabel; det tager optegningsrutinerne sig af.

Underobjektstrukturen beskriver, hvordan de enkelte dele af objektet hænger sammen. Den ser sådan ud:

IsubStatus(0)	W: Status-word. Bruges af optegningsrutinen - kun til internt brug!
subNoPoly(2)	W: Antal sideflader i underobjektet fratrukket 1.
subPolys(4)	<subNoPoly+1>L: Adresse på hver enkelt sideflade, som udgør underobjektet.*

Som du ser, er det umuligt at sige hvor lang underobjekt-strukturen er. Længden varierer alt efter, hvor mange flader, som underobjektet har.

Sidefladerne beskrives af en "poly"-struktur. Den er ret kompleks, og indeholder blandt andet fuldstændig information om sorterings-rækkefølgen af underobjekterne.

Opbygningen er nu alligevel ganske enkel:

IpolyStatus(0)	W:	Status-word. Kun til internt brug!
polyFCol(2)	B:	Farve på forsiden (ydersiden). -1 betyder usynlig forside. OBS! BYTE!
polyBCol(3)	B:	Farve på bagsiden (indersiden). -1 betyder usynlig bagside. OBS! BYTE!
polySame(4)	L:	Adresse på en flade i samme plan **(figur 22)**. Sættes til 0, hvis det ikke er nogen. Bruges af synligheds-checkrutinen for en hurtigere afvikling.
polyNoFrn(8)	W:	Antal underobjekter foran denne flade fratrukket 1
polyFrnTb(10)	L:	Adresse på en tabel over underobjekter, som ligger foran denne flade.*
polyNoCrd(14)	W:	Antal hjørner i polygonet fratrukket 1. Et polygon skal have mindst 2 hjørner.
polyCords(16)	<polyNoCrd+l>W:	Koordinatindeks. Husk! Disse skal være koordinatnummer * 4!

Tabellen over underobjekter er også ganske enkel. Den er opbygget af <polyNoFrn+1> longwords med adresser på underobjektstrukturerne, som er foran den aktuelle flade. Alle værdier i denne tabel skal sættes på forhånd som et led i objektets design.

Valg af koordinater til polygonet

Når du skal vælge koordinater til hvert enkelt polygon, er det vigtigt at huske, at få dem i den rigtige rækkefølge. På **figur 23** ser du en enkel pyramide, som er et godt eksempel til at illustrere hvad der er den rigtige rækkefølge. Koordinaterne nævnes i en rækkefølge, således at du kan trække en streg efter dem, og det vil f.eks for bund-polygonet ikke være korrekt at sige at rækkefølgen er 1-3-2-4.

Derudover skal alle koordinaterne være i urets retning, når du ser fra **ydersiden** af underobjektet mod fladen.

Den korrekte rækkefølge for hver af fladerne på pyramiden på **figur 23** bliver da:

1-5-2,2-5-3,3-5-4,4-5-1 og 1-2-3-4.

Det er ligegyldig hvilket af hjørnerne du begynder med, så den første flade kunne lige så godt have haft koordinatrækkefølgen 5-2-1 eller 2-1-5.

Hvis koordinaterne ikke er i urets retning, vil rutinen, som skal checke om fladen er synlig eller ej, melde fejl. Det vil medføre fejl i optegningen af både underobjektet og i den rækkefølge underobjekterne optegnes i. Det er derfor vigtigt at få dette rigtigt fra starten.

Kube/pyramiderutine

Nu skal vi til at bruge al det vi har lært, til at lave et enkelt fyldt vektorobjekt, som roterer på skærmen.

Koordinaterne på hjørnerne er opgivet i **figur 24,** og det er derfor naturligt at begynde med koordinattabellen:

```
CoordTable
        dc.w    -100,-50,-50
        dc.w    0,-50,-50
        dc.w    0,50,-50
        dc.w    -100,50,-50
        dc.w    -100,-50,50
        dc.w    0,-50,50
        dc.w    0,50,50
        dc.w    -100,50,50
        dc.w    50,-50,-50
        dc.w    50,-50,50
        dc.w    50,50,50
        dc.w    50,50,-50
        dc.w    100,0,0
```

Det skulle være det, 13 koordinater i alt. Derudover skal vi lave to andre koordinattabeller, en for udregning og en for optegning:

```
CalcCoords
        ds.w    13*3            ;Afsætter 39 words, som er nok til 13 udregnings-koordinater.
DrawCoords
        ds.l    13              ;Afsætter 13 longwords, som er nok til 13 optegnings-
                                koordinater.
```

Vi kan da nu gå direkte til hovedobjektstrukturen:

```
objektet
        dc.w    0,0,0,0,0       ;De 5 første words bruges kun af interne optegningsrutinen
                                Vi sætter dem til startværdien 0.
        dc.w    0,0,0,0,0,0     ;x,y og z koordinat og rotation sættes af rotationsrutinen.
        dc.w    12              ;13 koordinater ialt, (antal -1 skal opgives her!)
        del     CoordTable      ;Adressen på koordinattabellen.
        del     CalcCoords      ;Adressen på udregningskoordinattabellen
        del     DrawCoords      ;Adressen på optegnings koordinattabellen
        del     SubTable        ;Adresse på liste over underobjekterne
        dc.w    1               ;Antal underobjekter -1 (objektet består af 2 underobjekter)
```

Nu er turen kommet til underobjekterne. Som du ser på figuren, består objektet af to tydeligt definerede underobjekter. Vi kalder kube-underobjektet for "Cube" og pyramide-underobjektet for "Pyramid".

Underobjekt-listen, som er kaldt "SubTable" i hovedobjektstrukturen, bliver sådan:

SubTable

```
del   Cube,Pyramid.
```

Kuben er nu underobjekt 0, og pyramiden er underobjekt 1.

Cube

```
dc.w   0     ;status-wordet bruges kun af de interne rutiner.
dc.w   5     ;6 sideflader i kube-underobjektet. Her skal antal -1 opgives.
del    poly0,poly1,poly2,poly3,poly4,poly5   ;Adresser på polygonerne, som udgør
                                              underobjektet.
```

Det **var** så **kuben,** og **her er Underobjektstrukturen** for pyramiden:

Pyramid

```
dc.w   0
dc.w   4   ;5 sideflader i pyramiden
del    poly6,poly7,poly8,poly9,poly10
```

Nu mangler vi bare polygonstrukturerne. Som du sikkert husker, foregår sorteringen af underobjekterne i polygonbeskrivelserne. Som du ser ligger pyramiden helt foran sideflade 2, og kuben ligger helt foran sideflade 10. Vi kan ikke sige noget om de andre sideflader, men de er heller ikke interessante. Det er nok med en sideflade for at afgøre sorteringen.

For at få flere farver på objektet, giver vi hver sideflade farvenummer 10 + nummeret på sidefladen. Sideflade 0 får altså farve 10, sideflade 1 farve 11 osv...

poly0

```
dc.w     0
dc.b     10,-1    ;Forsidefarve:  10. Bagsiden: usynlig (objektet er lukket)
del      0        ; Ingen flader i samme plan
dc.w     -1       ;Ingen underobjekter foran denne sideflade
del      0        ;Vi opgiver adressen 0 når der ikke er nogen objekter foran denne flade.
dc.w     3        ;Dette polygon har 4 hjørner. Her skal antal hjørner -1 opgives.
dc.w     0,1*4,2 *4,3*4
                  ;Koordihatindeksene til dette polygon ganget med 4. At gange koordinatindeksene
                  med 4 er ret vigtigt.
```

```
poly1
        dc.w    0
        dc.b    11,-1   ;Forside: farve 11. Bagside: usynlig
        del     0       ;Ingen flader i samme plan
        dc.w    -1      ;Ingen underobjekt foran fladen
        del     0
        dc.w    3       ;4 hjørner
        dc.w    4*4,5*4,1*4,0
                        ;Koordinat 4, 5, 1 og 0.

poly2
        dc.w    0
        dc.b    12,-1   ;Farve 12, bagside usynlig
        del     0
        dc.w    0       ;Et underobjekt er foran denne flade (pyramiden)
        del     FrontPol2Table
                        ;Adresse på en liste over underobjekteme, som er foran denne flade.
        dc.w    3       ;4 hjørner
        dc.w    1*4,5*4,6*4,2*4
                        ;Hjørner: 1-5-6-2

FrontPol2Table
        del     Pyramid ;pyramiden er foran polygon 2.

poly3
        dc.w    0
        dc.b    13,-1   ;Farve 13, bagside usynlig
        del     0
        dc.w    -1
        del     0
        dc.w    3       ;4 hjørner
        dc.w    5*4,4*4,7*4,6*4
                        ;Koordinat 5-4-7-6

poly4
        dc.w    0
        dc.b    14,-1   ;Farve 14, bagside usynlig
        del     0
        dc.w    -1
        del     0
        dc.w    3       ;4 hjørner
        dc.w    3*4,2*4,6*4,7*4     ;Koordinat 3-2-6-7

poly5
        dc.w    0
        dc.b    15,-1   ;Farve 15, bagside usynlig
        del     0
        dc.w    -1
        del     0
        dc.w    3       ;4 hjørner
        dc.w    4*4,0,3*4,7*4   ;Koordinat 4-0-3-7
```

Dette var polygonstrukturerne til kuben. Læg specielt mærke til **poly2,** hvor der er en reference til pyramiden. Hvis poly2 er synlig, skal pyramiden tegnes efter kuben, ellers

skal den tegnes før. Studer dette eksempel grundigt, sådan at du er sikker på, hvordan det fungerer.

Nu kommer resten af polygonstrukturerne:

```
poly6
        dc.w    0
        dc.b    16,-1   ;Forside: 16, bagside usynlig
        del     0
        dc.w    -1
        del     0
        dc.w    2       ;3 hjørner
        dc.w    8*4,12*4,11*4  .-Koordinaterne  8-12-11

poly7
        dc.w    0
        dc.b    17,-1   ;Forside 17, bagside usynlig
        del     0
        dc.w    -1
        del     0
        dc.w    2       ;3 hjørner
        dc.w    9*4,12*4,8*4   ;Koordinaterne  9-12-8

poly8
        dc.w    0
        dc.b    18,-1   ;Farve 18, bagside usynlig
        del     0
        dc.w    -1
        del     0
        dc.w    2       ;3 hjørner
        dc.w    12*4,9*4,10*4  ;Koordinaterne  12-9-10

poly9
        dc.w    0
        dc.b    19,-1   ;Farve 19, bagside usynlig
        del     0
        dc.w    -1
        del     0
        dc.w    2       ;3 hjørner
        dc.w    11*4,12*4,10*4      ;Koordinateme  11-12-10

poIylO
        dc.w    0
        dc.b    20,-1   ;Farve 20, bagside usynlig
        del     0
        dc.w    -1      ;Ingen flader foran er ikke helt sandt - se forklaring nedenfor
        del     0
        dc.w    3       ;4 hjørner
        dc.w    9*4,8*4,11*4,10*4  ;Koordinat  9-8-11-10
```

Når vi siger, at kuben ikke er fuldstændig foran flade 10, så er det faktisk forkert, men det er alligevel ikke nødvendigt at opgive dette i poly10-strukturen.

Det er muligt at afgøre, hvilken af de to underobjekter, som skal tegnes først helt og holdent ved hjælp af flade 2. Det har derfor ingen betydning at bestemme det samme en gang til ved hjælp af en anden sideflade. En gylden regel er derfor, at det ikke har nogen betydning at have sådanne referencer tilbage til underobjekt 0, dvs start-underobjektet. Eftersom alle underobjekter skal være sorteret i forhold til det første underobjekt, medfører det bare ekstraarbejde med at finde ud af dette for hvert eneste underobjekt, som man kommer til.

Nu har vi faktisk lavet hele objektet! Det var ikke så svært, vel? Under alle omstændigheder: Studer dette eksempel grundigt vær sikker på, at du forstår alt førend du går videre til det næste eksempel.

Ovennævnte eksempel ligger i to versioner på disketten og hedder henholdsvis "obj1nb.s" og "obj1b.s". Den første er et objekt på en ensfarvet baggrund, mens den anden indlæser et 32-farvers billede og lægger det bagved.

Stjernerutine

Nu er tiden inde for at tage fat på et lidt mere avanceret objekt. På **figur 25** ser du hvilket objekt, som vi har tænkt at lave. Det er ikke så meget vanskeligere, men det indeholder 7 underobjekter, hvoraf et af dem er et fantom-underobjekt.

Vi begynder som sædvanlig med koordinattabellen:

```
StarCoords
        dc.w    0,-200,0
        dc.w    -50,-50,-50
        dc.w    50,-50,-50
        dc.w    50,-50,50
        dc.w    -50,-50,50
        dc.w    0,0,-200
        dc.w    200,0,0
        dc.w    0,0,200
        dc.w    -200,0,0
        dc.w    -50,50,-50
        dc.w    50,50,-50
        dc.w    50,50,50
        dc.w    -50,50,50
        dc.w    0,200,0
```

Alt i alt er der 14 koordinater. Vi sætter som sædvanlig plads af til udregningskoordinater og optegningskoordinater:

```
StarCalcCoords
        ds.w  14*3

StarDrawCoords
        ds.l   14
```

Før vi begynder på hovedobjektstrukturen, skal vi finde ud af, hvor mange underobjekter der skal være. Der er 6, som umiddelbart skiller sig ud: Hver stjernetak er et underobjekt. Problemet er, at stjernerne ikke umiddelbart indeholder nogen sideflader, som gør at de kan sorteres i forhold til hinanden. Vi har så to muligheder; enten at indlægge et ekstra fantom-underobjekt eller at indlægge fantom-flader i underobjekterne.

Begge metoder virker, og her skal vi se på fantom-flader. Vi får derfor 6 underobjekter totalt. På disketten ser du også, hvordan det kan gøres med et fantom-underobjekt - mere om dette senere.

