CODEN:LUTFD2/(TFRT-5280)/1-067/(1982)

TREDIMENSIONELL DATORGRAFIK OCH ANIMERING AV INDUSTRIROBOT

MIKAEL RIGNELL

INSTITUTIONEN FÖR REGLERTEKNIK
LUNDS TEKNISKA HÖGSKOLA

SEPTEMBER 1982

LUND INSTITUTE OF TECHNOLOGY
DEPARTMENT OF AUTOMATIC CONTROL
Box 725
S 220 07 Lund 7 Sweden

Document name
Report

Date of issue
September 1982

Document number
LUTFD2/(TFRT-5280)/1-067/(1982)

Author(s)
Mikael Rignell

Supervisor
Hilding Elmqvist

Sponsoring organization

Title and subtitle

Three dimensionel computer graphics and animation of an industrial robot

(Tre dimensionell datorgrafik och animering av industrirobot.)

Abstract

A three dimensionel graphics package has been developed. It encludes scan conversion of polyqous, hidden surface removal, shading and transformation of coordinate systems. It has been used for animation of an industrial robot. The movement of the robot is specified in a cartesian coordinate system using jousticks.

Key words

Computer graphics, robotics

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
Swedish

Number of pages
67

Security classification

Recipient's notes

Distribution: The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Telex: 33248 lubbis lund.

DOKUMENTDATABLAD RT 3/81

Tredimensionell datorgrafik och animering av industrirobot.

Examensarbete av

Mikael Rignell

Handledare:

Hilding Elmqvist

Institutionen för Reglerteknik, LTH

Innehåll

1. INLEDNING
2. GRUNDER I DATORGRAFIK
3. TREDIMENSIONELL GRAFIK
4. ELIMINERING AV DOLDA YTOR
5. UPPBYGGNAD AV GRAFISKA MODELLER
6. ANIMERING AV ROBOT
7. BESKRIVNING AV HUVUDPROGRAM
8. SAMMANFATTNING
9. REFERENSER
Appendix 1 Programlistning
Appendix 2 Beräkningar till kap 6

1. INLEDNING

Färgbildsgrafik är ett område som utvecklas alltmer och man finner ständigt nya användningsområden. Ett viktigt område är operatörskommunikation, där stora manöverpaneler kan ersättas med en terminal med tillhörande färgbildsmonitor. Ett annat exempel på tillämpning är i flygsimulatorer. CAD-CAM är ett välkänt begrepp. Också här tar man hjälp av färgbildsgrafik. Slutligen kan nämnas de otal TV-spel som finns i marknaden.

Avsikten med det här examensarbetet har varit att studera några av de möjligheter färgbildsgrafik ger. Första steget var att med hjälp av det interface som fanns, kunna rita bilder med djupverkan ("tredimensionella bilder"). För att få mer realism i bilden måste dolda ytor tas bort. Detta sker med en s.k. hiddensurface-algoritm. Slutligen studerades animering med specialtillämpningen industrirobot. Här lades särskild vikt vid att kunna flytta robotarmen i ett kartesiskt koordinatsystem, till en godtycklig punkt och med en godtycklig riktning på armens spets.

Jag vill framföra ett stort tack till Hilding Elmqvist, min med oändligt tålamod utrustade handledare.

2. GRUNDER I FÄRGBILDSGRAFIK

Datorgrafik kan uppdelas i passiv datorgrafik, där man ritar upp en bild på skärmen och för övrigt inte kan påverka dess utseende, och i interaktiv datorgrafik, där observatören i större eller mindre utsträckning kan ändra bilden. Exempel på passiv datorgrafik är uppritandet av detaljerade och verklighetstrogna bilder, eller om man så vill "datorkonst". Beräkningarna för att rita upp dessa bilder tar i allmänhet lång tid. Vid interaktiv datorgrafik får man i regel nöja sig med enklare bilder för att inte få för långsam ändring i bilden. Men i t.ex. en flygsimulator får man en naturlig rörelse även i en avancerad bild.

Ett modernt system för visning av en bild består av ett digitalt bildminne (frame buffer), en TV-monitor och ett interface (display controller). I bildminnet lagras information om varje pixels* tillstånd i en matris. Interfacet överför innehållet i frame buffer till skärmen. I vårt fall arbetar vi med två bildplan (två frame buffers), ett främre och ett bakre. På så sätt kan man hela tiden visa en fullständig bild på skärmen, genom att rita i det bakre planet och sedan byta plats på de båda bildplanen.

För att kunna plotta punkterna behövs ett koordinatsystem, där varje punkt har en x-koordinat och en y-koordinat. Dessa koordinater är heltal. I det aktuella fallet har skärmen 512x512 pixel. Följaktligen går x- och y-koordinaterna från 0 till 511. Skärmens koordinatsystem framgår av fig 1.

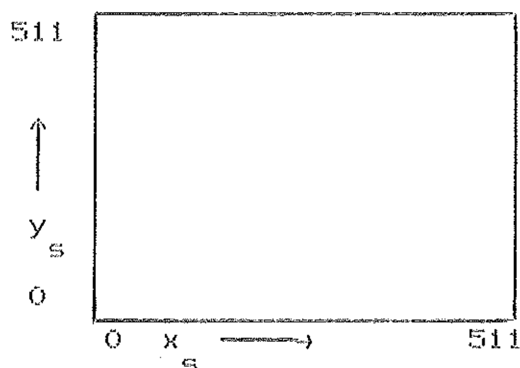


fig 1

När man skall rita en bild på skärmen, är det lämpligt att ha en procedur för att rita ut en linje så man slipper behandla en pixel i taget. (Kurvor kan approximeras med en följd av korta, räta linjer). P.g.a. att skärmen har ändlig upplösning (ändligt antal pixel), kan inte andra linjer än lodräta och vågräta visas korrekt på skärmen. I övriga fall

(*) Pixel = det minsta bildelementet eller "punkt på skärmen".

hamnar en del av pixeln vid sidan av linjen. Det finns flera olika algoritmer för att bestämma vilka punkter som skall sättas när en rät linje skall skrivas ut och för den intresserade hänvisas till Newman&Sproull. Numera har denna linjedragning ofta realiserats i hårdvaran.

Hårdvaruinterface

```
procedure CmiCol(col:integer);
            anger den färg pixeln skall ha
procedure CmiDot(x,y:integer);
            "tänder" pixeln med koordinaten (x,y)
procedure PixelLine(xpixelold, ypixelold,
                    xpixel, ypixel: integer);
            Drar en linje från (xpixelold, ypixelold)
             till (xpixel, ypixel)
procedure TwoPlanes;
            två bildplan används
procedure DrawBackground;
            rita i det bakre bildplanet
procedure DrawForeground;
            rita i det främre bildplanet
procedure SwapPLane;
            främre och bakre bildplanet byter plats
procedure CmiErase;
            suddar det bildplan man ritar i
procedure JoySticks(var values:joysticktype);
            ger styrspakarnas värde
procedure PrintExactScreen;
            ger en papperskopia av bilden på skärmen
procedure BlueShade;
            anger att ett anrop till CmiDot ger en blå nyans
```

3. TREDIMENSIONELL GRAFIK

Ofta behöver man kunna rita ut tredimensionella föremål och scener. Då är nyckelordet realism. Första steget är att ta tillvara den djupinformation som finns i beskrivningen av objektet. Detta görs genom en perspektivtransformation av det tredimensionella föremålet på den tvådimensionella skärmen. I perspektivtransformationen ingår att dividera alla punkters bredd- resp. höjdkoordinater med deras resp. djupkoordinater.

Ett stort steg mot realism är att ta bort dolda ytor så att bara de som syns i verkligheten återfinns på skärmen, s.k. hiddensurface removal (se kap 4). Här kan även begreppet toning (shading) nämnas. Toning innebär man tänker sig att en eller flera ljuskällor belyser föremålet och sedan räknar man ut hur mycket ljus som reflekteras från de olika ytorna och sätter de olika pixelns intensiteter efter detta.

En lämplig byggsten för att bygga upp en tredimensionell modell är polyedern. Dess kantytor utgöres av polygoner som kan beskrivas av en lista på dess hörnpunkter.

3.1 Koordinatsystem

För att beskriva det tredimensionella föremålet, behövs ett koordinatsystem. Det benämns världskoordinatsystemet eller world coordinate system (wcs) och är högerortonormerat. Dess origo, axlarnas riktning samt längdmåttet kan fritt väljas beroende på användning. I vårt fall befinner sig i utgångsläget origo mitt på skärmen samt har koordinataxlarna orienterade enligt fig 2.

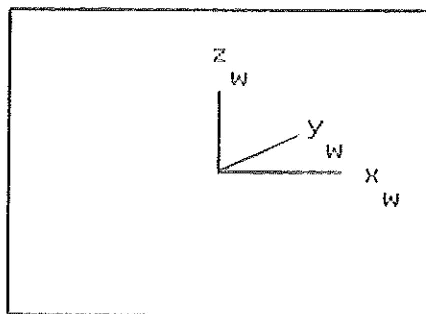


fig 2

3.2 Transformationer

Geometriska transformationer spelar en viktig roll vid genereringen av tredimensionella bilder. De används för att ange olika objekts position och orientering i förhållande till varandra. De används också när man vill flytta den punkt varifrån man ser på objekten. Slutligen används de även vid perspektivtransformation för att projicera det

tredimensionella objektet på den tvådimensionella skärmen.

Varje rörelse kan beskrivas som en kombination av translationer och rotationer. Translationer och rotationer kan representeras av 4x4 matriser. Dessa multipliceras på vektorn som beskriver den aktuella punkten.

Translation

Transformationen som translaterar en punkt (x,y,z) till en annan punkt (x',y',z') är:

$$[x' \quad y' \quad z' \quad 1] = [x \quad y \quad z \quad 1]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Tx & Ty & Tz & 1 \end{bmatrix} \qquad (1)$$

Tx, Ty, Tz är translationens komponenter i x-, y- resp. z-riktningen. Den fjärde koordinaten (ettan) gör beskrivningen av translationen enkel.

Rotation

Rotation kring x-axeln beskrivs med följande transformation:

$$[x' \quad y' \quad z' \quad 1] = [x \quad y \quad z \quad 1]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2)$$

Rotation kring y-axeln ges av:

$$[x' \quad y' \quad z' \quad 1] = [x \quad y \quad z \quad 1]\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3)$$

Rotation kring z-axeln ges av:

$$[x' \quad y' \quad z' \quad 1] = [x \quad y \quad z \quad 1]\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4)$$

Alla rotationer sker moturs om man ser i respektive koordinataxels riktning från origo, se fig 3. Rotation kring en godtycklig axel kan ske genom kombination, (s.k. konkatenering, se nedan), av rotationerna kring koordinataxlarna. Om axeln inte går genom origo, måste den först translateras så att den gör det. Sedan kan rotationen utföras och slutligen skall inversen till den första translationen ske.

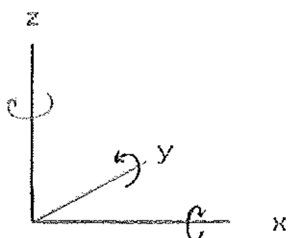


fig 3

Skalning

En tredje typ av transformation är skalning. Man kan här bestämma skalfaktorn i de olika koordinatriktningarna separat.

$$[x' \; y' \; z' \; 1] = [x \; y \; z \; 1]\begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Konkatenering

Samma resultat som vid succesiv användning av transformationer kan uppnås med en enda transformationsmatris, konkateneringen (ung. sammanbindningen) av sekvensen. Antag att två transformationer T_1 och T_2 skall succesivt appliceras. Dessa kan ersättas med en enda transformation T_3 som helt enkelt är produkten av T_1 och T_2.

Detta kan direkt demonstreras:
Punkten (x, y, z) transformeras till (x', y', z') av T_1:

$$[x' \; y' \; z' \; 1] = [x \; y \; z \; 1] \; T_1 .$$

Punkten (x'', y'', z'') fås genom att multiplicera på T_2:

$$[x'' \; y'' \; z'' \; 1] = [x' \; y' \; z' \; 1] \; T_2 = ([x \; y \; z \; 1] \; T_1) \; T_2 =$$

$$[x \; y \; z \; 1] \; (T_1 T_2).$$

Ögonkoordinatsystemet

Första steget i transformationen från wcs till skärmkoordinatsystemet blir att överföra från wcs till ögonkoordinatsystemet (eye coordinate system, ecs). Origo för ecs är fast placerad i den s.k. betraktelsepunkten (viewpoint) och z_e-axeln pekar mot mitten i synfältet. För att x_e- och y_e- axlarna skall ha samma riktningar som skärmens x- och y-axlar samt för att z_e-axeln skall peka mot föremålet, väljs ecs att vara vänsterortonormerat. Övergången från wcs till ecs bestäms av transformationen V (viewing transformation).

$$[x_e \; y_e \; z_e \; 1] = [x_w \; y_w \; z_w \; 1] \, V$$

ecs förhållande till wcs framgår av fig 4.

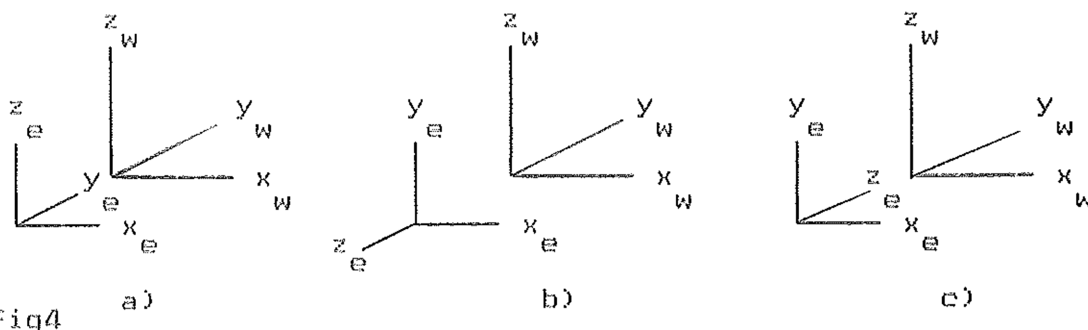


fig4

Gången i beräkningen av V är i vårt fall som följer.

1. Translation av koordinatsystemet till $(0, y_{e0}, 0)$.

Punkten $(0, y_{e0}, 0)$ i det ursprungliga koordinatsystemet blir origo (fig 4a). y_{e0} är här negativt.

$$T_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -y_{e0} & 0 & 1 \end{bmatrix}$$

2. Rotation +90° kring x-axeln (fig 4b).

$$T_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. Omvandling från höger- till vänster-koordinatsystem (fig 4c).

$$T_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$V = T_1 T_2 T_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspektivtransformation

Nu kan en bild med perspektiv genereras genom att i tur och ordning projicera föremålets punkter på bildskärmen. Skärmens koordinater för en punkt i ecs ges av:

$$x_s = \frac{Dx_e}{Sz_e} \qquad y_s = \frac{Dy_e}{Sz_e}$$

D är avståndet från viewpoint till skärmen och S är halva skärmens bredd. För att få ett riktigt perspektiv måste man alltså dividera med varje punkts djup. För räta linjer räcker det dock att transformera endast ändpunkterna och sedan interpolera, eftersom transformationen är linjär.

När man skall ta bort dolda ytor eller linjer måste man veta exakt djup för varje punkt på föremålet. Eftersom beräkningarna i hiddensurface-algoritmen sker i skärmens koordinatsystem (screen coordinate system, scs), måste en djupkoordinat införas i detta koordinatsystem, z_s. (z_s-axeln pekar inåt skärmen).

En punkt (x_s, y_s, z_s) projiceras på skärmpunkten (x_s, y_s). Eftersom alla punkter som projiceras på en speciell punkt på skärmen, bara kan skilja sig på z-koordinaten, blir jämförelsen av djup enkel.

Skärmens koordinatsystem

Beräkningen av djupkoordinaten z_s måste vara noggrann om resultatet skall vara användbart vid hiddensurface-beräkningarna. Speciellt måste räta linjer i ecs transformeras till räta linjer i scs. Lika viktigt är att plan transformeras på plan. Följande perspektivtransformation uppfyller dessa villkor.

$$x_s = \frac{x_e}{w} \qquad y_s = \frac{y_e}{w} \qquad z_s = \frac{S(z_e/D - 1)}{(1-D/F)w}$$

$$w = \frac{Sz_e}{D}$$

Några kommentarer:

- Om man vet att $D \leq z_e \leq F$, kan man maximera upplösningen i djupled genom att välja dessa gränser i "synfältspyramiden", se fig 5.

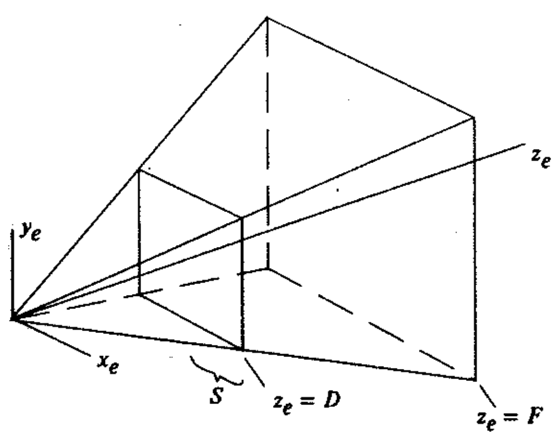


fig 5 (hämtad från Newman&Sproull)

z_e=D transformeras på z_s=0 och z_e=F transformeras på z_s=1, dvs. $0 \leq z_s \leq 1$.

- Perspektivtransformationen överför den trunkerade synfältspyramiden till en "låda" (viewbox) i scs där $-1 \le x_s \le 1$, $-1 \le y_s \le 1$ och $0 \le z_s \le 1$. Dessa gränser ger en "standard viewbox". Värdena kan sedan genom skalning och translation justeras så att de passar hårdvaran.

- Värdet av w är direkt relaterat till en punkts avstånd från viewpoint längs z_e-axeln. w skall ses som en fjärde koordinat i ecs, innehållande "perspektivinformation".

Transformationen kan enklast beskrivas m.h.a. en 4x4 matris för de linjära delarna av beräkningen, och ett separat steg för divisionen.

$$[x_h \; y_h \; z_h \; w_h] = [x_e \; y_e \; z_e \; 1] \, P$$

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & S/(D(1-D/F)) & S/D \\ 0 & 0 & -S/(1-D/F) & 0 \end{bmatrix}$$

$$x_s = \frac{x}{w} \qquad y_s = \frac{y}{w} \qquad z_s = \frac{z}{w}$$

Indexet h står för homogent koordinatsystem. Det homogena koordinatsystemet (homogeneous coordinate system, hcs) är ett matematiskt hjälpmedel för att beskriva projektions-tranformationer. För att transformera en punkt (x, y, z) i vanliga tredimensionella koordinater till en homogen representation, väljer vi bara ett tal w skilt från noll och bildar vektorn [wx wy wz w] Talet w kallas skalfaktorn eller den homogena koordinaten. En homogen punkt [a b c d] kan återföras till vanliga tredimensionella koordinater genom att dividera med skalfaktorn: (a/d b/d c/d). Translation, rotation och skalning kan utföras som tidigare även på de homogena koordinaterna.

Klippning

För att linjer eller delar av linjer bakom viewpoint eller på annat sätt utanför viewbox, inte skall ritas ut på skärmen, måste de klippas. Detta skall göras efter transformationen till homogena koordinater (P-matrisen), men innan divisionen med den homogena koordinaten.
Klippningsgränserna blir: $-w \le x \le +w$, $-w \le y \le +w$ och $0 \le z \le +w$
I appendix 1 sid 2 återfinns klippningsalgoritmen i procedure Clip3. Där har s.k. window-edge koordinater

använts. De ger avståndet mellan den aktuella punkten och var och en av klippningsplanen. Punkten (x, y, z, w) representeras av 6-elementsvektorn [w+x, w-x, w+y, w-y, 0+z, wz]. Om ingen av komponenterna är negativ, ligger punkten innanför klippningsplanen. Om en eller flera komponenter är negativa, måste klippning ske. P.g.a. att transformationen till window-edge koordinater är linjär, kan punkter längs en linje interpoleras m.h.a. ändpunkterna. Detta är nödvändigt om en linje bara delvis skall klippas. Om linjen ligger helt utanför viewbox, sätts flaggan outsideclipbox till true. Härmed anger man att denna linje inte skall ritas ut, vilket annars kan bli fallet om viewbox gränser ligger innanför skärmens gränser.