På **figur 25** har vi illustreret, hvordan en fantom-flade skal lægges på en af takkerne - Det skal gøres på tilsvarende måde for alle de andre sider, da objektet er fuldstændigt symmetrisk opbygget. Det er derfor også ligegyldigt, hvilken en af takkerne, som man vælger som objekt 0 - vi har valgt det øverste. Hvert af underobjekterne får fantom-fladen med på ialt 5 sideflader. Vi er nu klar til at begynde på hovedobjektstrukturen:

star

```
        dc.w    0,0,0,0,0       ;De-5 første words bruges kun internt i optegningsrutinerne
        dc.w    0,0,0,0,0,0     ;x, y og z-koordinater og rotationsvinkler...
        dc.w    13              ;14 koordinater ialt
        del     StarCoords      ;Adresse på koordinattabellen
        del     StarCalcCoords  :Adresse på udregnings-koordinattabellen
        del     StarDrawCoords  ;Adresse på optegnings-koordinattabellen
        del     StarSubTable    ;Adresse på liste over alle underobjekterne
        dc.w    5               ;6 underobjekter (antal minus 1 skal opgives her!)
```

Nu er turen kommet til underobjekterne. Først skal vi imidlertid have en tabel med over alle underobjekterne:

StarSubTable

```
        del     StarUp,StarLf,StarFr,StarRg,StarBk,StarDn
```

På figuren ser du, at vi har givet underobjekterne følgende navn og nummer:

0:Up, 1:Lf, 2: Fr, 3:Rg, 4:Bk, 5:Dn

Her har vi også tilføjet "Star" foran alle navnene således, at det er tydeligt at de tilhører "Star"-hovedobjektet.

StarUp

```
        dc.w    0
        dc.w    4  ;5 sideflader i hver "tak"
        del        upPoly0,upPoly1,upPoly2,upPoly3,upPolySort
```

Som du ser her, kan du give polygonerne de navne, som du vil. Det er ofte bedst at give dem et navn, så man kan se hvilket underobjekt, de tilhører.

```
StarLf
        dc.w    0
        dc.w    4
        del     lfPoly0,lfPoly1.lfPoly2,lfPoly3,lfPolySort

StarFr
        dc.w    0
        dc.w    4
        del     frPoly0,frPoly1,frPoly2,frPoly3,frPolySort

StarRg
        dc.w    0
        dc.w    4
        del     rgPoly0,rgPoly1,rgPoly2,rgPoly3,rgPolySort

StarBk
        dc.w    0
        dc.w    4
        del     bkPoly0,bkPoly1,bkPoly2,bkPoly3,bkPolySort

StarDn
        dc.w    0
        dc.w    4
        del     dnPoly0.dnPoly1,dnPoly2,dnPoly3,dnPolySort
```

Så enkelt var det. Nu mangler vi imidlertid alle polygonstrukturerne. For dette objekts vedkommende er det der, hvor det største arbejde ligger: Der er 6*4 = 24 forskellige flader i denne figur. For at få det maksimale antal farver med, giver vi dem forskellige farver.

```
upPoly0
        dc.w    0
        dc.b    1-1             ;Forsidefarve 1, bagsiden usynlig.
        del     0               ;Ingen flader i samme plan
        dc.w    -1              ;Ingen underobjekter foran denne sideflade
        del     0
        dc.w    2               ;Trekant
        dc.w    1*4,0,2*4       ;Koordinaterne 1-0-2

upPoly1
        dc.w    0
        dc.b    2-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    2*4,0,3*4       ;2-0-3

upPoly2
        dc.w    0
        dc.b    V I
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    3*4,0,4*4       ;3-0-4
```

```
upPoIy3
        dc.w    0
        dc.b    4-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    4*4,0,1*4       ;4-0-1

upPolySort
        dc.w    0
        dc.b    -1,-1
        del     0
        dc.w    4               ;5 underobjekter foran denne flade (alle de andre faktisk!)
        del     FrontUpTable
        dc.w    3               ;4 hjørner
        dc.w    1*4,2*4,3*4,4*4

FrontUpTable
        del     StarLf,StarFr,StarRg,StarBk,StarDn

lfPoly0
        dc.w    0
        dc.b    5,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    8*4,1*4,9*4     ;Koordinaterne 8-1-9

lfPoly1
        dc.w    0
        dc.b    6-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    8*4,4*4,1*4     ;8-4-1

 lfPoly2
        dc.w    0
        dc.b    7,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    4*4,8*4,12*4    ;4-8-12

lfPoly3
        dc.w    0
        dc.b    8,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    8*4,9*4,12*4    ;8-9-12
```

```
lfPolySort
        dc.w    0
        dc.b    -1,-1
        del     0
        dc.w    3               ;4 underobjekter foran denne flade.
        del     FrontLfTable
        dc.w    3               ;4 hjørner
        dc.w    1*4,4*4,12*4,9*4    ;1-4-12-9

FrontLfTable
        del     StarFr,StarRg,StarBk,StarDn

frPoly0
        dc.w    0
        dc.b    9,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    1*4,2*4,5*4     ;Koordinaterne  1-2-5

frPoly1
        dc.w    0
        dc.b    10,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    2*4,10*4,5*4    ;2-10-5

frPoly2
        dc.w    0
        dc.b    11,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    5*4,10*4,9*4    ;5-10-9

 frPoly3
        dc.w    0
        dc.b    12,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    1*4,5*4,9*4     ;1-5-9

frPolySort
        dc.w    0
        dc.b    -1,-1
        del     0
        dc.w    3               ;4 underobjekter foran denne flade.
        del     FrontFrTable
        dc.w    3               ;4 hjørner
        dc.w    2*4,1*4,9*4,10*4    ;2-1-9-10
```

```
FrontFrTable
        del     StarLf,StarRg,StarBk,StarDn

rgPoly0
        dc.w    0
        dc.b    13,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    2*4,7*4,6*4     ;Koordinaterne 2-7-6

rgPoly1
        dc.w    0
        dc.b    14,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    6*4,7*4,11*4    ;6-7-11

rgPoly2
        dc.w    0
        dc.b    15,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    10*4,6*4,11*4   ;10-6-11

rgPoly3
        dc.w    0
        dc.b    16,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    2*4,6*4,10*4    ;2-6-10

rgPolySort
        dc.w    0
        dc.b    -1,-1
        del     0
        dc.w    3               ;4 underobjekter foran denne flade.
        del     FrontRgTable
        dc.w    3               ;4 hjørner
        dc.w    3*4,2*4,10*4,11*4   ;3-2-10-11

FrontRgTable
        del     StarFr,StarLf,StarBk,StarDn
```

```
bkPoly0
        dc.w    0
        dc.b    17,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    3*4,4*4,7*4     ;Koordinaterne  3-4-7

bkPoly1
        dc.w    0
        dc.b    18,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    7*4,4*4,12*4    ;7-4-12

bkPoly2
        dc.w    0
        dc.b    19,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    11*4,7*4,12*4   ;11-7-12

bkPoly3
        dc.w    0
        dc.b    20,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    3*4,7*4,11*4    ;3-7-11

bkPolySort
        dc.w    0
        dc.b    -1,-1
        del     0
        dc.w    3               ;4 underobjekter foran denne flade.
        del     FrontBkTable
        dc.w    3               ;4 hjørner
        dc.w    4*4,3*4,11*4,12*4       ;4-3-11-12

FrontBkTable
        del     StarFr,StarRg,StarLf,StarDn

dnPoly0
        dc.w    0
        dc.b    21,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    9*4,10*4,13*4   ;Koordinateme   9-10-13
```

```
dnPoly1
        dc.w    0
        dc.b    22,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    10*4,11*4,13*4      ;10-11-13

dnPoly2
        dc.w    0
        dc.b    23,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    11*4,12*4,13*4      ;11-12-13

dnPoly3
        dc.w    0
        dc.b    24,-1
        del     0
        dc.w    -1
        del     0
        dc.w    2
        dc.w    12*4,9*4,13*4  ;12-9-13

dnPolySort
        dc.w    0
        dc.b    -1-1
        del     0
        dc.w    3               ;4 underobjekter foran denne flade.
        del     FrontdnTable
        dc.w    3               ;4 hjørner
        dc.w    12*4,11*4,10*4,9*4      ;12-ll-10-9

FrontdnTable
        del     StarFr.StarRg.StarBk.StarLf
```

Læg mærke til at StarUp aldrig nævnes i Front-tabellerne i de andre underobjekter, selv om den kunne have været med. Det er dette vi forklarede tidligere - det har ingen betydning at tage den med, eftersom optegningsrutinen altid starter på underobjekt 0.

Dette var alt som skal til for at kunne lave objektet i **figur 25.**

På disketten ligger der to versioner af dette programeksempel. Begge versioner viser stjernen på et billede, men stjernen er lavet på to forskellige måder.

I **"startsides"** ligger det eksempel, som vi lige har gennemgået, hvor vi brugte "fantom-sideflader" for at bestemme underobjektssorteringen.

I **"starfsub.s"** har vi brugt et "fantom-underobjekt". Lige netop i dette tilfælde er det faktisk, det som er det letteste. Bortset fra den måde, som underobjekterne er lavet på er de to programeksempler helt identiske.

Double- og triplebuffering

Hidtil har vi ikke talt om, hvordan vi har fået jævne bevægelser, selv om vi har haft det med i de fleste af programeksemplerne. Metoden kaldes "double-buffering". I nogle af eksemplerne bruges til og med "triple-buffering".

Det går ud på, at man har mere end en skærm i hukommelsen samtidig. Samtidig med at en skærm vises, tegnes en anden i en buffer. Dette gør, at du aldrig ser skærmen blive opdateret, og du slipper derfor for den generende flimren. Hvis man kun bruger 2 skærme må man i visse tilfælde vente lidt før man kan begynde at tegne. For at opnå maksimal hastighed ved optegning, skal man bruge 3 skærme. Det vil sige: Den første vises, den anden er klar til at vises, mens der tegnes på den tredie. Hvis man har for lidt RAM kan det godt blive lidt problematisk med så mange skærme. 3 skærme med 32 farver i 320x256 opløsning optager 150 kB. Hvis du så også skal bruge baggrundsgrafik, skal der afsættes yderligere 50 kB. Det giver tilsammen 200 kB - og alt dette er CHIPMEM, så det kan blive et alvorligt problem.

Når du er færdig med at optegne en skærm, sættes bitplane-pointer-registrene således som du vil have dem. Det, som kan medføre forsinkelser er, at den forrige skærm vises, og elektronstrålen kan være hvor som helst på den nye skærm imens. Hvis du skal opnå en jævn overgang, skal du vente indtil elektronstrålen har gennemkørt hele skærmen før du skifter til den skærm, som nu skal vises. Altså, hvis du kun har to skærmbilleder, skal du vente på at elektronstrålen skal blive færdig, før du kan begynde at tegne igen. Med 3 skærme undgås netop dette.

Figur 26 viser, hvordan triple-buffering fungerer, og hvorfor double-buffering ikke er lige så hurtigt.

For at kunne skifte det skærmbillede, som vises mens elektronstrålen ikke optegner billedet på skærmen, benyttes et interrupt: "vertical blank"-interruptet. Det kaldes op hver gang et billede er vist færdigt på monitoren, og hvis du er hurtig kan du nå at ændre skærm-registrene før skærm-optegningen begynder igen.

Dette interrupt løser også et andet problem for os. Hvis du har en hurtig maskine, A3000 etc..., kan du komme ud for det problem, at maskinen når at optegne vektorobjekterne for hurtigt; det vil sige mere end et objekt per skærmopdatering. Maskinen skal derfor tvinges til at vente til skærmopdateringen er færdig.

Det er ikke så svært som det umiddelbart lyder. Hvis objekterne optegnes så hurtigt, fås jævne bevægelser på skærmen. Det er helt umuligt at få bevægelserne til at se bedre ud. Det kan kun opnås kun ved relativt enkle objekter når de er "langt væk" og de derfor skal se små ud. Ved at indlægge denne pause opnås, at der aldrig kommer rod på skærmen, uanset hvor hurtig maskinen er.

Vi har lavet et program, som viser sådan en bevægelse. Sourcefilen hedder "obj2ef.s", og i assembleret form "obj2ef".

Animationseksempel

Indtil nu har vi kun diskuteret, hvordan vi skal tegne enkle objekter, og i princippet hvordan vi kan tegne flere objekter samtidig på skærmen. Det er imidlertid et stort skridt fra dette og til at lave en lille "film" eller lignende.

På disketten ligger programmet "anim1.s", i eksekverbar form "anim1". Det er en relativt kort animation, som viser nogle af de muligheder du har.

Mulige Forbedringer

Der er en del ting, som du selv kan gøre for at forbedre de rutiner, som er blevet gennemgået i dette kursus. De fleste af programmerne er lavet så enkelt som muligt, for at de skulle være både tydelige og nemme at forstå. Vi har dog, der hvor det har været nødvendigt, optimeret rutinerne, sådan at hastigheden blev forbedret. Der er imidlertid masser af muligheder tilbage for at forbedre rutinerne.

Vektorrutinerne på Interrupt

Den største forbedring får du ved at lægge vektorrutinerne ind på Interrupt. Det medfører at du sparer en masse tid, idet du slipper for at vente på at blitteren skal blive færdig.

Du skal starte med at lave din egen "Interrupt-handler". Hver gang blitteren bliver færdig med en opgave, sætter den en Bit i INTREQ-registret. Hvis den tilsvarende bit i INTENA-registret er sat, vil dette give et Interrupt, som du kan bruge. I stedet for at skrive direkte til blitter-registrene i linietegningsrutinen og i de andre optegningsrutinen kan du gemme disse værdier i en tabel. Interrupt-handleren skal så læse disse værdier fra tabellen og give dem til blitteren, så snart der kommer et nyt Interrupt.

På denne måde sikrer du, at blitteren har noget at lave hele tiden samtidig med at processoren også arbejder på fuld tid.

Lav en rutine til at tegne trekanter, firkanter.....osv

Du kan også optimere rutinerne ved at lave rutiner til at tegne trekanter, firkanter osv. En rutine, som udelukkende er beregnet på at tegne trekanter, vil tegne en trekant noget hurtigere end vores generelle rutine, som kan styre polygoner med over 50 hjørner.

Ulempen ved denne metode, er at programmet bliver noget større, og at du i polygonstrukturen skal opgive, hvilken rutine, som skal tegne det aktuelle polygon.

Simulation af lyskilde på 3D-objekter

I mange sammenhænge kan det være flot at simulere en lyskilde på 3D-objekterne. Det burde være relativt nemt at tilføje en lyskilde til rutinerne i dette kursus - de er forberedt til det.

I "udregnings-koordinattabellen" har du koordinaterne efter rotation, og disse kan bruges til at udregne lysstyrken på de enkelte flader. Der findes flere gode "lysmodeller" at vælge imellem; den mest kendte er sikkert "Phong-shading". Det kræver dog ganske gode matematiske kundskaber for at kunne programmere lyskilder.