Viewport

Genom att efter klippning och innan division applicera ytterliggare en matris benämnd S, kan man placera bilden på valfritt ställe på skärmen. Även skalfaktorerna i x- och y-led kan väljas,

$$S = \begin{bmatrix} V_{sx} & 0 & 0 & 0 \\ 0 & V_{sy} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ V_{cx} & V_{cy} & 0 & 1 \end{bmatrix}$$

Denna bildruta på skärmen kallas viewport. Den har centrum i punkten (V_{cx}, V_{cy}) och är $2V_{sx}$ bred och $2V_{sy}$ enheter hög.

Sammanfattning av transformationerna

Alla de behandlade transformationerna kan sammanfattas enligt

$$[x_w \;\; y_w \;\; z_w \;\; w] = [x \;\; y \;\; z \;\; 1]\; V\; P\; (\text{clip})\; S$$

$$x_s = \frac{x}{y}, \quad y_s = \frac{y}{w}, \quad z_s = \frac{z}{w}$$

V transformerar till ett vänsterortonormerat system med origo mitt på skärmen och med synfältet riktat längs +z-axeln. Matrisen P beskriver perspektivtransformationen. Klippningen appliceras på de homogena representationerna av linjer för att begränsa dem till viewbox. Slutligen används transformationen S för att utvidga standard viewbox till en godtycklig viewport på skärmen. Efter divisionen är koordinaterna i en lämplig form för användning i hiddensurface algoritm.

3.3 Grafikpaket

I ett tredimensionellt grafikpaket måste man kunna utföra transformationerna enl. sammanfattningen ovan. Det görs i

```
procedure Transform3(xm,ym,zm:real; var xs1,ys1,zs1,
                     xs2,ys2,zs2:real; var notwrite:boolean)
```

procedure MoveTo3(x,y,z:real) anger vid vilken punkt nästa linje skall starta. Eftersom man till klippningsalgoritmen behöver en linjes båda ändpunkter, kan man i MoveTo3 bara utföra transformationen till och med homogena koordinater.

procedure LineTo3(x,y,z:real) anger en linjes slutpunkt och om MoveTo3 inte anropas precis efter LineTo3 anropats, är det även nästa linjes begynnelsepunkt.

Anm. Matrisen M's funktion förklaras i kapitel 5.

Dessutom har i vårt fall införts en del primitiver för att kunna betrakta bilden från olika håll. En möjlighet att bestämma viewport har också införts.

procedure Pan3(theta:real) : rotation av ecs kring y_e-axeln vinkeln theta. (theta i grader). Till matrisen V multipliceras en rotationsmatris (4). ecs roterar medurs då fi>0.

procedure Tilt3(fi:real) : rotation kring x_e-axeln vinkeln fi. Matrisen (2) används. ecs roterar medurs då fi>0.

procedure Roll3(Zchange:real) : translation av ecs i z_e-axelns riktning. Matris (1) med T_x och T_y = 0. Positivt T_z betyder att origo för ecs förflyttas längre ifrån origo för wcs.

Genom kombination av Pan3 och Roll3 kan man få en horisontell rörelse enligt:

```
Pan3(90);
Roll3(x);
Pan3(-90);
```

En vertikal rörelse ges av:

```
Tilt3(90);
Roll3(y);
Tilt(-90);
```

procedure Zoom3(Dchange:real) : Här går man direkt in och ändrar konstanten D i P-matrisen. Om förhållandet D/S är litet blir bilden lik den man får med ett vidvinkelobjektiv. Om förhållandet i stället är stort får man en telefoto-bild.

Man kan med hjälp av Zoom3 bestämma hur mycket av en bild man vill se, man bestämmer ett fönster (window).

procedure NewViewport3(vxl,vxr,vyb,vyt:real) : Var innehållet i det fönster som bestäms i Zoom3, skall placeras på skärmen, kan avgöras m.h.a. NewViewport3. Argumenten är x-koordinaterna för viewportens vänstra resp. högra kant samt y-koordinaterna för undre och övre kanten.

4. ELIMINERING AV DOLDA YTOR

I verkliga livet döljer ogenomskinliga föremål vissa ytor och hindrar oss från att se dem. Inom datorgrafik måste man m.h.a. en speciell algoritm räkna ut vilka ytor som skall visas på skärmen och vilka som inte skall göra det. Det finns flera olika algoritmer och här skall bara den tas upp som går under namnet "depth-buffer algorithm". I den håller man för varje pixel på skärmen reda på djupet för det objekt inom pixeln som ligger närmast observatören. Detta görs i matrisen depth, som är en 512x512 matris = antal pixel på skärmen.

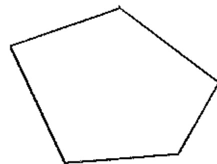
Algoritmen

1. För alla pixel på skärmen: sätt depth[x,y] till ett värde större än det maximalt tillåtna z-värdet i scs.

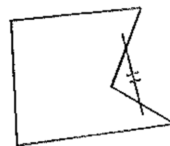
2. Finn för varje polygon de pixel som ligger innanför polygonens gränser. Gör för varje av dessa pixel följande.
   a) Beräkna djupet z för polygonpunkten (x,y).
   b) Om z<depth[x,y], så ligger den del av polygonen närmare betraktaren än någon tidigare. I så fall skall depth[x,y] få värdet z, och punkten skall sättas. Om z>depth[x,y] skall inget göras.

Objektens ytor

Alla ytor förutsättes bestå av konvexa polygoner. Denna typ av polygon karakteriseras av att från varje punkt i polygonen kan man nå vilken annan punkt som helst i polygonen, utan att gå utanför denna. Se fig 6.



konvex polygon



ej konvex

fig 6

Detta villkor underlättar beräkningarna i proceduren HiddenSurface.

En yta behandlas i taget. Först sätts hörnpunkterna, angivna i scs, in i en dubbellänkad lista. Detta görs m.h.a. procedure StartPolygon, procedure NextVertex och procedure LastVertex. De har alla en punkts x, y och z-koordinat som argumemnt. I StartPolygon initieras den dubbellänkade listan samt utföres stegen fram till klippningen i tranformationen

från wcs (egentligen från modellkoordinatsystemet, se kap 5) till scs. I NextVertex sätts en kantlinjes ändpunkter in i listan och om linjen ev. p.g.a. klippning fått en ny begynnelsepunkt eller om föregående anrop var till StartPolygon, så sätts även linjens begynnelsepunkt in. Med ett anrop till LastVertex sätts den sista hörnpunkten in. LastVertex fungerar på samma sätt som NextVertex förutom att proceduren HiddenSurface anropas efter det att den sista punkten satts in i listan.

Det är viktigt att en polygons hörnpunkter anges medurs, sett framifrån (utifrån). (De objekt som behandlas antas kunna byggas upp av ogenomskinliga polyedrar och man kan inte krypa inuti polyedern). Varför det är viktigt att ange hörnpunkterna i denna ordning framgår nedan.

HiddenSurface

Depth-buffer algoritmen är enkel. Att det inte är lika enkelt att realisera den programmässigt torde framgå av procedure HiddenSurface (appendix 1 sid 4).
Depth-buffer algoritmen utföres för en polygon i taget och man arbetar hela tiden med skärmkoordinater.

Som första steg beräknas normalen till ytan. Detta görs genom att ta kryssprodukten av vektorn mellan första och andra hörnpunkten i den länkade listan och den mellan andra och tredje.

PQ = punkt2 - punkt1 , PR = punkt3 - punkt2
Normalen = PQxPR =
$(PQ_y*PR_z - PQ_z*PR_y , PQ_z*P_x - PQ_x*PR_z , PQ_x*PR_y - PQ_y*PR_x)$.

För att normalen skall peka åt rätt håll, är det viktigt att hörnpunkterna lagts in medurs i listan. Om normalens z-komponent har ett positivt värde, betyder det att normalen pekar inåt skärmen (scs är ett vänsterortonormerat koordinatsystem, med z-riktningen inåt skärmen). Eftersom vi bara arbetar med slutna volymer, innebär en positiv z-komponent att ytan täcks av en annan yta och kan elimineras. En förenklad hiddensurface kallad backfaceremoval, kan göras genom att om z<0 rita ut den aktuella polygonen, annars skall inget göras.

Om z-komponenten är negativ, blir nästa steg att beräkna polygonplanets ekvation på formen ax+by+cz+d=0 . (a,b,c) är ytans normal och är redan beräknad. Återstår att beräkna d genom att sätta in en känd punkt som ligger i planet, i ekvationen ovan: d = -ax-by-cz . Om det visar sig att både a,b och c har värden nära noll, ligger de tre polygonpunkter, som använts för beräkningen, praktiskt taget på sammma linje. Då har en felaktig polygon lagts in i listan och en felutskrift sker varpå man hoppar ut ur HiddenSurface via borttagning av den länkade listan.

Annars kontrolleras att alla övriga hörnpunkter ligger i samma plan, genom insättning i planets ekvation. Om en punkt inte ligger i planet, lämnas felutskrift och hopp sker ut ur HiddenSurface via borttagning av den länkade listan.

Om hopp inte skett ut ur proceduren, blir nästa steg att dividera koefficienterna a,b och d med c. Detta för att underlätta beräkningarna av z-koordinater senare. Är $c<10^{-6}$ får a,b och d närmevärden enligt $a=a*10^{6}$, $b=b*10^{6}$ och $d=d*10^{6}$. Detta för att inte riskera alltför stora värden på koefficienterna.

Scanconversion

Den metod som används för att gå igenom varje pixel inom polygonen, kallas scanconversion. För varje x-värde går man igenom alla y-värden och beräknar de olika punkternas z-koordinat. Dessa jämföres med värdet i depth-buffern. Man sveper eller "scannar" polygonen systematiskt (se fig 7).