Animationsmodul

For at kunne lave flotte præsentationer, demoer og lignende er det vigtigt at få bevægelse i objekterne. Det betaler sig at udvikle sit eget animationsmodul, sådan at du slipper for at lave i hundredvis af tabeller for at få tilføjet en lille bevægelse. Ofte ser man effekter, hvor f.eks en klods forvandles til f.eks et rumskib eller lignende. Der kan godt være 50 billeder i sådan en "forvandling", og det smarteste er da bare at droppe klodsen og rumskibet, og så lave en rutine, som kan udiegne alle "mellemformerne".

Animations-strukturer

Du bør også overveje, at lave dine egne "animations-strukturer", sådan at de animerede objekter bliver lette at holde styr på. Det skulle så ikke være en uoverkommelig opgave at lave f.eks en robot, der går på en naturlig måde.

Editor

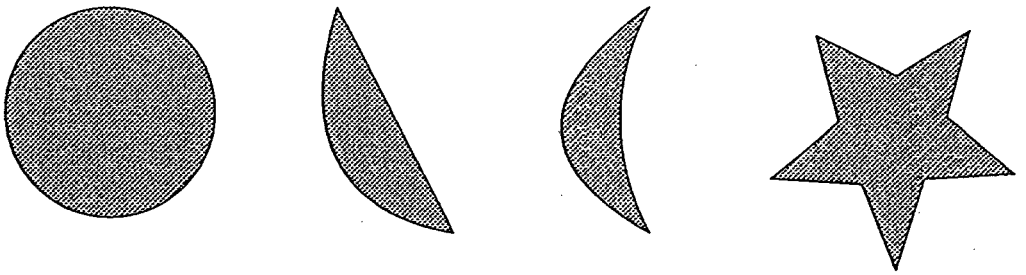
Når man har designet nogle få 3D-objekter, finder man hurtigt ud af, at man godt kan bruge en editor specielt til 3D-objekter. Det vil sige et program, hvori du kan tegne objekterne, hvorefter programmet (og ikke dig) vil finde koordinaterne for hjørnerne. Det kan laves i assembler, men det går også an at lave en i Basic. Hvis du har en editor, går det meget hurtigere at lave objekter - specielt hvis de også skal animeres. Der findes i øjeblikket ikke en sådan editor på markedet, så det er noget du kan tage som en udfordring at lave selv.

Oversigt over programmer på disketten

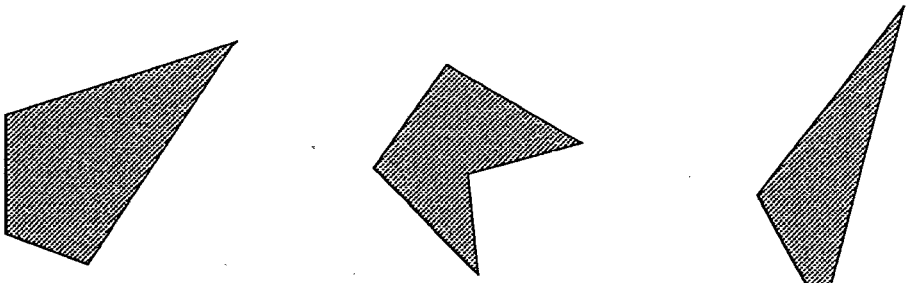
Background.Raw	Baggrundsgrafik til Anim1, StarFSide, StarFSub
LesMeg	Oplysninger om disketten, som ikke er med i selve kursus-materialet
MakeSinCosTable.bas	Basic program, der er brugt til at lave SinCosTable
MakeSinCosTable.bas.info	
MakeSRCSinCosTable.bas	Basic program, der er brugt til at lave SinCosTable.s
MakeSRCSinCosTable.bas.info	
C (dir)	
Copy	
Delete	
Rename	
List	
Dir	
Type	
S (dir)	
startup-sequence	Der kan bootes direkte fra disketten.
DEVS (dir)	
system-configuration	Denne sekvens laver 80-tegns skærmbredde, for at give et pænt skærmbillede
Execs (dir)	Indeholder assemblerede filer
Anim1	Animationseksempel
Dots1	3D-demo med Joystick styring
Kube1	Opbygning af en terning
Obj1B	Terning og pyramide på 32-farvers baggrund
Obj1Nb	Samme som Obj1b, men på ensfarvet baggrund
Obj2Ef	Bevægelse med jævne overgange
StarFSide	Stjerne med fantomflade
StarFSub	Stjerne med fantomobjekt
Sources (dir)	Sourcekoderne til de assemblerede filer i execs
Anim1.s	
Dots1.s	
Kube1.s	
Obj.s	Sourcekode til rutiner, der bruges i andre programmer
Obj1B.s	
Obj1Nb.s	
Obj2Ef.s	
Poly.s	Sourcekode til udfyldning, beskæring og kopiering
Skeleton.s	Sourcekode med en grundstruktur, som andre programeksempler er bygget over
StarFSide.s	
StarFSub.s	

Includes (dir)	
Custom.i	Liste over hardware-registre
Line.s	Linietegningsrutiner. Indeholder rutinerne: draw_line Tegner en linie draw_Xline Tegner en linie uden den første prik init_line Sætter blitteren i linie-tegningsmode
Rotate.s	Rotation om 3 akser (sourcekode)
sincostable	Sinus-/Cosinustabel (binær fil)
SinCosTable.s	Sourcekoden til sincostable
SEKA (dir)	Indeholder progameksemplerne skrevet til K-SEKA
Sources (dir)	
Anim1.s	
Dots1.s	
Kube1.s	
Obj.s	
Obj1B.s	
Obj1Nb.s	
Obj2Ef.s	
Poly.s	
Skeleton.s	
StarFSide.s	
StarFSub.s	
Includes (dir)	
Custom.i	
Line.s	
Rotate.s	
SinCosTable	
SinCosTable.s	

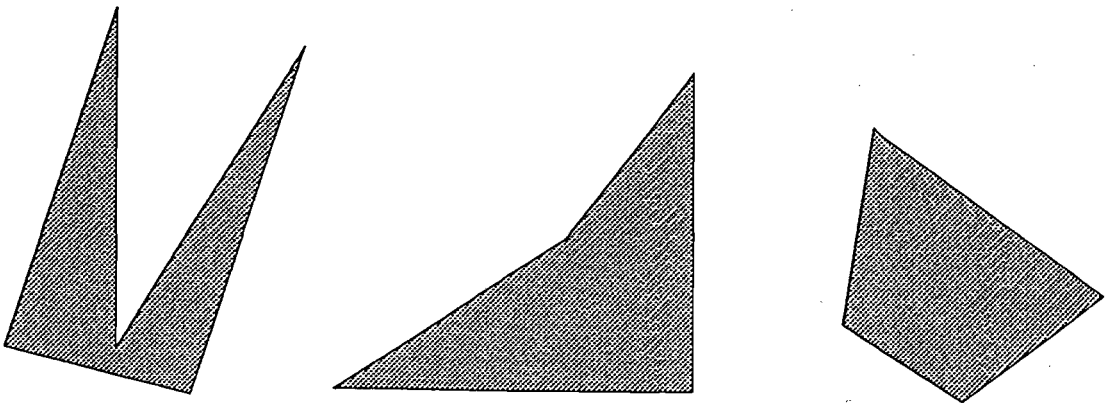
OPGAVE 1



1 2 3 4



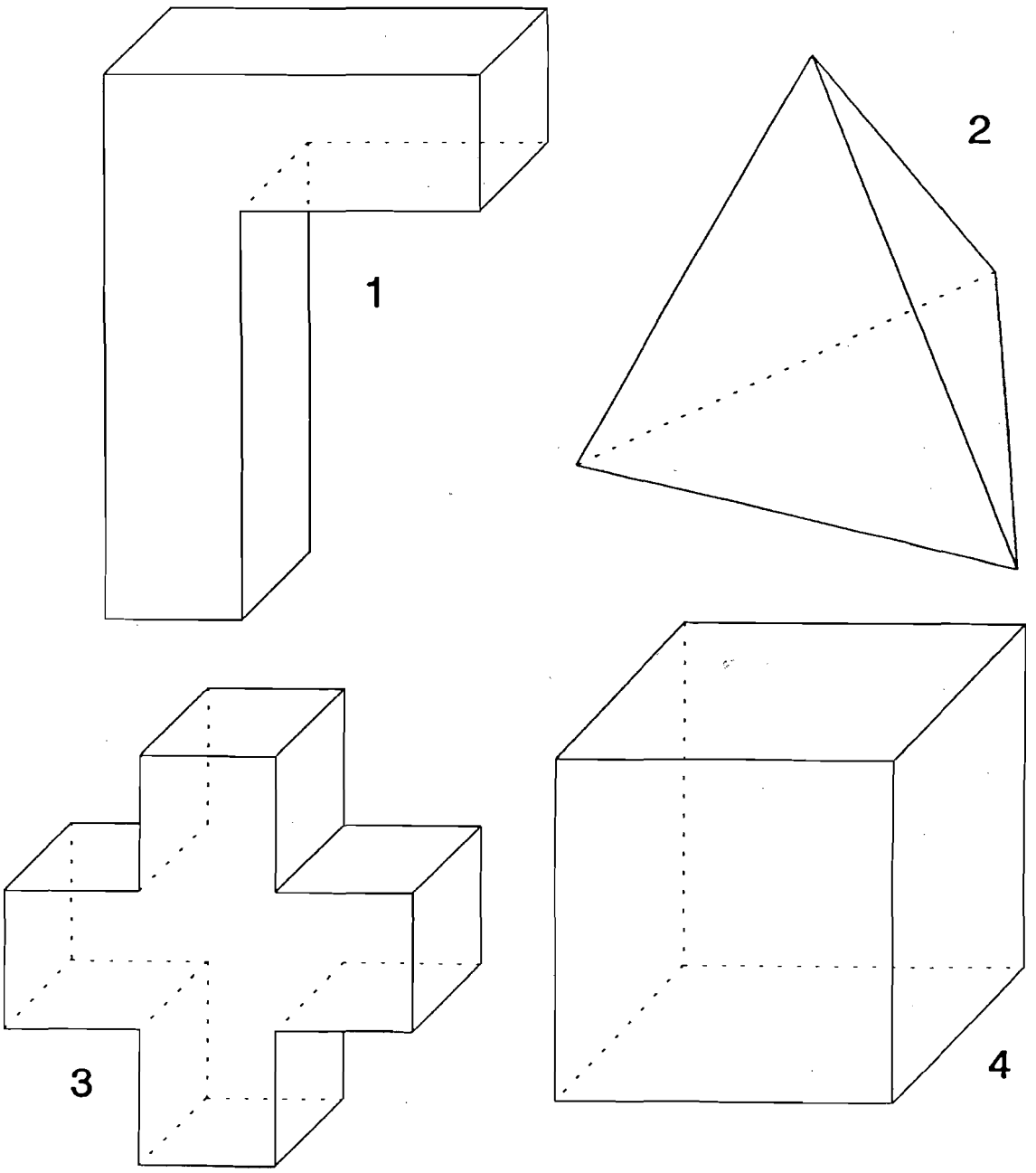
5 6 7



8 9 10

Hvilke af disse objekter er KONVEKSE?

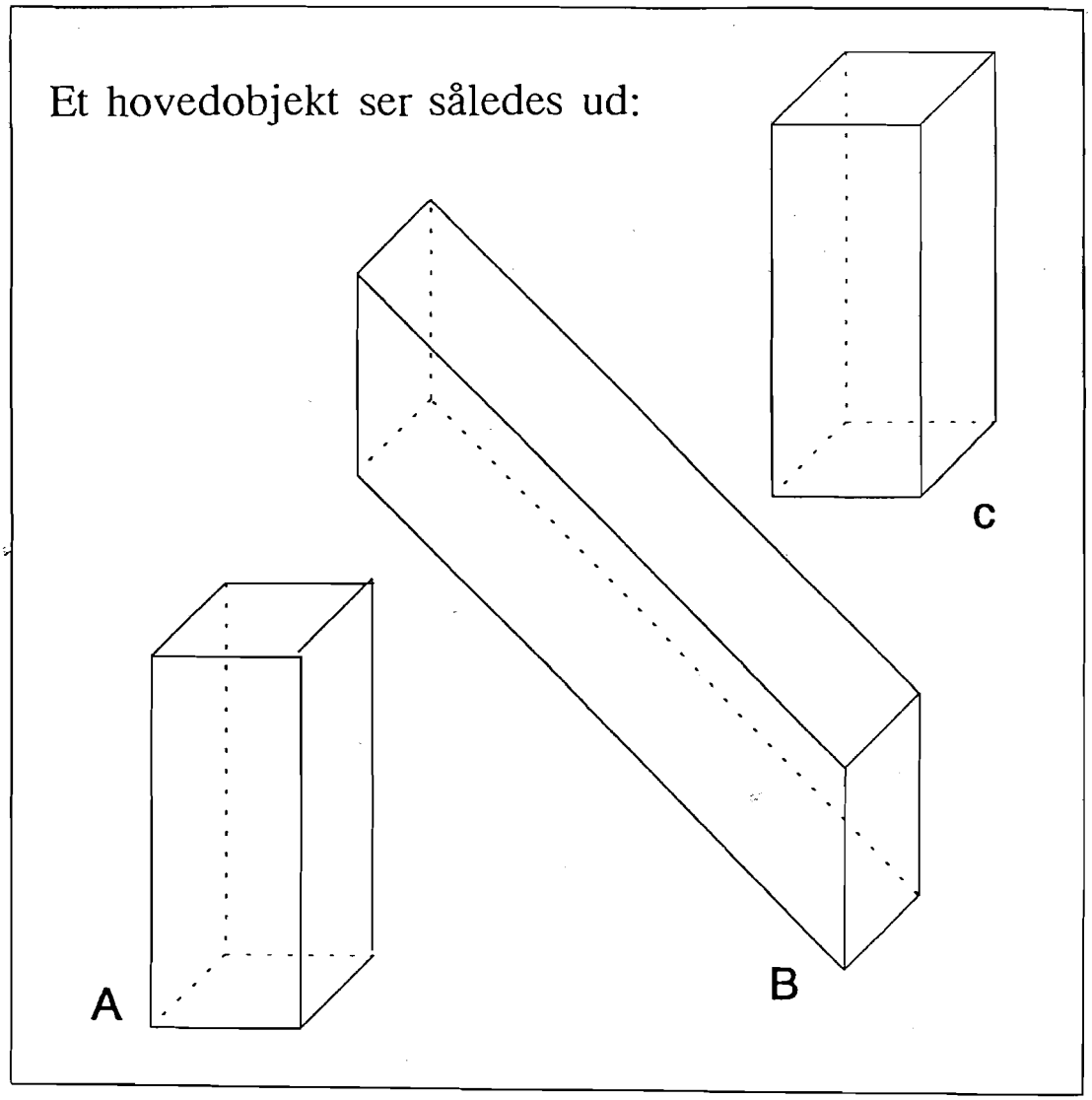
OPGAVE 2



Hvilke af disse objekter kan være et underobjekt?

Hvordan vil du opdele de andre for at få færrest mulige underobjekter?

OPGAVE 3



Objekterne er navngivet A, B og C.

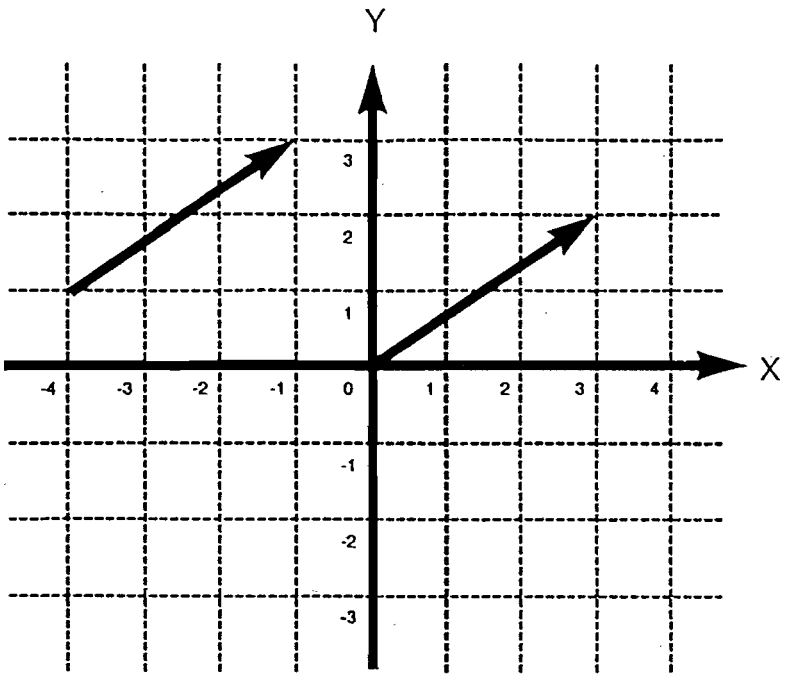
Hvilket (eller hvilke) af underobjekterne kan være underobjekt 0 i hovedobjektet?

Tænk dig grundigt om!

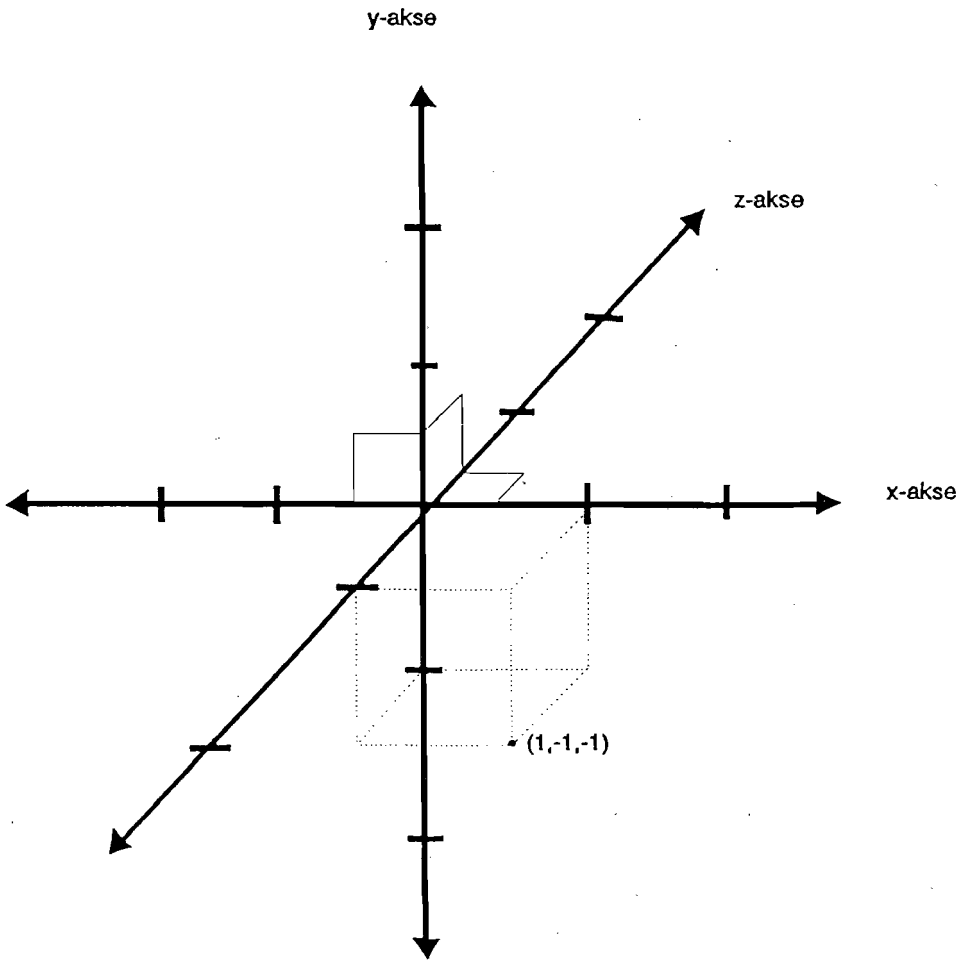
Gennemgå checklisterne, både for hoved- og underobjekterne.

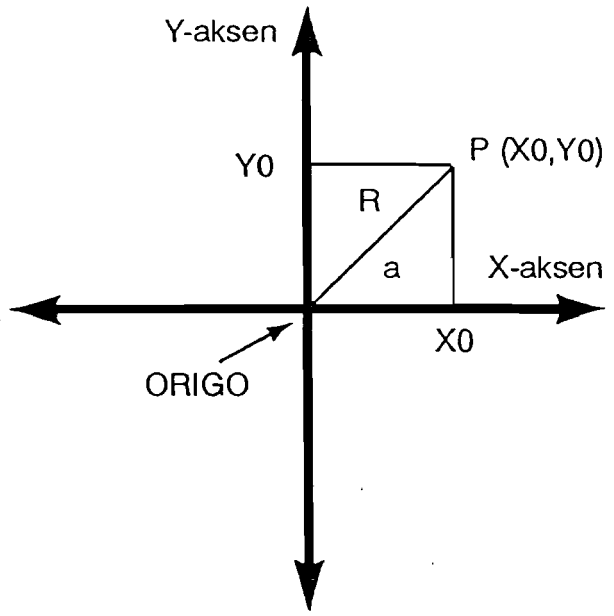
Giv en begrundelse for dit svar.

Figur 1A



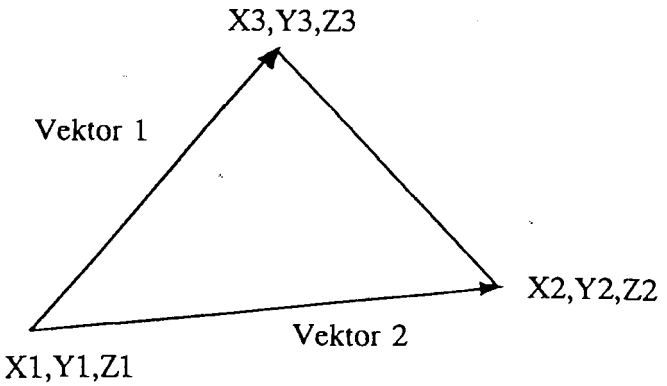
Figur 1B



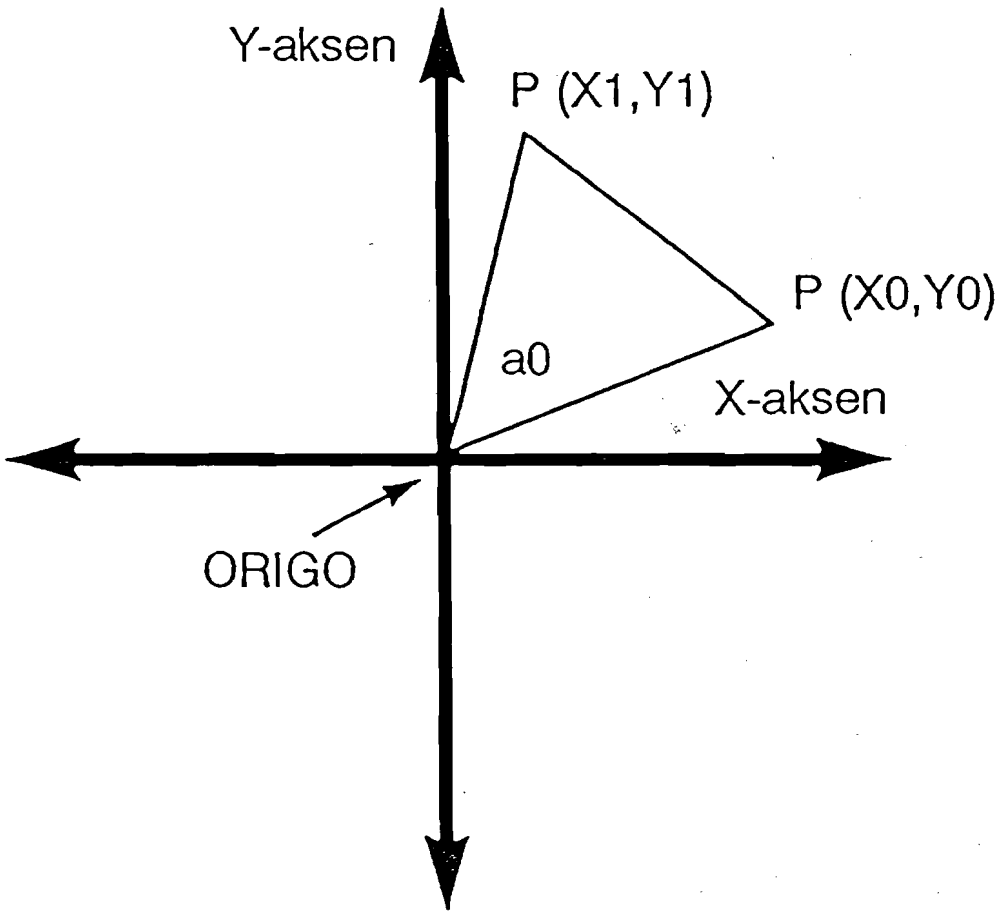


Figur 2

Figur 3

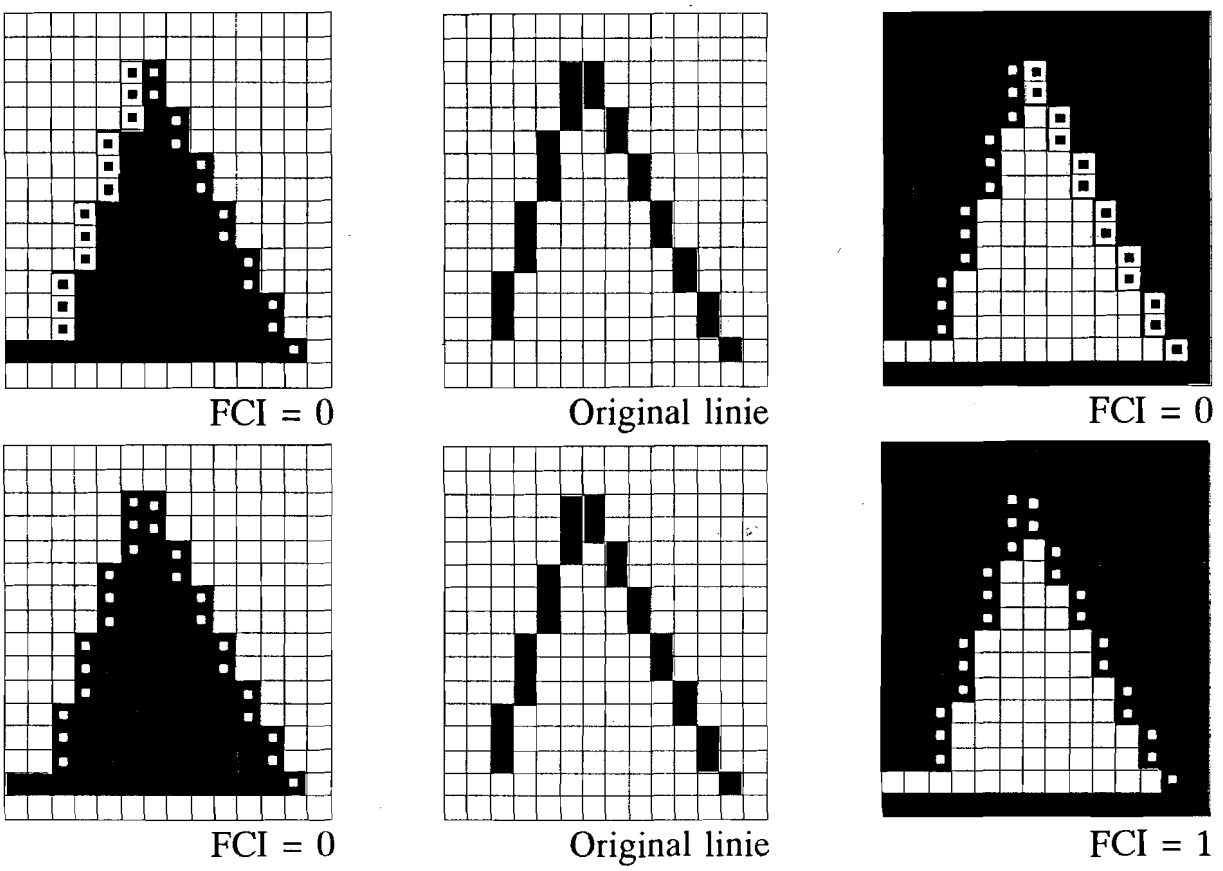


Figur 4



Figur 5

Blitterens to udfyldnings-modes



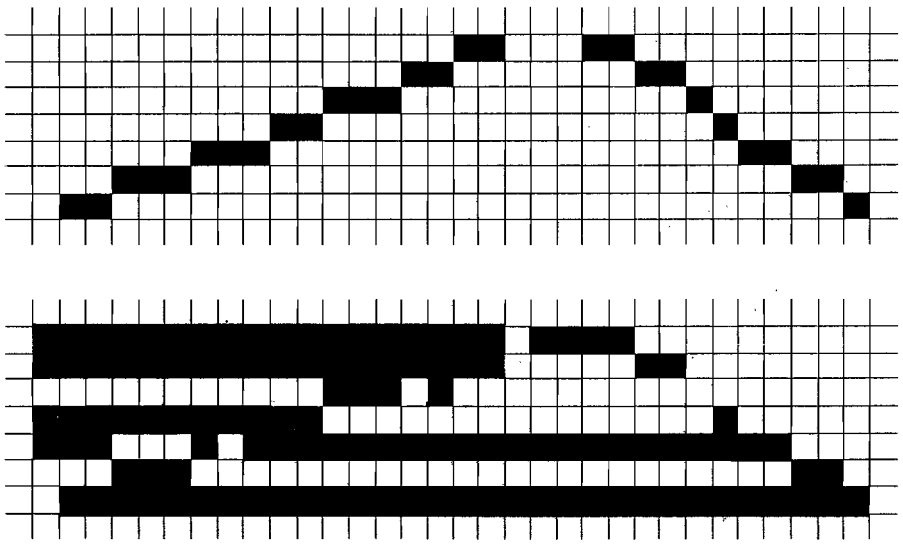
De tre øverste figurer i EFE og de tre nederste er i IFE

Her ser du resultatet af de 4 forskellige måder, der kan udfyldes på.