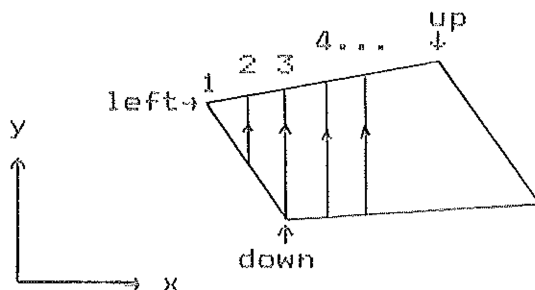


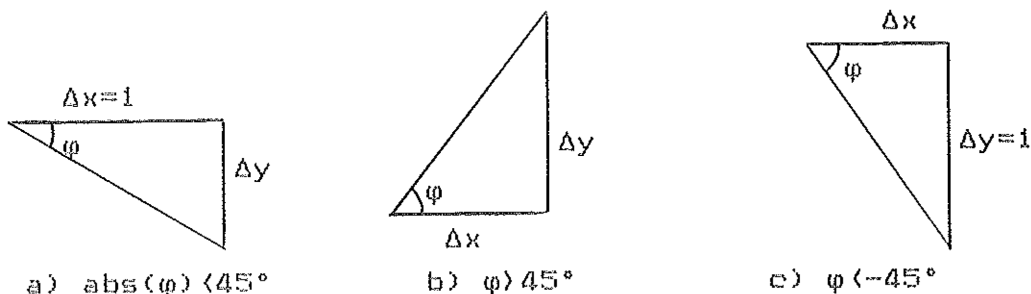
fig 7

Pekaren left sätts att peka på polygonpunkten med den minsta x-koordinaten. Om flera hörnpunkter har samma minsta x-koordinat, väljs den först funna. Pekarna up och down pekar på de hörnpunkter som "står i tur" på ovan- resp undersidan av polygonen, dvs. de punkter som ligger närmast före och efter den left pekar på.

Man måste hela tiden veta lutningen för de båda kantlinjerna som begränsar det aktuella svepet. Detta för att rätt antal y-koordinater skall behandlas. Efter ett svep som börjar eller slutar i en hörnpunkt, måste man beräkna en ny lutning för den sida där hörnpunkten befinner sig. Flaggorna DeltaUpChange och DeltaDownChange anger om övre resp. undre begränsningslinjens lutning skall ändras.

Vid beräkningen av en kantlinjes lutning, skiljer vi på fallen (a) linjen lutar mindre än 45° och (b) linjen lutar mer än 45°. I fall (a) sätts Δx till ett (Δx kallas i programet DeltaxUp resp. DeltaxDown för övre resp. undre begränsningslinjen), och Δy (DeltayUp resp. DeltayDown)

beräknas, se fig 8a. (Lutningen = $\Delta y/\Delta x$). $-1 \leq \Delta y \leq +1$.



a) abs(φ)<45°   b) φ>45°   c) φ<-45°

fig 8

I fall (b) sätts Δy till +1 (positiv lutning) eller -1 (negativ lutning) och Δx beräknas, se fig 8b och c. Det gäller $0 \leq \Delta x < 1$. I specialfallet att linjen bara ändras i z-led, kommer Δx och Δy båda att bli noll. Men en polygon som innehåller en sådan linje, ligger i det plan vinkelrätt mot skärmplanet, och kommer bara att synas som ett streck.

Kantpunkterna behandlas för sig och punkterna inuti polygonen för sig. På så sätt kan man markera konturerna genom att sätta annan färg på kanterna än på polygonen i övrigt.

Först beräknas lutningarna för de båda kantlinjer som utgår från den punkt left pekar på. Därefter sker följande: För varje x-värde, (betecknas i programet xt), mellan två hörnpunkter kollas först övre kantlinjen. I hjälpvariabeln yhelpup och dess trunkerade version yhelpuptrunc, håller man reda på vilket y-värde den aktuella kantpunkten har. Det finns även en hjälpvariabel xhelpup med tillhörande trunkerade värde xhelpuptrunc, som ger kantpunktens x-värde. Genom insättning i planets ekvation av xhelpup och yhelpup beräknas punktens z-värde. Om detta värde är mindre än depth[xt,yhelpuptrunc], sätts pixeln och depth[xt,yhelpuptrunc] tilldelas värdet av z. Därefter inkrementeras yhelpup med det tidigare beräknade DeltayUp varefter yhelpuptrunc får det nya yhelpups trunkerade värde. Till xhelpup adderas DeltaxUp. xhelpuptrunc får xhelpups trunkerade värde. Innan någon punkt med x-koordinat xt behandlats, är xhelpuptrunc=xt. Om det nya xhelpuptrunc har samma värde som innan inkrementeringen, innebär detta att linjen lutar mer än 45°, och yhelpuptrunc har ändrats en enhet. Därmed har man en ny punkt precis ovanför eller under den tidigare och beräkning av z-koordinat etc. sker som tidigare. Om linjens lutning är nära 90° är DeltaxUp litet och man kan på detta sätt ligga flera varv i while-satsen och flera pixel ovanpå varandra kan sättas.

Om xhelpuptrunc > xt fortsätter vi med den undre kanten. (Nästa punkt som behandlas på den övre kanten blir (xhelpup,yhelpup,z).) Om yhelpuptrunc får ett värde större än y-koordinaten för den hörnpunkt på övre kanten som "står i tur" (vid positiv lutning mer än 45°) eller mindre än

hörnpunktens y-värde (vid negativ lutning mindre än -45°), har vi nått hörnpunkten och för att undvika att punkter utanför polygonen sätts, fortsätter vi med undre kanten.

Den undre kantlinjen behandlas analogt med den övre. Hjälpvariablerna är här yhelpdown, yhelpdowntrunc, xhelpdown, xhelpdowntrunc. Också här ser man till att ingen punkt utanför kanten sätts.

Nästa steg blir att behandla punkterna mellan kanterna. Detta görs för x-värdena left^.x t.o.m. Nextxtrunc-1. Hjälpvariablerna xscanup och yscanup får värdena av x- resp. y-koordinaten för den punkt på övre begränsningslinjen som har minst y-koordinat i det aktuella svepet. P.s.s. får xscandown och yscandown x- och y-värdena för den punkt på undre begränsningslinjen som har störst y-koordinat. För att få så bra nogrannhet som möjligt i beräkningen av z-värdet för de inre punkterna, beräknas inkrement i x- resp. y-led för svepet av de inre punkterna (DeltaxMiddle resp. DeltayMiddle). Svepet sker för yt=trunc(yscandown)+1 till yt=trunc(yscanup)-1. I varje varv beräknas z-värdet för punkten (xhelpinternal,yhelpinternal,z). Jämförelse sker med depth[xt,yt] som tidigare. (De trunkerade värdena av xhelpinternal och yhelpinternal är xt och yt).

Specialfallet att första begränsningslinjen är vertikal måste behandlas för sig. I första steget prickas kantpunkterna ut och inget svep av inre punkter sker. I nästa steg är den vertikala linjens ändpunkter utgångspunter för den övre resp. den undre begränsningslinjen. Eftersom svepet av inre punkter sker med början vid x=left^.x, riskerar man att den vertikala kantlinjen skrivs över av inre punkter. M.h.a. flaggan verticalline elimineras denna risk.

Man kan enkelt ge punkterna inuti polygonen en annan färg än kantpunkterna genom att före svepet anropa proceduren CmiCol med ett annat färgnummer än kantens som argument. Efter svepet återställs kantfärgen. När en hel polygon behandlats, förstörs listan med hörnpunkter genom att göra dispose på varje element.

Shading

Om man tänker sig att figuren belyses av en punktljuskälla, kan man uppnå effekten att vissa ytor blir ljusare än andra. Hur ljus eller mörk en yta blir beror på hur mycket ljus den reflekterar. Formeln för diffus reflektion är enligt Lambert's lag: E=Konst·cos(i). E anger hur mycket ljus en yta reflekterar. Konst är en normeringskonstant och i är vinkeln mellan den infallande ljusstrålen och ytans normal. Vi behandlar en ljuskälla som ligger i betraktelsepunkten. Detta innebär att ingen säskild hänsyn behöver tas till skuggor. cos(i) beräknas som skalärprodukten mellan

polygonens normal och skärmens normal.

Exempel

Detta exempel illustrerar hu scanconversion går till. Polygonen som behandlas ges i fig 9.

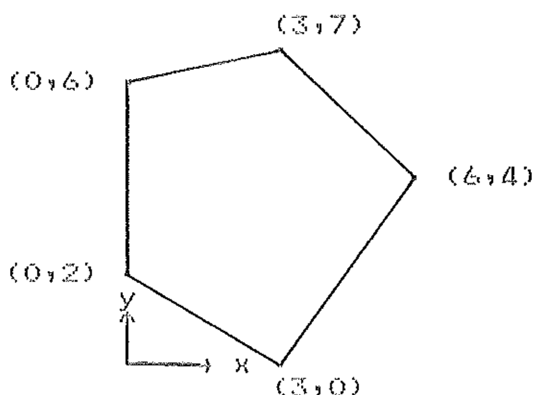


fig 9

Planets ekvation antas ha bestämts tidigare och z-koordinaterna kan beräknas om man känner x- och y-koordinaterna. Pekaren left antas sättas att peka på punkten (0,2). Punkten (0,6) är också möjlig om den ligger före (0,2) i den länkade listan. Pekaren up pekar på (0,6) och down på (3,0). Beräkning av lutning ger DeltaxUp=0, DeltayUp=1, DeltaxDown=1, DeltayDown=-0.667. Vi har specialfallet vertikal kantlinje och punkterna (0,2), (0,3)...(0,6) behandlas varpå left flyttas att peka på (0,6) och up på (3,7). Den övre kantens lutning beräknas och ger värdena DeltaxUp=1 och DeltayUp= 0.333...
För den övre kantlinjen kommer följande att hända när xt går från 0 till 3.

xhelpup	xhelpuptrunc	yhelpup	yhelpuptrunc	xt	yscanup-1
0.0	0	6	6	0	5
1.0	1	6.333	6	1	5.333
2.0	2	6.667	6	2	5.667
3.0	3	7.0	7	3	

Punkterna (0,6), (1,6), (2,6), (3,7) sätts på ovankanten. Ev. kan också punkten (3,6) sättas p.g.a. trunkeringen. På den nedre kanten sätts på motsvarande sätt punkterna (0,2), (1,1), (2,0), (3,0). De inre punkterna (1,2)-(1,5), (2,1)-(2,5) sätts till den inre färgen. (Inga inre punkter för x=0 behandlas ty här finns en vertikal begränsningslinje.)

left flyttas att peka på (3,0) och down pekar på (6,4). DeltaxDown=0.75, DeltayDown=1. Punkten (3,7) behandlas igen liksom punkten (3,0).

left flyttas till (3,7) och up till (6,4). DeltaxUp=1, DeltayUp-=1. På ovansidan sätts punkterna (3,7), (4,6), (5,5), (6,4).
På undersidan händer följande:

xhelpdown	xhelpdowntrunc	yhelpdown	yhelpdowntrunc	xt	yscandown+1
3.0	3	0.0	0	3	1.0
3.75	3	1.0	1	3	2.0
4.5	4	2.0	2	4	3.0
5.25	5	3.0	3	5	4.0
6.0	6	4.0	4	6	

Punkterna (3,0), (3,1), (4,2), (5,3), och (6,4) sätts. De

inre punkterna (3,2)-(3,6), (4,3)-(4,5), (5,4) sveps.
Den utritade polygonen får utseende enligt fig 10.