Hver firkant repræsenterer en pixel. De små firkanter inde i nogle af ruderne viser, hvor linierne gik før udfyldningen.

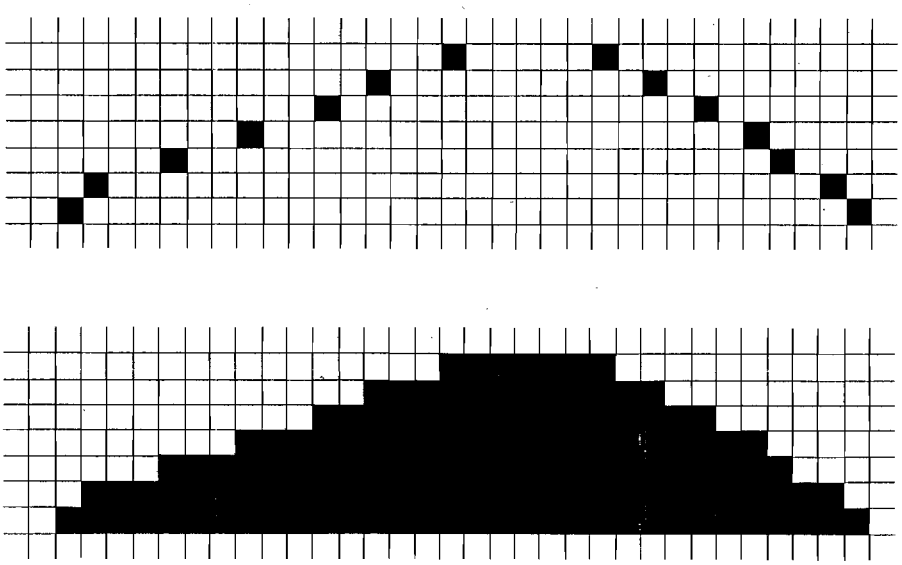
Læg også mærke til, at FCI-biten indlæses igen for hver linie, sådan at fejlen ved de manglende pixels minimeres.

Blitterens to linietyper



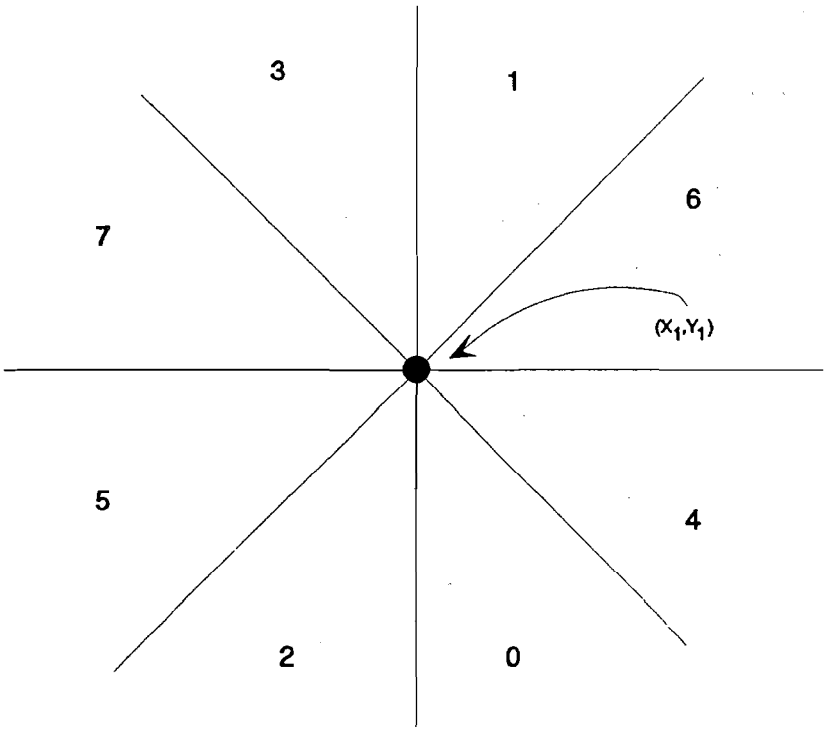
Figur 6A

Dette er en normal sammenhængende linie. Som du kan se, giver det grimme fejl, når objektet udfyldes.

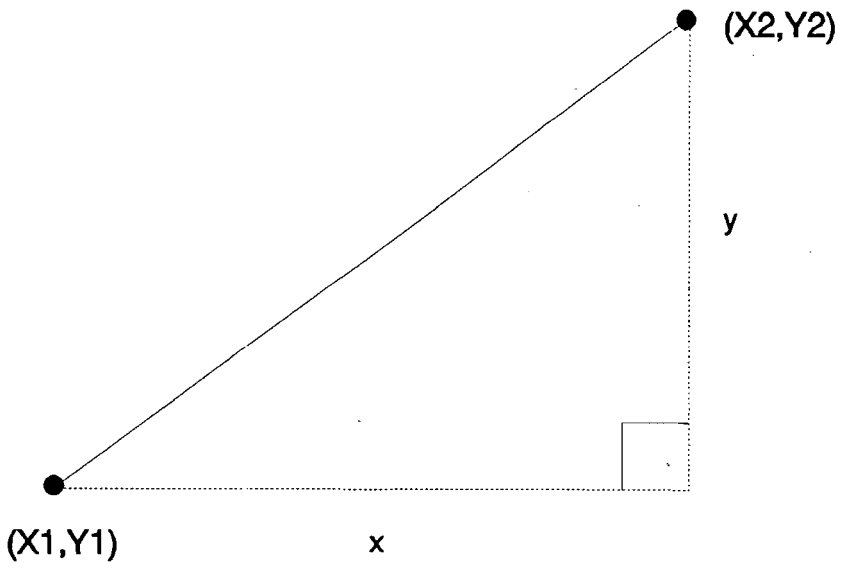


Figur 6B

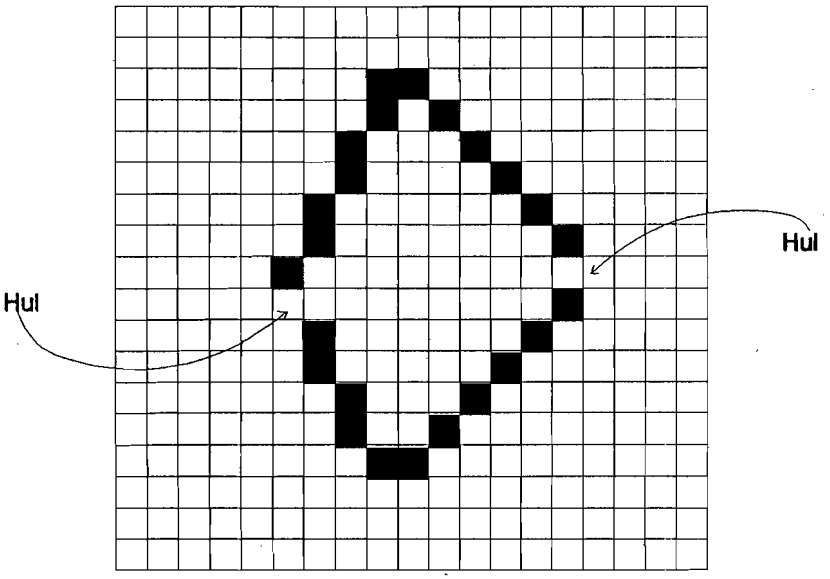
Dette er en speciel linietype, hvor man kun skal tegne én pixel pr horisontal linie. Dermed fungerer udfyldningen perfekt. Denne linietype bruges kun til linier rundt om et objekt, som skal udfyldes med farve.



Figur 7

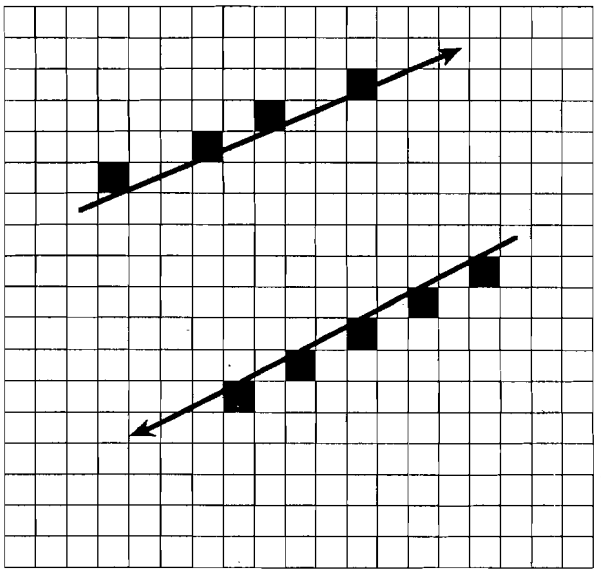


Figur 8



Figur 9

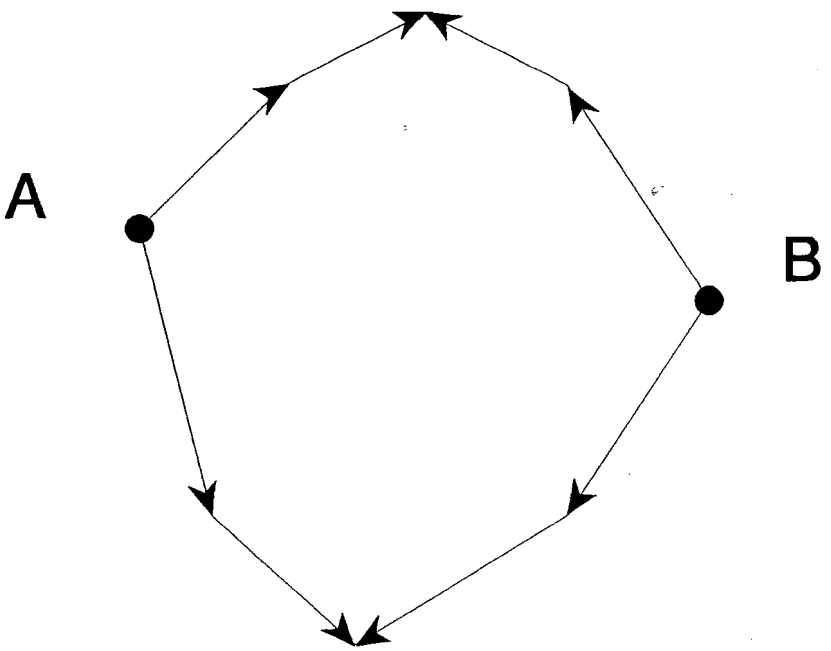
Der opstår grimme huller i nogle af hjørnerne, hvis ikke der bruges specielle linier.



Figur 10

En linie, der tegnes opad mod højre, kommer længere til venstre end en, der tegnes nedad mod venstre.

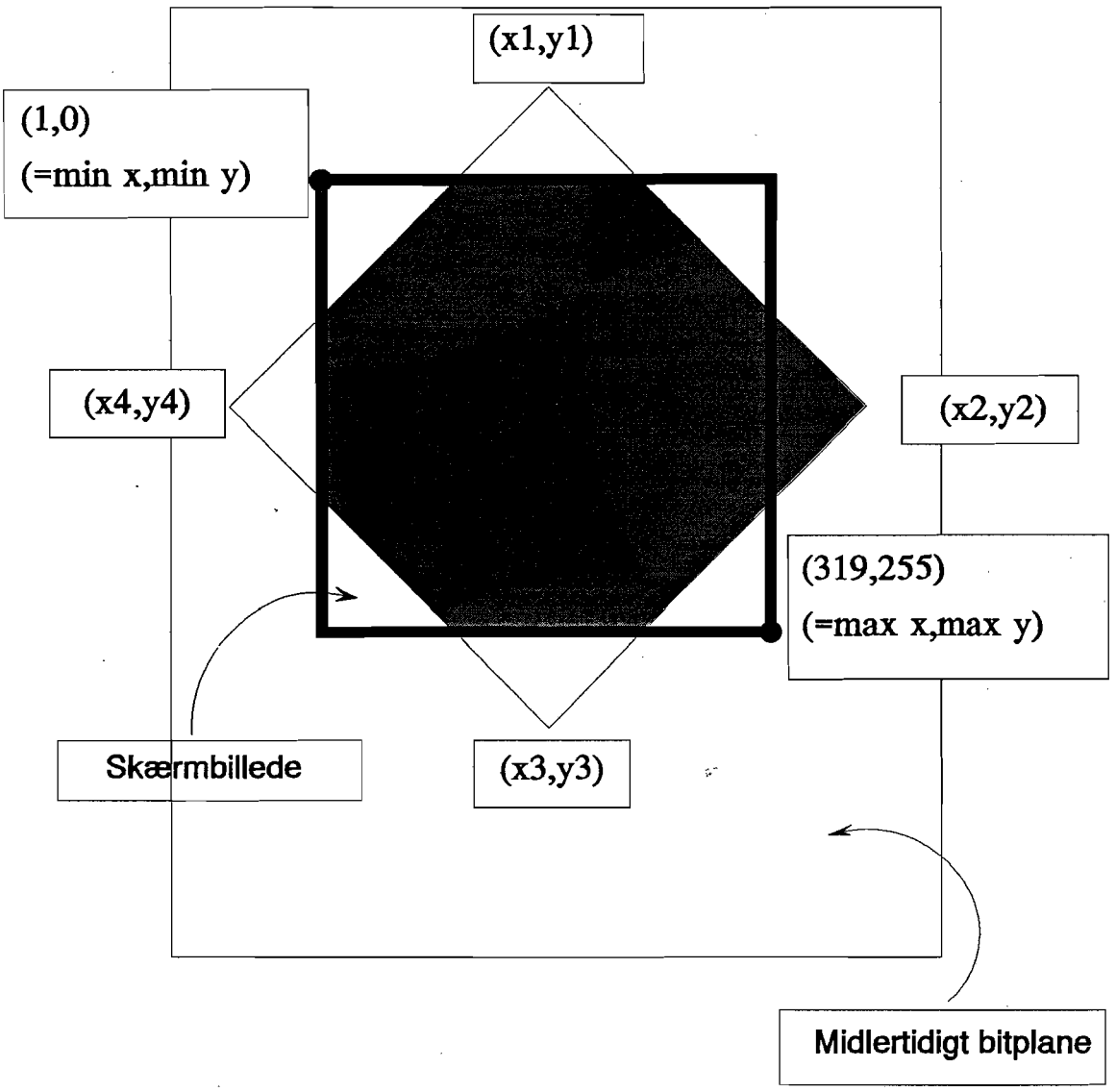
Figur 11



Hvis linierne tegnes rundt om objektet på denne måde, bliver polygonet så stort som muligt. Dermed undgås huller mellem fladerne.

KLIPNING

Figur 12



Her har vi sat min x, max x, min y og max y til en lo-res skærm (320 x 256 pixels).

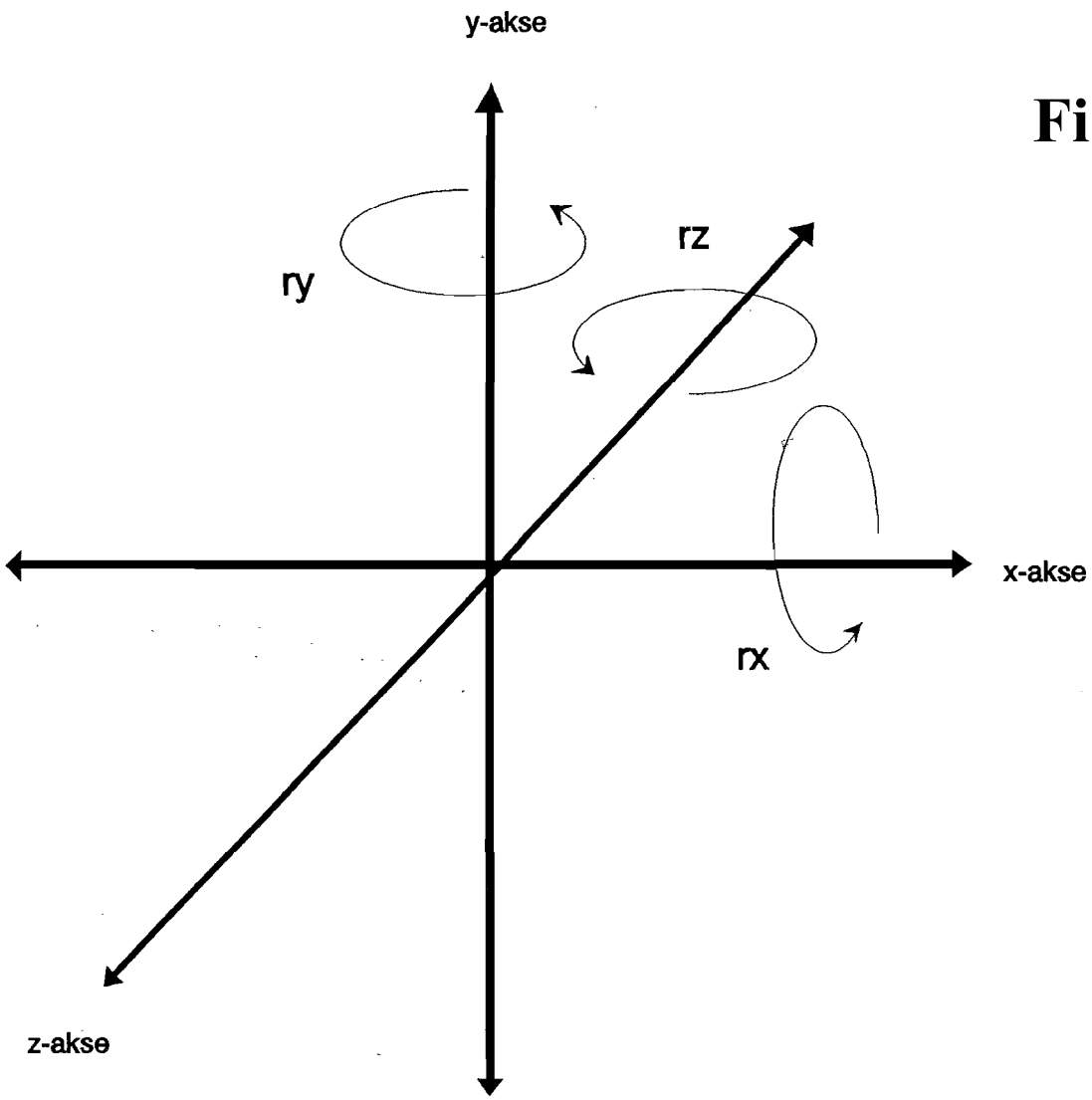
På den venstre side skal (min x - x4 + 15)>>4 words klippes væk.

Øverst skal (min y - y1) linier klippes væk.

Nederst skal (y3 - max y) linier klippes væk.

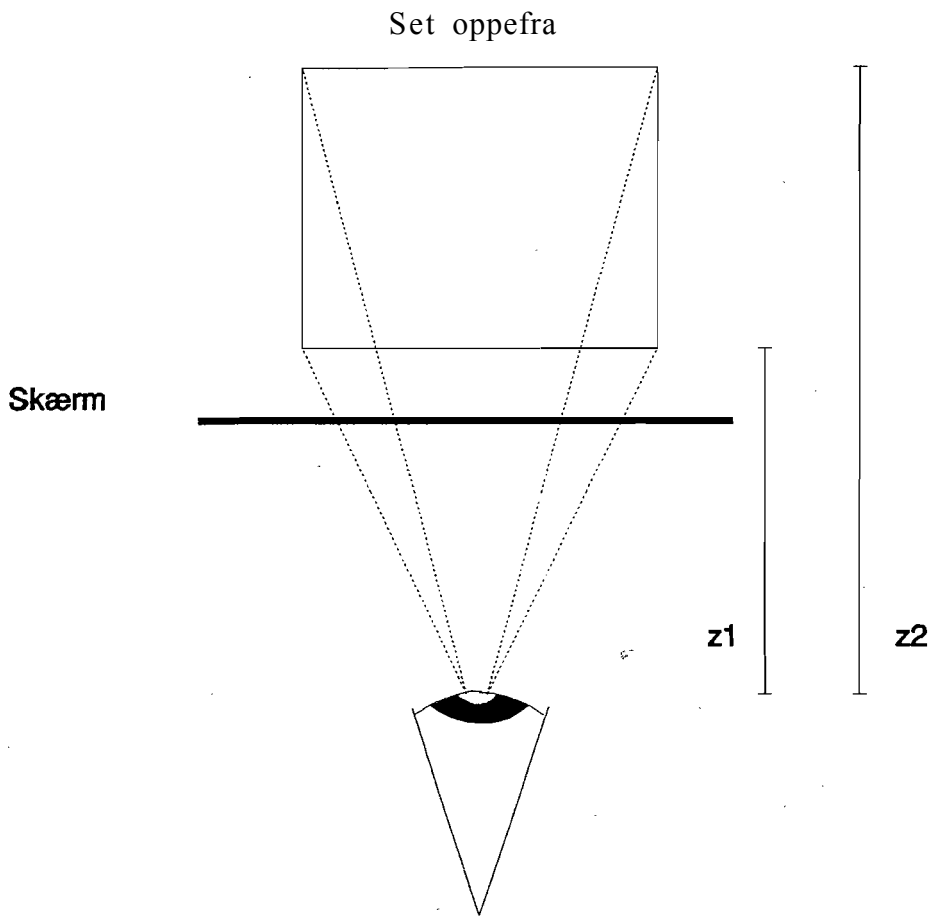
På den højre side skal (x2 - max x + 15)>>4 words klippes væk.

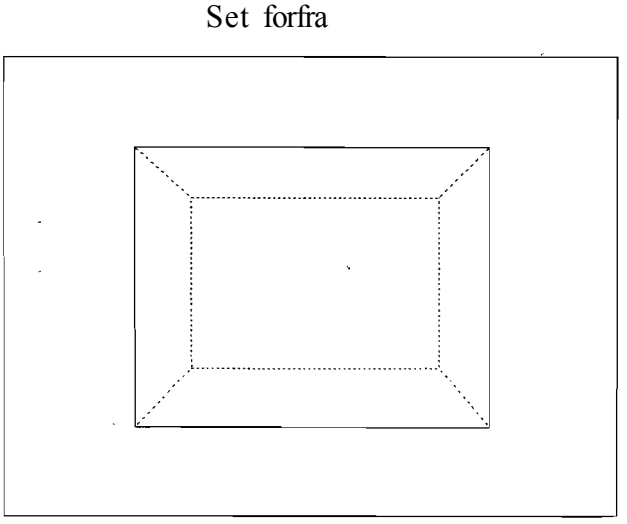
Det er kun det skraverede område af figuren, der skal udfyldes. De områder, der ikke udfyldes, kopieres ikke til skærmen.