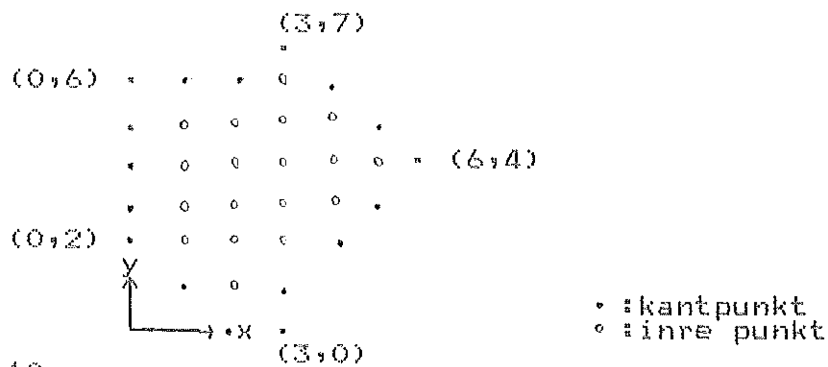


Fig 10

5. UPPBYGGNAD AV GRAFISKA MODELLER

Om man har flera olika objekt på en bild, kan man få dem att röra sig oberoende av varandra genom att beskriva dem var och en i ett eget koordinatsystem, ett modellkoordinatsystem. Beskrivningen i detta koordinatsystem överföres sedan, m.h.a. transformationsmatrisen M, till en beskrivning i världskoordinatsystemet. Genom att ändra M kan man få objektet att röra sig i wcs.

Om punkten P_m är en punkt på objektet i modellkoordinatsystemet, blir samma punkt i wcs $P_w = P_m M$.

Om man t.ex. vill att objektet skall rotera kring en axel, multiplicerar man M-matrisen med rotationsmatrisen framifrån och får en ny M-matris. Om rotationsmatrisen ges av T fås den nya M-matrisen enl. M = T M.

Alla objekt beskrivs nu i sina modellkoordinatsystem, men som tidigare måste de transformeras till skärmens koordinatsystem innan de kan visas på skärmen. Första steget i denna transformation blir övergången från beskrivning i modellkoordinatsystemet till beskrivning i wcs. Sedan sker transformationer som tidigare via ögonkoordinatsystemet, homogena koordinatsystemet, klippning och viewportplacering till skärmens koordinatsystem. Enda skillnaden mot tidigare är att i proceduren Transform3 M,V och P-matriserna multipliceras på punkten mot tidigare bara V och P.

M sätts initiellt till enhetsmatrisen. För att flytta modellkoordinatsystemets origo samt ändra koordinataxlarnas orientering relativt wcs, behövs speciella primitiver. De är:

```
procedure Translate3(Tx,Ty,Tz:real);
procedure Rotate3x(fi:real);
procedure Rotate3y(fi:real);
procedure Rotate3z(fi:real);
```

Beskrivningen av roboten (appendix 1 sid 22 samt fig 11), visar hur förflyttningen av modellkoordinatsystemet sker så att proceduren Block hela tiden kan användas för att rita ut den aktuella delen. De punkter där modellkoordinatsystemet roteras, blir också vridningsaxlar när roboten rör sig. Vinklarna som används i rotationsmatrisen, utgör argumenten till proceduren Robot6.

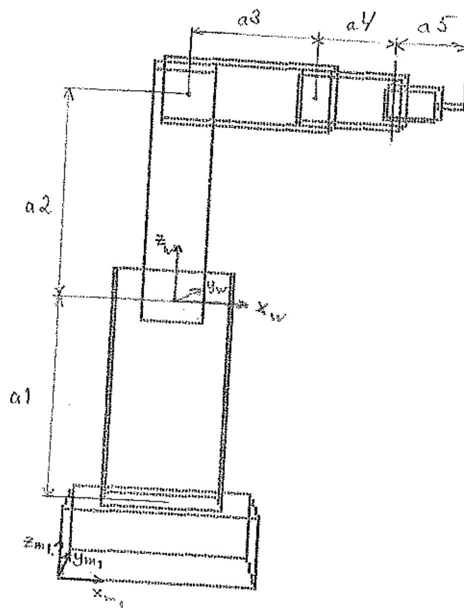


fig 11

6. ANIMERING AV ROBOT

Följande problem skulle lösas: Gör det möjligt att flytta robotarmen i ett kartesiskt koordinatsystem, dvs. i x-, y- och z-led. Dessutom skall handen i spetsen kunna ställas in med godtycklig orientering. (Vissa geometriska begränsningar finns dock.)

Världskoordinatsystemet väljs lämpligen för att beskriva robotens rörelse. Dess origo ligger mitt på robotens andra axel (se fig 12).
För att sätta spetsen i en bestämd punkt behövs tre frihetsgrader. Om dessutom robothandens orientering skall kunna väljas, behövs ytterliggare tre frihetsgrader. Detta motiverar att Robot6 har sex frihetsgrader.

Robotspetsens önskade läge och orientering kan anges med en matris:

$$\begin{bmatrix} n_x & n_y & n_z & 0 \\ o_x & o_y & o_z & 0 \\ a_x & a_y & a_z & 0 \\ p_x & p_y & p_z & 1 \end{bmatrix} \qquad (*)$$

$p = (p_x, p_y, p_z)$ är den önskade punkten.

$n = (n_x, n_y, n_z)$ är den önskade riktningen. Den är normerad.

$o = (o_x, o_y, o_z)$ och $a = (a_x, a_y, a_z)$ bestämmer tillsammans med n, hur koordinatsystemet i spetsen skall orienteras, se fig 12. n motsvarar x-axeln, o motsvarar y-axeln och a motsvarar z-axeln i ett högerortonormerat koordinatsystem.

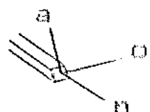


fig 12

Eftersom o och därmed a lätt kan väljas genom en rotation kring n när väl denna vektor är bestämd (rotation vinkeln θ6), så kan vi förenkla räkningarna genom att räkna med de fem vinklar som behövs för att välja p och n.

Genom att beräkna M-matrisen för robotarmens spets, får vi reda på dess läge och orientering. (*) anger det önskade läget och orienteringen. Genom att sätta M och (*) lika fås ett antal villkor varur vinklarna θ1-θ5 kan beräknas.

Spetsens M-matris fås genom att multiplicera alla de matriser anropen till Translate resp. Rotate3x, Rotate3y och Rotate3z ger upphov till. För beräkningarna hänvisas till appendix 2.

Följande villkor fås:

p_x=a5*C5*C234*C1+a4*C234*C1+a3*C23*C1-a5*S5*S1-a2*S2*C1 (a1)

p_y=-a5*C5*C234*S1-a4*C234*S1-a3*C23*S1-a5*S5*C1+a2*S2*S1 (b1)

p_z=a5*C5*S234+a4*S234+a3*S23+a2*C2+a1+z_0 (c1)

n_x=C5*C234*C1-S5*S1 (d1)

n_y=-C5*C234*S1-S5*C1 (e1)

n_z=C5*S234 (f1)

S1=sinθ1, C1=cosθ1, C234=cos(θ2+θ3+θ4) etc.
a1-a5 är avstånd mellan vridningsaxlar enl. fig 12.
z_0 är den första vridningspunktens z-koordinat. I och med att origo för wcs ligger på den andra axeln, blir z_0=-a1.

I fortsättningen används uppdelningen av M enl. appendix 2: $M=A_5A_4A_3A_2A_1$.

Resultatet av beräkningarna nedan används i proceduren NewAngles. I denna betecknas vinklarna α1-α5.

Multiplikation med A_1^{-1} från vänster ($MA_1^{-1}=A_5A_4A_3A_2$) ger villkoren:

C1*p_x-S1*p_y = (a5*C5+a4)*C234+a3*C23-a2*S2 (a2)

S1*p_x+C1*p_y = -a5*S5 (b2)

p_z-z_0 = a5*C5*S234+a4*S234+a3*S23+a2*C2+a1 (c2)

C1*n_x-S1*n_y = C5*C234 (d2)

S1*n_x+C1*n_y = -S5 (e2)

n_z = C5*S234 (f2)

(b2) och (e2) ger: S1*p_x+C1*p_y = a5*(S1*n_x+C1*n_y)

$$\frac{S1}{C1} = \tan \alpha 1 = \frac{a5*n_y - p_y}{p_x - a5*n_x}$$

$$\alpha 1 = \arctan\left(\frac{a5*n_y - p_y}{p_x - a5*n_x}\right)$$

Anm. En arctanfunktion som ger värden mellan $-\pi$ och $+\pi$. används. Den klarar även av fallet med noll i nämnaren.