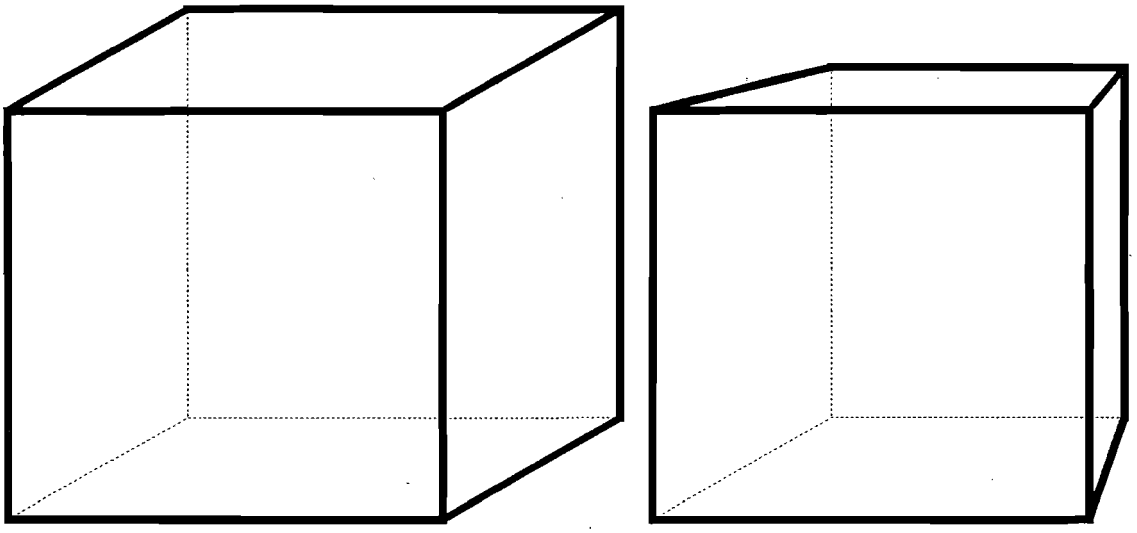
Figur 14

Figur 15





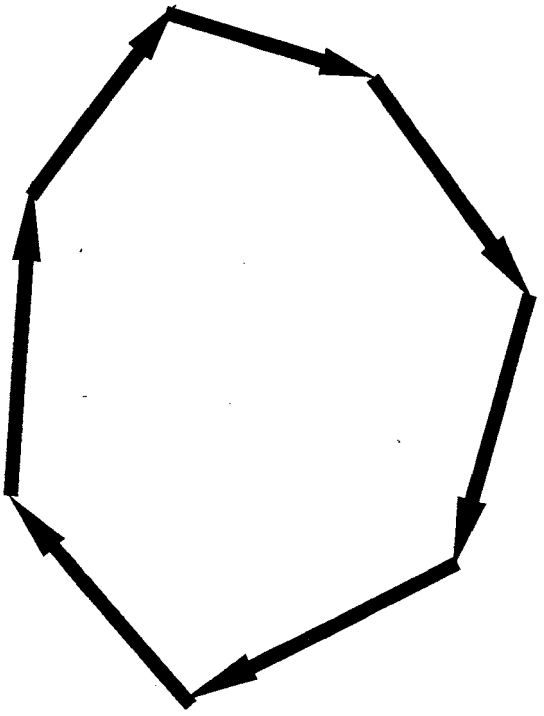
Figur 16



Terning (kube) tegnet uden perspektiv

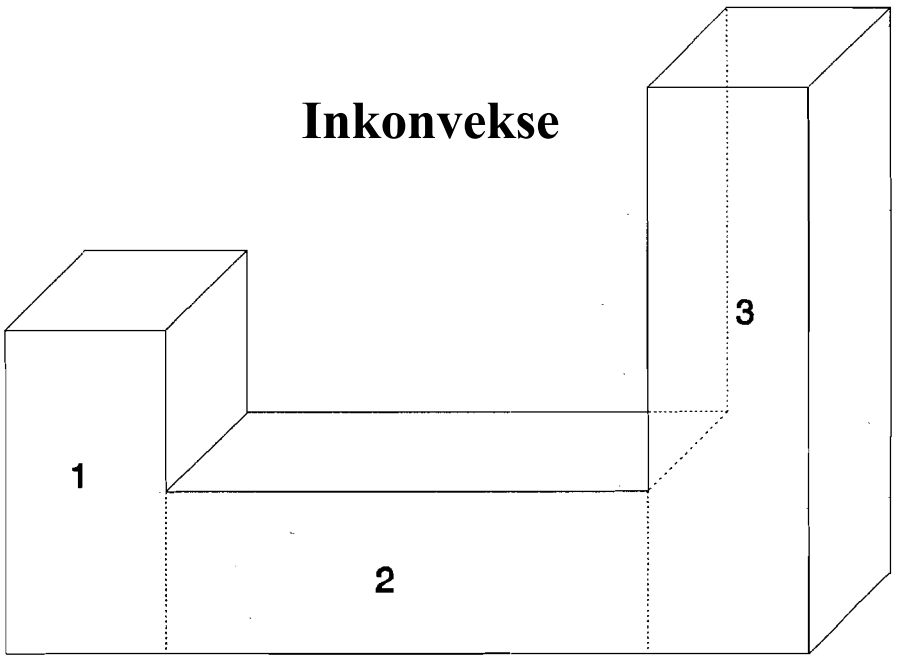
Terning (kube) tegnet rperspektiv

Figur 17

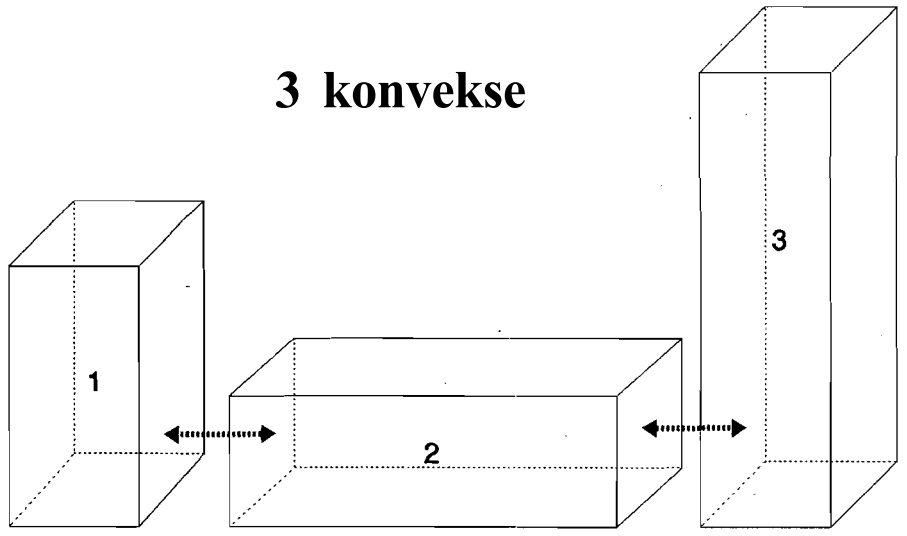


I urets retning (eller "clockwise")

Figur 18A

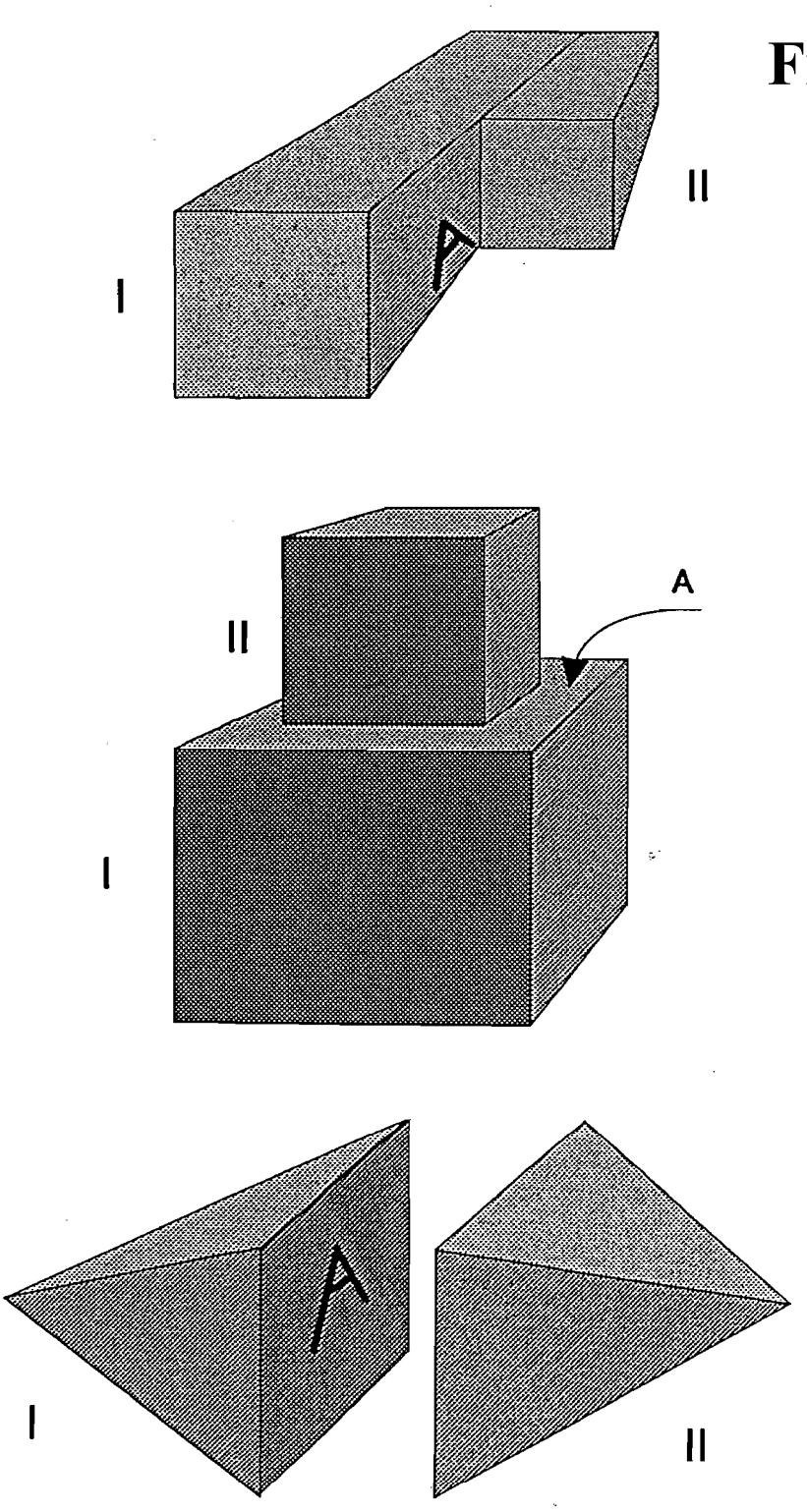


Figur 18B



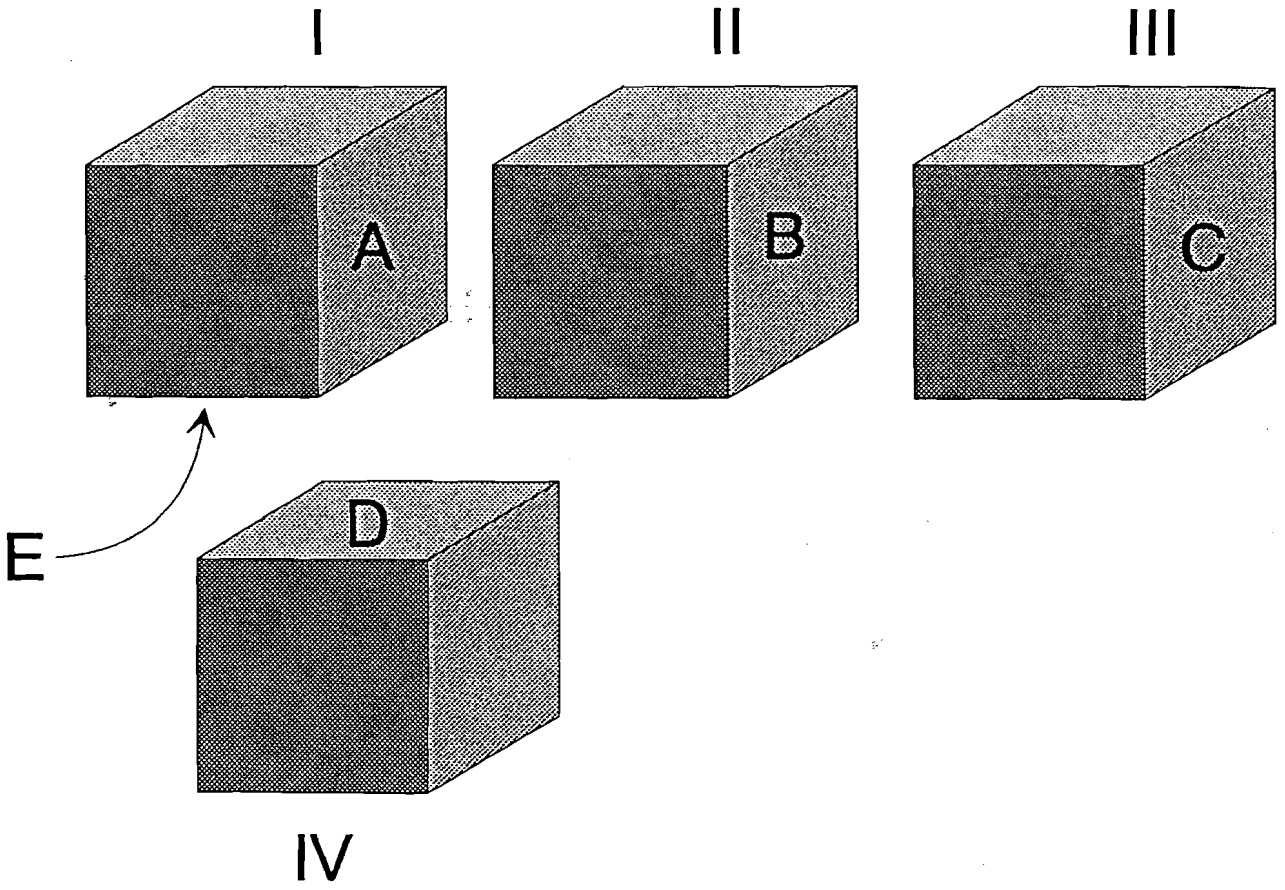
figur b samles til figur a

Figur 19



Her er tre forskellige objekter. For alle gælder, at når sideflade A er synlig, kommer objekt I foran objekt II

Figur 20



Ovenfor er der vist et objekt, som består af 4 underobjekter.

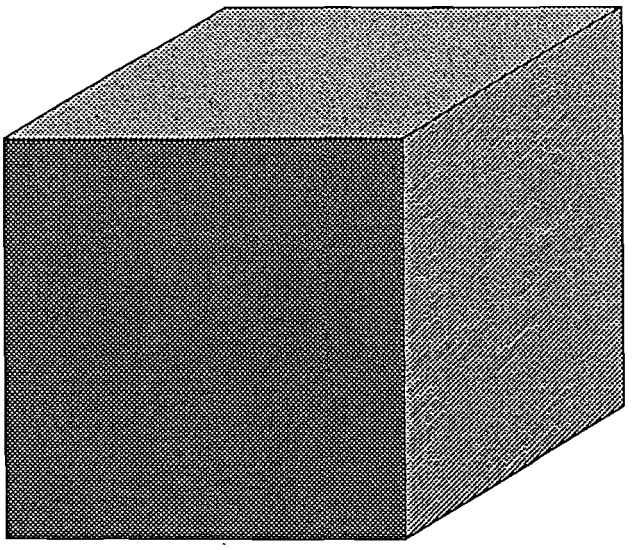
Underobjekt II og III ligger helt til højre for flade A.

Underobjekt III ligger også helt til højre for flade B.

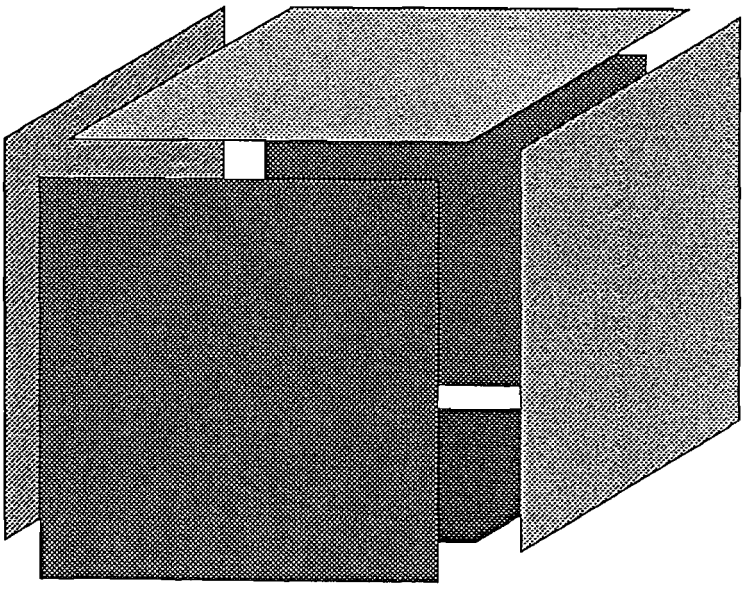
Underobjekt IV kan ikke sorteres ved hjælp af disse flader. Derimod ligger underobjekteme I, II og III helt på oversiden af flade D.

Tilsvarende ligger underobjekt IV helt på undersiden af flade E.