S1:=sinα1

C1:=cosα1

(b2) ⇒ $S5 = -(S1*p_x + C1*p_y)/a5$

$C5 = \pm\sqrt{1-(S5)^2}$

Det positiva värdet väljs, vilket ger vridningsvinkeln $-\pi \le \alpha 5 \le +\pi$.

$$\alpha 5 = \arctan\left(\frac{S5}{C5}\right)$$

(d2,f2) ⇒ $\alpha 234 = \arctan\left(\frac{n_z}{C1*n_x - S1*n_y}\right)$

S234=sin(α234)

C234=cos(α234)

Villkoret $-\pi \le \alpha 234 \le +\pi$ ger en begränsning i robotens rörelsefrihet.

$$MA_1^{-1}A_2^{-1} = A_5A_4A_3 \Rightarrow$$

$$C2*(C1*p_x - S1*p_y) + S2*(p_z - z_0) - S2*a1 = (a5*C5+a4)*C34 + a3*C3 \quad (a3)$$

$$S1*p_x + C1*p_y = -S5*a5 \quad (b3)$$

$$-S2*(C1*p_x - S1*p_y) + C2*(p_z - z_0) - C2*a1 =$$

$$C2*(C1*n_x-S1*n_y)+S2*n_z = C5*C34 \qquad (d3)$$

$$S1*n_x+C1*n_y = -S5 \qquad (e3)$$

$$-S2*(C1*n_x-S1*n_y)+C2*n_z = C5*S34 \qquad (f3)$$

Dessa villkor ger ingen ny information, utan vi fortsätter med nästa steg.

$$MA_1^{-1}A_2^{-1}A_3^{-1} = A_5A_4 \quad \Rightarrow$$

$$C23*(C1*p_x-S1*p_y)+S23*(p_z-z_0-a1)-S3*a2-a3 = (a5*C5+a4)*C4 \qquad (a4)$$

$$S1*p_x+C1*p_y = -a5*S5 \qquad (b4)$$

$$-S23*(C1*p_x-S1*p_y)+C23*(p_z-z_0-a1)-a2*C3 = (a5*C5+a4)*S4 \qquad (c4)$$

$$C23*(C1*n_x-S1*n_y)+S23*n_z = C5*C4 \qquad (d4)$$

$$S1*n_x+C1*n_y = -S5 \qquad (e4)$$

$$-S23*(C1*n_x-S1*n_y)+C23*n_z = C5*S4 \qquad (f4)$$

Inför p_x' och p_z' enligt

$$p_x' = C1*p_x-S1*p_y-(a5*C5+a4)*C234 = a3*C23-a2*S2$$

$$p_z' = p_z-z_0-(a5*C5+a4)*S234-a1 = a3*S23+a2*C2$$

Mittenleden är kända. Vinklarna α2 och α3 söks.

$(p_x')^2 + (p_z')^2 = (a3)^2*(C23)^2 + (a2)^2*(S2)^2 - 2*a2*a3*C23*S2 +$

$+ (a3)^2*(S23)^2 + (a2)^2*(C2)^2 + 2*a2*a3*S23*C2 =$

$= (a3)^2 + (a2)^2 + a*a2*a3*S3$

$$S3 = \frac{(p_x')^2 + (p_z')^2 - (a3)^2 - (a2)^2}{2*a2*a3}$$

abs(S3) ≤ 1 utgör ett geometriskt villkor. Om abs(S3) > 1 så kan den önskade punkten ej nås.

$$C3 = \pm\sqrt{1-(S3)^2}$$

$$\alpha 3 = \arctan(\frac{S3}{C3})$$

$$MA_1^{-1}A_2^{-1}A_3^{-1}A_4^{-1} = A_5 \Rightarrow$$

$$C234*(C1*p_x-S1*p_y)+S234*(p_z-z_0-a1)-a2*S34-a3*C4 = a5*C5+a4 \quad (a5)$$

$$S1*p_x+C1*p_y = -a5*S5 \quad (b5)$$

$$-S234*(C1*p_x-S1*p_y)+C234*(p_z-z_0-a1)-a2*C34+a3*S4 = 0 \quad (c5)$$

$$C234*C1*n_x-C234*S1*n_y+S234*n_z = C5 \quad (d5)$$

$$S1*n_x+C1*n_y = -S5 \quad (e5)$$

$$-S234*C1*n_x+S234*S1*n_y+C234*n_z = 0 \quad (f5)$$

Inför:

$$p_x'' = C234*(C1*p_x-S1*p_y)+S234*(p_z-z_0-a1)-a5*C5-a4$$

$$p_z'' = -S234*(C1*p_x-S1*p_y)+C234*(p_z-z_0-a1)$$

$$(a5) \Rightarrow \quad p_x'' = a2*S34+a3*C4$$

$$(c5) \Rightarrow \quad p_z'' = a2*C34-a3*S4$$

Vinkeln α4 sökas. genom att utveckla S34 och C34 fås ett linjärt ekvationssystem.

$$C4*(a2*S3+a3) + S4*a2*C3 = p_x''$$

$$C4*a2*C3 - S4*(a2*S3+a3) = p_z''$$

Lösning av ekvationssystemet ger :

$$\frac{S4}{C4} = \frac{p_x''*a2*C3 - p_z''*(a2*S3+a3)}{p_z''*a2*C3+p_x''*(a2*S3+a3)}$$

$$\alpha4 = \arctan(\frac{S4}{C4}) \quad \text{om } C4 \neq 0 \text{ annars } \alpha4 = \pm\pi/2$$

En spärr har införts så att armen inte kan vikas ihop som en fällkniv. Begränsningarna för α4 blir:
$-9*\pi/10 \leq \alpha4 \leq +9*\pi/10$

Slutligen beräknas α2 enligt α2 = α234-α4-α3. Denna beräkning kan ge ett värde för α2 utanför intervallet -π/2-+π/2. Men α2 måste ligga innanför detta intervall och därför adderas eller subtraheras π ifrån det uträknade α2, så att vinkeln hamnar innanför intervallet.

De beräknade vinklarna sätts in i uttrycken a1-f1 för att kontrollera att vinklarna ger det önskade läget och den önskade orienteringen. Om de överensstämmer väl får de globala vinklarna θ1-θ5 värdena av α1-α5.

En procedur MoveRob har införts med vars hjälp man kan placera robotspetsen på önskad punkt och med önskad riktning. Med denna kan man skriva program för hu roboten skall röra sig, t.ex. vid "svetsning".

7. BESKRIVNING AV HUVUDPROGRAM

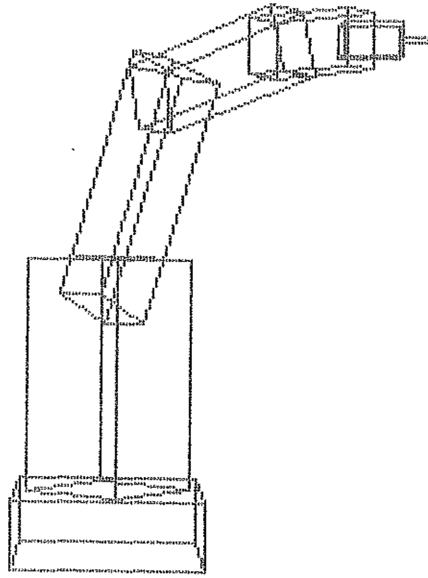
De beskrivna primitiverna finns på filen threed.pak. Där finns dessutom procedurer för multiplikation av en vektor och en matris (MultiplyVectMat),för multiplikation av två matriser (MultiplyMatMat) och en procedur för initialisering av M,V,P och S-matriserna (InitThreeDim). Dessutom finns en del skalfaktorer.

I filen robot.pak finns ett huvudprogram som använder procedurerna i threed.pak. Om vi tittar närmare på detta huvudprogram (appendix 1 sid 24) så anger vi först genom anropet TwoPlanes att två plan skall användas vid uppritning av bilden. DrawBackground säger att uppritningen skall ske i ett bakre plan, osynligt för betraktaren. Genom SwapPlane byter man sedan plats på bakre och främre planet. Genom CmiErase nollställs alla pixel i det plan man skriver i. Ändringen i synfält kan ske genom anrop till Pan3, Tilt3, Roll3 och Zoom3. Denna interaktion sker via terminal. Det finns även direkta kommando för att röra sig horisontellt och vertikalt samt för att rotera kring z_e-axeln.

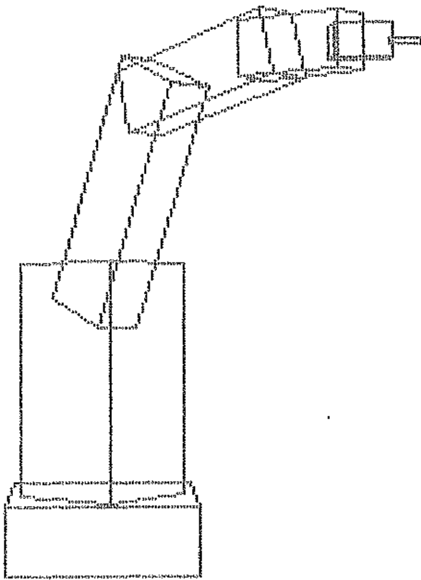
Roboten styrs m.h.a. styrspakar med fyra frihetsgrader. Man arbetar i två moder, en där spetsens x-, y- resp. z-koordinat ändras och en där spetsens orientering ändras. De globala variablerna p_x, p_y, p_z resp. n_x, n_y, n_z ändras m.h.a. styrspakarna varpå NewAngles anropas. Om den önskade punkten ej kan nås måste p_x, p_y, p_z resp. n_x, n_y, n_z återställas till det värde de hade innan senaste kommandot. Vinklarna theta får de värde de hade innan kommandot. Detta för att undvika att roboten fastnar p.g.a. att den försöker nå en punkt som är omöjlig att nå. Denna återställning sker i NewAngles.

Vidare finns kommando för backfaceremoval och hiddensurface. Hur dessa inverkar på den slutliga bilden framgår av fig 13 b och c. Jämför med fig 13 a, där inga dolda linjer alls har tagits bort.

Slutligen finns kommando för att tona figuren (shading), samt för att bestämma inre färg och kantfärg för figuren.



a)



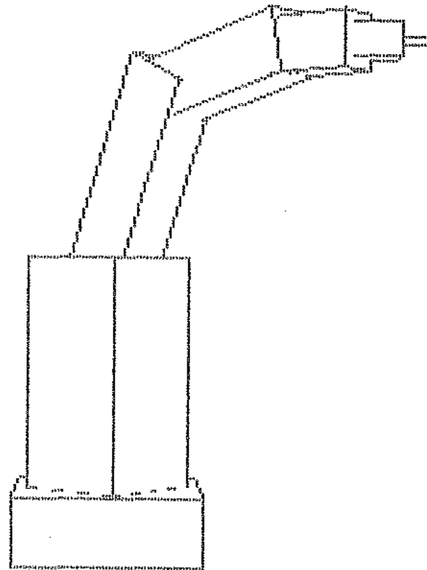


fig 13 b) c)

8. SAMMANFATTNING

Ett generellt tredimensionellt paket har utvecklats.
Det utför:

- perspektivtransformationer
- hiddensurface-beräkningar
- shading
- ändring av betraktelsepunkten

Animering av en industrirobot har gjorts. Robotens rörelse beskrives i ett kartesiskt koordinatsystem och ledernas vinklar beräknas.