Figur 21A

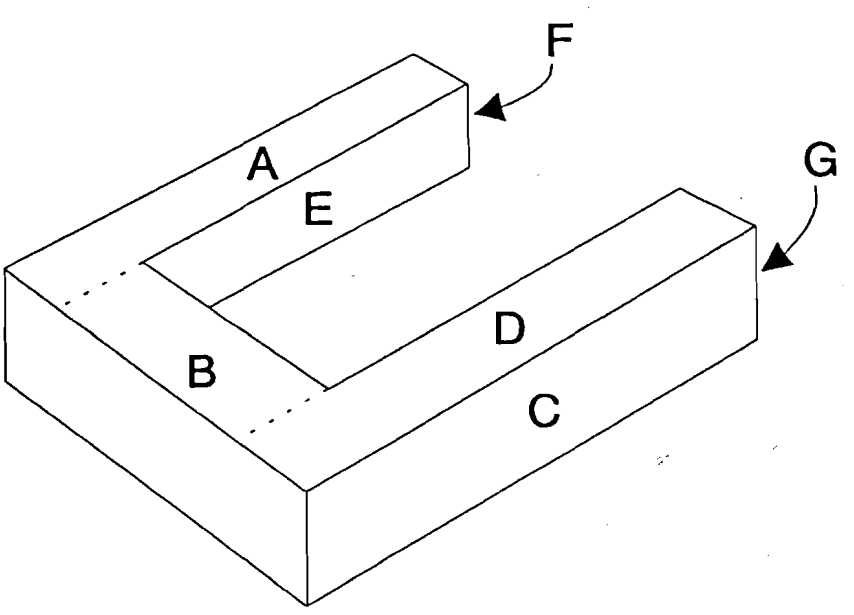


Figur 21B



Figur 22

POLYGON I SAMME PLAN

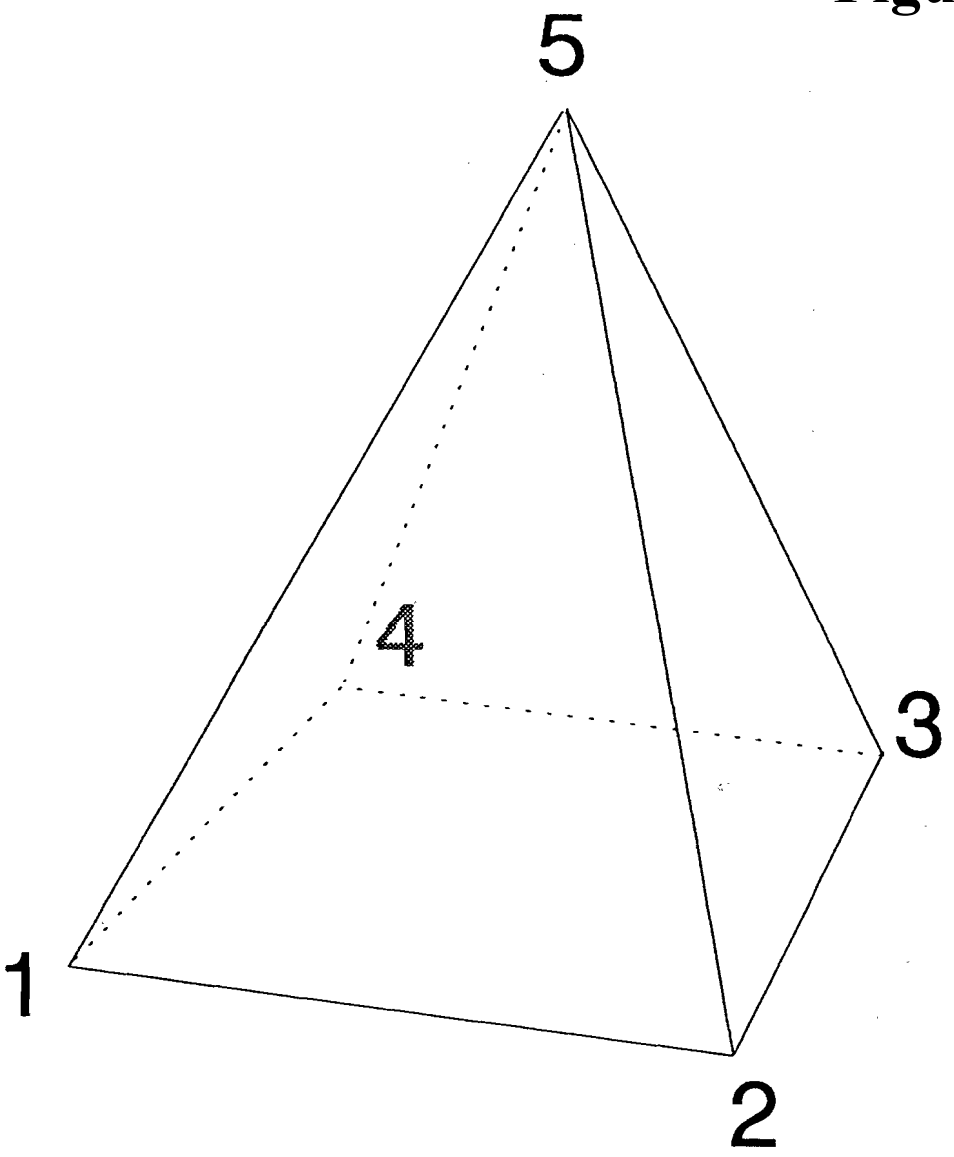


A, B og D er i samme plan.

F og G er i samme plan.

E og C er parallelle, men IKKE i samme plan!

Figur 23

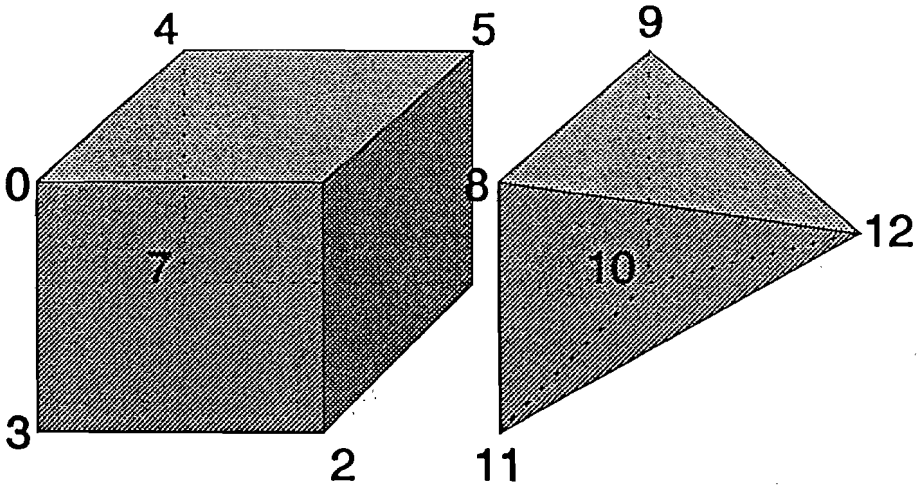


Når du kun opgiver koordinaterne til hver enkelt sideflade, er det meget vigtigt at angive dem i den rigtige rækkefølge.

Figurens hjørner skal - for at figuren skal kunne tegnes - have sine hjørner mærket i den rigtige rækkefølge - f.eks i urets retning.

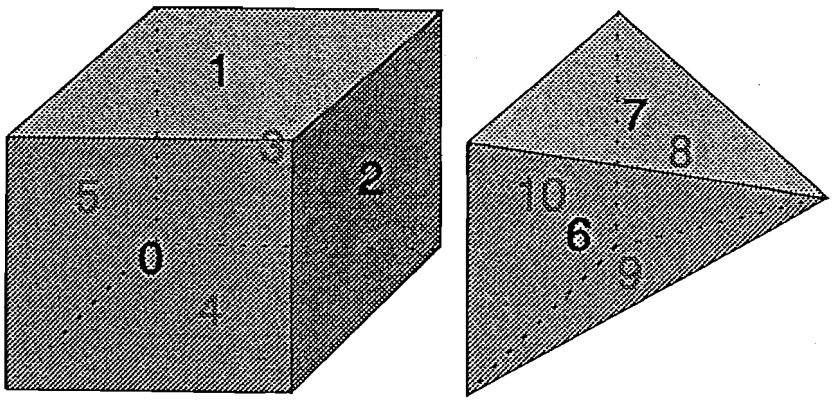
Koordinaterne for hjørnerne skal angives i urets retning når du ser på hver enkelt flade fra ydersiden af objektet. I denne pyramide bliver den rigtige rækkefølge for hjørnerne: 1-5-2, 2-5-3, 3-5-4, 4-5-1 og 1-2-3-4.

Figur 24



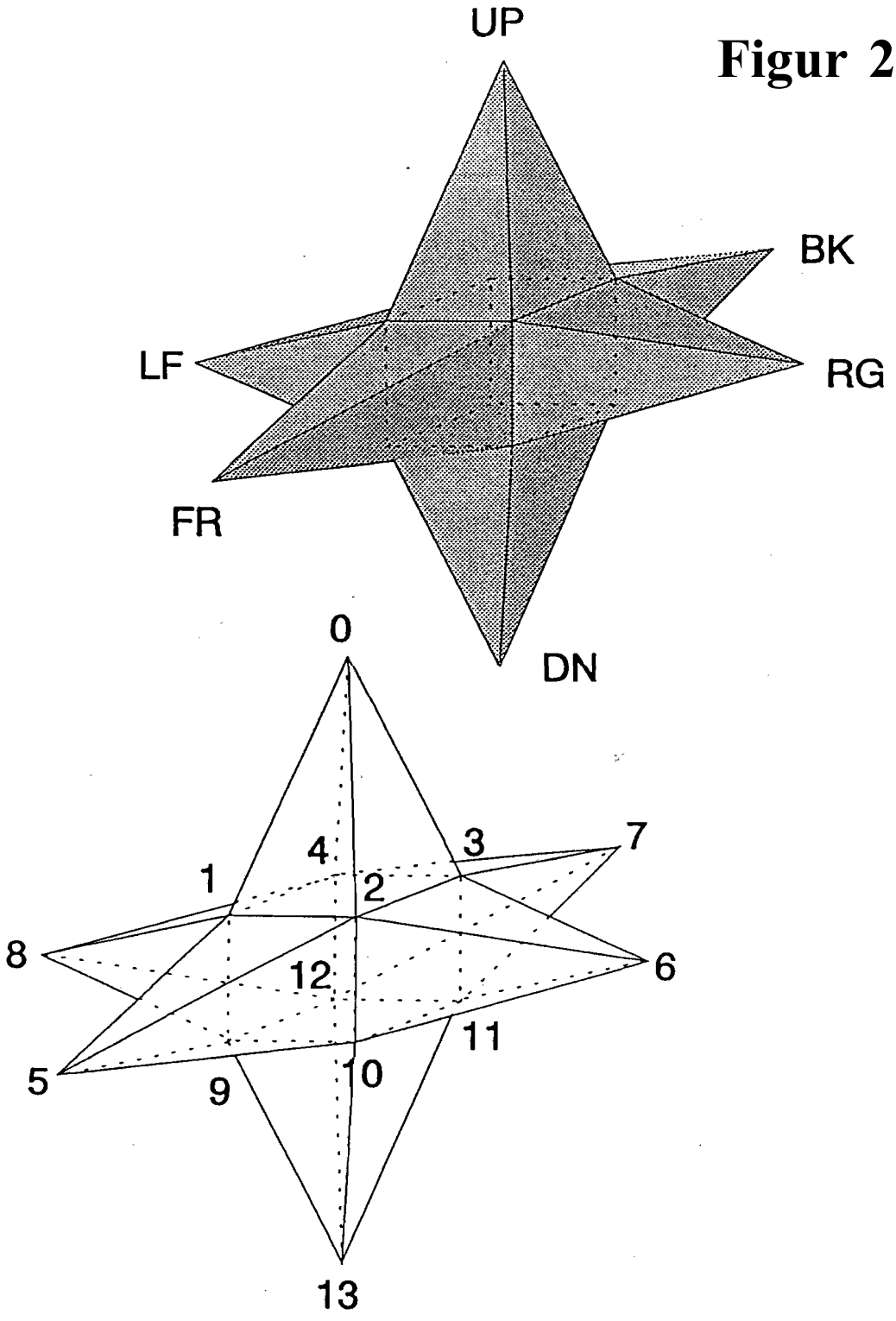
HJØRNERNES KOORDINATER

0	(-100,-50,-50)	7:	(-100,50,50)
1	(0,-50,-50)	8:	(50,-50,-50)
2	(0,50,-50)	9:	(50,-50,50)
3	(-100,50,-50)	10:	(50,50,50)
4.	(-100,-50,50)	11:	(50,50,-50)
5	(0,-50,50)	12:	(100,0,0)
6:	(0,50,50)		



I denne figur angiver tallene de ulige fladers indbyrdes nummer.

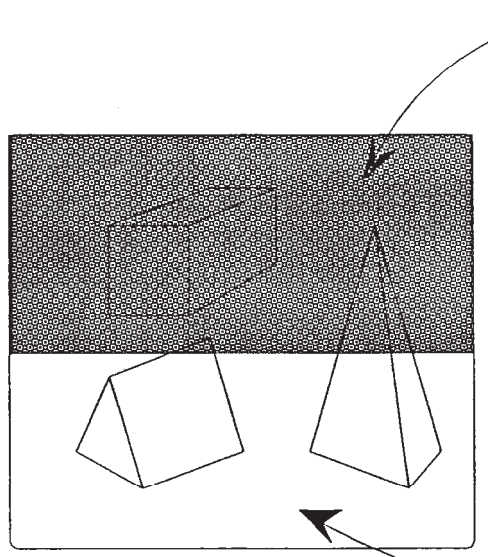
Figur 25



Den øverste figur viser navnene på de enkelte under-objekter. Hver stjernetak er et underobjekt.

Den nederste figur viser koordinatnumrene for de enkelte hjørner.

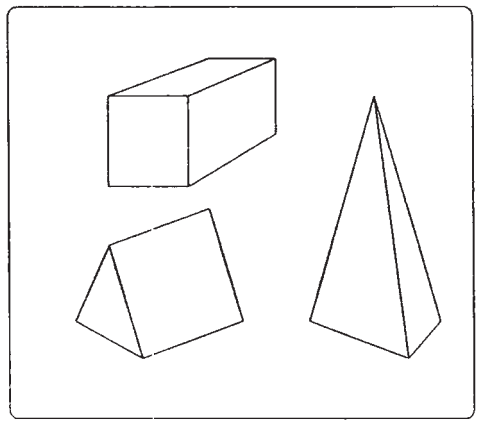
Figur 26



Dette område har elektron-strålen allerede tegnet på monitoren

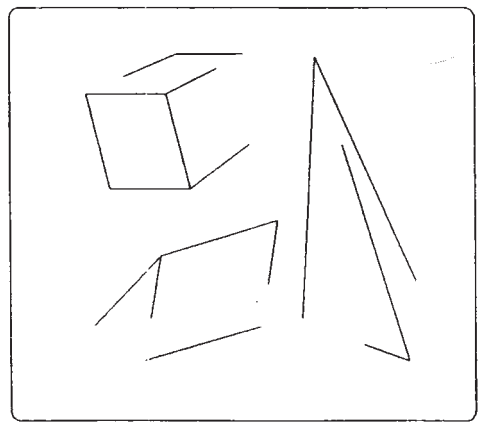
Dette område er endnu ikke tegnet op.

1



Denne skærm er klar til at blive vist, og vil blive tegnet op på monitoren, så snart elektronstrålen er færdig med billede 1.

2



Her er næste skærm ved at blive tegnet op

3

Ved double-buffering skal skærm 2 fjernes. Det er derfor nødvendigt at vente til elektronstrålen er færdig med at tegne billede 1 før vi kan tage fat på billede 1 igen. Det forsinker opdateringen af skærmen....