9. REFERENSER

Newman W., Sproull R.:
Principles of Interactive Computer Graphics
2:a uppl., McGraw-Hill, 1979
särskilt kapitlen 20, 22-24

Paul R.:
Robot Manipulators: Mathematics, Programming, and Control
The MIT Press
kapitlen 1-3
Observera att i denna bok skall alla matriser transponeras för att få samma beskrivning som i Newman&Sproull. Dessutom är alla rotationer definierade åt motsatt håll.

Ekman T., Karlsson J.:
Pascal för dig som kan programmera.
Studentlitteratur, 1981

Sparr G.:
Linjär algebra 1
Sigma-Tryck, TLTH Lund, 1978

Appendix 1

```
{threed.pak}

{Author:Mikael Rignell
 Date:1982-09-04}

{reference:Newman W., Sproull R.:
           Principles of Interactive Computer Graphics
           Second Edition, McGraw-Hill, 1979
           chapters 20,22-24}

.FORWARD

{These are the primitives used to make three-dimensional
images, to perform the hidden-surface algorithm and to move
the viewingpoint.}

procedure LineTo3(xm,ym,zm:real);forward;
procedure MoveTo3(xm,ym,zm:real);forward;

procedure StartPolygon(xm,ym,zm:real);forward;
procedure NextVertex(xm,ym,zm:real);forward;
procedure LastVertex(xm,ym,zm:real);forward;

procedure Translate3(Tx,Ty,Tz:real);forward;
procedure Rotate3x(fi:real);forward;
procedure Rotate3y(fi:real);forward;
procedure Rotate3z(fi:real);forward;

procedure Pan3(theta:real);forward;
procedure Tilt3(fi:real);forward;
procedure Zoom3(Dchange:real);forward;
procedure Roll3(Zchange:real);forward;

procedure NewViewport3(vxl,vxr,vyb,vyt:real);forward;

procedure FrameColor(colno:integer);forward;
procedure InternalColor(colno:integer);forward;
procedure Reset3;forward;
procedure Shade(shadeon:boolean);forward;
procedure BackFaceRemoval(bfrem:boolean);forward;

.TYPE
matrix=array[1..4,1..4]of real;
pointer=^point;
point=record
        x,y,z:real;
        pre,suc:pointer;
      end;

.VAR
K,M,V,P,S,ZERO,UNIT:matrix;
```

```
{M is the transformationmatrix from the model coordinate
system to the world coordinate system. V transforms from the
world coordinate system to the eye coordinate system. P
transforms from the eye c.s. to the homogeneous c.s.. S
choses the viewport. K=M*V*P.}
xh,yh,zh,wh, xh1,yh1,zh1,wh1, xs1,ys1,zs1, xs2,ys2,zs2:real;
xs2pre,ys2pre,zs2pre:real;
orderno:integer;
firstvertexlist,vertexlist:pointer;
depth:array[0..511,0..511] of real;
SP,DP,FP:real;
framecol,internalcol:integer;
shading:boolean;
shadeconst: real;
backfacerem:boolean;
xpixelscale3, ypixelscale3, xpixeloffset3, ypixeloffset3:real;

.PROCEDURE
procedure MultiplyVectMat(xi,yi,zi,wi:real;var Mx:matrix;
                          var xo,yo,zo,wo:real);

begin
   xo:=xi*Mx[1,1]+yi*Mx[2,1]+zi*Mx[3,1]+wi*Mx[4,1];
   yo:=xi*Mx[1,2]+yi*Mx[2,2]+zi*Mx[3,2]+wi*Mx[4,2];
   zo:=xi*Mx[1,3]+yi*Mx[2,3]+zi*Mx[3,3]+wi*Mx[4,3];
   wo:=xi*Mx[1,4]+yi*Mx[2,4]+zi*Mx[3,4]+wi*Mx[4,4];
end;

procedure MultiplyMatMat(A,B:matrix;var C:matrix);
var i,j:integer;
begin
   for i:=1 to 4 do
      for j:=1 to 4 do
         C[i,j]:=A[i,1]*B[1,j]+A[i,2]*B[2,j]+A[i,3]*B[3,j]
                                               +A[i,4]*B[4,j];

end;

procedure MultMVP;
begin
   MultiplyMatMat(M,V,K);
   MultiplyMatMat(K,P,K);
end;

procedure Clip3(var x1,y1,z1,w1,x2,y2,z2,w2:real;
                         var outsideclipbox:boolean);
{From Newman&Sproull p. 360.}
   type edge=1..6;
        outcode=set of edge;
   var wc:array[1..2,1..6] of real;
       c1,c2:outcode;
       dx,dy,dz,dw,t,t1,t2:real;
       i:integer;
   procedure MakeWindowCoords(p:integer;x,y,z,w:real;
                                           var c:outcode);

      var i:integer;
      begin
```

```
      wc[p,1]:=w+x;
      wc[p,2]:=w-x;
      wc[p,3]:=w+y;
      wc[p,4]:=w-y;
      wc[p,5]:=z;
      wc[p,6]:=w-z;
      c:=[];
      for i:=1 to 6 do
         if wc[p,i]<0 then c:=c+[i];
   end;
begin
   MakeWindowCoords(1,x1,y1,z1,w1,c1);
   MakeWindowCoords(2,x2,y2,z2,w2,c2);
   if (c1*c2)=[] then
   begin
      outsideclipbox:=false;
      t1:=0;
      t2:=1;
      for i:=1 to 6 do
         if (wc[1,i]<0) or (wc[2,i]<0) then
         begin
            writeln('clipping has occured');
            t:=wc[1,i]/(wc[1,i]-wc[2,i]);
            if wc[1,i]<0 then
            begin
               if t>t1 then t1:=t;
            end
            else
            begin
               if t<t2 then t2:=t;
            end;
         end;
      if t2>=t1 then
      begin
         dx:=x2-x1;
         dy:=y2-y1;
         dz:=z2-z1;
         dw:=w2-w1;
         if t2<>1 then
         begin
            x2:=x1+t2*dx;
            y2:=y1+t2*dy;
            z2:=z1+t2*dz;
            w2:=w1+t2*dw;
         end;
         if t1<>0 then
         begin
            x1:=x1+t1*dx;
            y1:=y1+t1*dy;
            z1:=z1+t1*dz;
            w1:=w1+t1*dw;
         end;
      end;
   end
   else outsideclipbox:=true;
```

```
    {The line is completely outside the viewingbox.}
  end;

procedure Transform3(xm,ym,zm:real;var xs1,ys1,zs1,
                     xs2,ys2,zs2:real; var notwrite:boolean);
{The image is from the beginning described in the model
coordinate-system. Via the eye coordinate-system, the
homogenous coordinate-system and clipping its final
description is in the screen coordinate-system. This
procedure performs the transformation.}
var xh,yh,zh,wh,xh2,yh2,zh2,wh2,x1,y1,z1,w1,x2,y2,z2,w2:real;
begin
   MultiplyVectMat(xm,ym,zm,1,K,xh,yh,zh,wh);
   xh2:=xh;
   yh2:=yh;
   zh2:=zh;
   wh2:=wh;
   Clip3(xh1,yh1,zh1,wh1,xh,yh,zh,wh,notwrite);
   MultiplyVectMat(xh1,yh1,zh1,wh1,S,x1,y1,z1,w1);
   MultiplyVectMat(xh,yh,zh,wh,S,x2,y2,z2,w2);
   xh1:=xh2;
   yh1:=yh2;
   zh1:=zh2;
   wh1:=wh2;
   xs1:=x1/w1;
   ys1:=y1/w1;
   zs1:=z1/w1;
   xs2:=x2/w2;
   ys2:=y2/w2;
   zs2:=z2/w2;
end;

procedure BackFace;
var i:integer;
    helpp:pointer;
begin
   for i:=1 to orderno do
   begin
      helpp:=vertexlist;
      vertexlist:=vertexlist^.suc;
      PixelLine(round(helpp^.x),round(helpp^.y),
                round(vertexlist^.x),round(vertexlist^.y));
   end;
end;

procedure HiddenSurface;
{This procedure performs the depth-buffer algorithm
described in Newman&Sproull, p. 369. Lines and surfaces not
visible to the viewer are excluded. The procedure assumes
that the model consists of convex polygons. The verticies of
one polygon is inserted into a circular two way list, by the
procedures NextVertex and LastVertex. (The list is
initialized by the procedure StartPolygon.) Lastvertex calls
the hiddensurface procedure.}
label 10;
```

```
const maxintens = 15;
      shadeconst = 175;
var P,Q,R:pointer;
    PQ,QR:array[1..3] of real;
    A,B,C,D:real;
    left,up,down,helppointer:pointer;
    helporder,i,k,xhelpuptrunc,yhelpuptrunc,xhelpdowntrunc,
    Nextxtrunc,yhelpdowntrunc,xt,yt:integer;
    xmin,xhelpup,yhelpup,xhelpdown,yhelpdown,Nextx,Nexty,
    UpNumerator,UpDenominator,DeltaxUp,DeltayUp,
    DownNumerator,DownDenominator,DeltaxDown,DeltayDown,z,
    DeltaxMiddle,DeltayMiddle:real;
    DeltaUpChange,DeltaDownChange,UpNext,nextupsuc:boolean;
    intens: real;
    colorno,incolorno:integer;
    yhelpinternal:real;
    xhelpinternal:real;
    xscanup,yscanup,xscandown,yscandown:real;
    verticalline:boolean;
begin
   verticalline:=false;
   vertexlist:=firstvertexlist;
   helporder:=orderno-3;
   {orderno is the number of verticies.}
   if orderno < 3 then goto 10;
   {The equation of the polygon-plane in the form
   a*x+b*y+c*z+d=0, must be calculated. To do this you need
   one point in the plane (P), and two non-parallell vectors
   in the plane (PQ and QR).}
   P:=firstvertexlist;
   Q:=P^.suc;
   R:=Q^.suc;
   PQ[1]:=P^.x-Q^.x;
   PQ[2]:=P^.y-Q^.y;
   PQ[3]:=P^.z-Q^.z;
   QR[1]:=Q^.x-R^.x;
   QR[2]:=Q^.y-R^.y;
   QR[3]:=Q^.z-R^.z;
   C:=PQ[2]*QR[1]-PQ[1]*QR[2];
   if C < 0 then goto 10;
   {if C < 0 it means that the normal to the surface has a
   negative z-coordinate and this surface could not be seen
   finally. Due to the hardware, the positive z-direction is
   out from the screen. This is corrected when calling
   CmiDot.}
   if backfacerem then
   begin
      BackFace;
      goto 10;
   end;
   A:=PQ[3]*QR[2]-PQ[2]*QR[3];
   B:=PQ[1]*QR[3]-PQ[3]*QR[1];
   D:=-(A*P^.x+B*P^.y+C*P^.z);
   if (abs(A)<1.0E-6) and (abs(B)<1.0E-6)
                      and (abs(C)<1.0E-6) then
```

```
begin
   writeln('The first three verticies are
                                    on the same line.');
   Goto 10;
end;
for i:=1 to helporder do
begin
   vertexlist:=vertexlist^.suc;
   with vertexlist^ do
   begin
      if abs(A*x+B*y+C*z+D))0.01 then
      begin
         writeln('A point doesn''t lie in the plane.');
         Goto 10;
      end;
   end;
end;

{ shading }
if shading then
begin
   intens :=   C/shadeconst / sqrt(A*A+B*B+
                C/shadeconst*C/shadeconst);
   colorno := round(intens*maxintens);
   incolorno := round(intens*maxintens);
end
else
begin
   colorno := framecol;
   incolorno := internalcol;
end;

if abs(C))1.0E-6 then
{The plane is described in the form
     z=-(a/c)*x-(b/c)*y-d/c.}
begin
   A:=A/C;
   B:=B/C;
   D:=D/C;
end
else
begin
   A:=A*1.0E6;
   B:=B*1.0E6;
   D:=D*1.0E6;
end;
helppointer:=firstvertexlist;
left:=firstvertexlist;
xmin:=1000;
for i:=1 to orderno do
{The point with the smallest x-coordinate should be found.
If there are more than one, the first one found is chosen.}
begin
   if helppointer^.x(xmin then
   begin
```

```
        xmin:=helppointer^.x;
        left:=helppointer;
     end;
     helppointer:=helppointer^.suc;
  end;
  if left^.suc^.y>left^.pre^.y then
  {Which way to go in the list to be on the upside
  respectively on the downside of the polygon, is found out.}
  begin
     up:=left^.suc;
     down:=left^.pre;
     nextupsuc:=true;
  end
  else
  begin
     up:=left^.pre;
     down:=left^.suc;
     nextupsuc:=false;
  end;
  CmiCol(colorno);
  DeltaUpChange:=true;
  DeltaDownChange:=true;
  for k:=1 to orderno-1 do
  {Is the next vertex on the upside
                           or downside of the polygon?}
  begin
     if up^.x<down^.x then
     begin
        Nextx:=up^.x;
        Nexty:=up^.y;
        UpNext:=true;
     end
     else
     begin
        Nextx:=down^.x;
        Nexty:=down^.y;
        UpNext:=false;
     end;
     if DeltaUpChange then
     {The slope of the upper edge have changed.}
     begin
        xhelpup:=left^.x{+0.5};
        xhelpuptrunc:=trunc(xhelpup);
        yhelpup:=left^.y{+0.5};
        yhelpuptrunc:=trunc(yhelpup);
        xscanup:=xhelpup;
        yscanup:=yhelpup;
        UpNumerator:=up^.y-left^.y;
        UpDenominator:=up^.x-left^.x;
        if abs(UpNumerator)>UpDenominator then
        begin
           DeltaxUp:=UpDenominator/abs(UpNumerator);
           DeltayUp:=UpNumerator/abs(UpNumerator);
        end
        else
```

```
   begin
      DeltaxUp:=1.0;
      if UpDenominator=0 then DeltayUp:=1 else
         DeltayUp:=UpNumerator/UpDenominator;
   end;
   DeltaUpChange:=false;
end;
if DeltaDownChange then
{The slope of the lower edge have changed.}
begin
   xhelpdown:=left^.x{+0.5};
   xhelpdowntrunc:=trunc(xhelpdown);
   yhelpdown:=left^.y{+0.5};
   yhelpdowntrunc:=trunc(yhelpdown);
   xscandown:=xhelpdown;
   yscandown:=yhelpdown;
   DownNumerator:=down^.y-left^.y;
   DownDenominator:=down^.x-left^.x;
   if abs(DownNumerator)>DownDenominator then
   begin
      DeltaxDown:=DownDenominator/abs(DownNumerator);
      DeltayDown:=DownNumerator/abs(DownNumerator);
   end
   else
   begin
      DeltaxDown:=1.0;
      if DownDenominator=0 then DeltayDown:=1 else
         DeltayDown:=DownNumerator/DownDenominator;
   end;
   DeltaDownChange:=false;
end;
Nextxtrunc:=trunc(Nextx);
for xt:=trunc(left^.x) to Nextxtrunc do
{For each x-coordinate on an edge, (each xt), you have
a specific y-coordinate.But since the screen can only
take integer values, one x-value can have more than
one y-value. This happens if the slope of an edge is
more than 45 degrees. Then DeltaxUp or DeltaxDown are
less than one. For each point on the edge, the
z-coordinate is set and if the z-value is smaller than
the one in the depth-matrix,the point is set.}
begin
   if not(verticalline) then
   begin
      xscanup:=xhelpup;
      yscanup:=yhelpup;
      xscandown:=xhelpdown;
      yscandown:=yhelpdown;
   end;
   while xhelpuptrunc=xt do
   {If the slope is less than 45 degrees, these
   computations are made only once for each
   x-coordinate. If the slope is more than 45 degrees,
   they could be performed more than once.}
   begin
```

```
      z:=-A*xhelpup-B*yhelpup-D;
      if z<=depth[xt,yhelpuptrunc] then
      begin
         CmiDot(xt,yScreenMax-yhelpuptrunc);
         depth[xt,yhelpuptrunc]:=z;
      end;
      if yhelpup<yscanup then
      begin
         xscanup:=xhelpup;
         yscanup:=yhelpup;
      end;
      xhelpup:=xhelpup+DeltaxUp;
      xhelpuptrunc:=trunc(xhelpup);
      yhelpup:=yhelpup+DeltayUp;
      yhelpuptrunc:=trunc(yhelpup);
      if DeltaxUp<1 then
         if DeltayUp > 0 then
         begin
            if yhelpuptrunc > trunc(up^.y) then
                        xhelpuptrunc:=xhelpuptrunc+1;
         end
         else
         begin
            if yhelpuptrunc < trunc(up^.y) then
                        xhelpuptrunc:=xhelpuptrunc+1;
         end;
         {If we reach a vertex,
                  we must stop at the right y-value.}
   end;
   while xhelpdowntrunc=xt do
   {Same as above but with the lower edge.}
   begin
      z:=-A*xhelpdown-B*yhelpdown-D;
      if z<=depth[xt,yhelpdowntrunc] then
      begin
         CmiDot(xt,yScreenMax-yhelpdowntrunc);
         depth[xt,yhelpdowntrunc]:=z;
      end;
      if yhelpdown>yscandown then
      begin
         xscandown:=xhelpdown;
         yscandown:=yhelpdown;
      end;
      xhelpdown:=xhelpdown+DeltaxDown;
      xhelpdowntrunc:=trunc(xhelpdown);
      yhelpdown:=yhelpdown+DeltayDown;
      yhelpdowntrunc:=trunc(yhelpdown);
      if DeltaxDown<1 then
         if DeltayDown < 0 then
         begin
            if yhelpdowntrunc < trunc(down^.y) then
                     xhelpdowntrunc:=xhelpdowntrunc+1;
         end
         else
         begin
```

```
            if yhelpdowntrunc > trunc(down^.y) then
                  xhelpdowntrunc:=xhelpdowntrunc+1;
         end;
   end;
   if not(trunc(yscanup)=trunc(yscandown)) then
   begin
      DeltaxMiddle:=(xscanup-xscandown)/
                    (trunc(yscanup)-trunc(yscandown));
      DeltayMiddle:=(yscanup-yscandown)/
                    (trunc(yscanup)-trunc(yscandown));
   end;

   { scanconversion }

   if xt < Nextxtrunc then
   {Don't do scanconversion beyond the next vertex.}
   begin
      CmiCol(incolorno);
      xhelpinternal:=xscandown+DeltaxMiddle;
      yhelpinternal:=yscandown+DeltayMiddle;
      for yt:=trunc(yscandown)+1 to trunc(yscanup)-1 do
      begin
         z:=-A*xhelpinternal-B*yhelpinternal-D;
         if z<depth[xt,yt] then
         begin
            CmiDot(xt,yScreenMax-yt);
            depth[xt,yt]:=z;
         end;
         xhelpinternal:=xhelpinternal+DeltaxMiddle;
         yhelpinternal:=yhelpinternal+DeltayMiddle;
      end;
      CmiCol(colorno);
   end;
   if trunc(left^.x) = Nextxtrunc then
   begin
      if (trunc(left^.x) = trunc(left^.pre^.x)) or
         (trunc(left^.x) = trunc(left^.suc^.x))
                           then verticalline:=true
                           else verticalline:=false;
   end
   else
   begin
      verticalline:=false;
   end;
end;
if UpNext then
{The vertex with the next x-coordinate in turn
                                 is on the upside.}
begin
   left:=up;
   if nextupsuc then up:=up^.suc
                else up:=up^.pre;
   DeltaUpChange:=true;
end
else
```

```
        {Next vertex on the downside.}
        begin
           left:=down;
           if nextupsuc then down:=down^.pre
                             down:=down^.suc;
           DeltaDownChange:=true;
        end;
        z:=-A*Nextx-B*Nexty-D;
        if z < depth[Nextxtrunc,trunc(Nexty)] then
        {Due to the truncation, you might miss a vertexpoint.}
        begin
           CmiDot(Nextxtrunc,yScreenMax-trunc(Nexty));
           depth[Nextxtrunc,trunc(Nexty)]:=z;
        end;
     end;
10:for i:=1 to orderno-1 do
     begin
        vertexlist:=vertexlist^.suc;
        dispose(vertexlist^.pre);
     end;
     dispose(vertexlist);
end;

procedure insert(e:pointer);
{Inserts a point into the cicular two way list.}
begin
   if firstvertexlist=nil then
   begin
      firstvertexlist:=e;
      firstvertexlist^.pre:=firstvertexlist;
      firstvertexlist^.suc:=firstvertexlist;
   end
   else
   begin
      firstvertexlist^.pre^.suc:=e;
      e^.pre:=firstvertexlist^.pre;
      e^.suc:=firstvertexlist;
      firstvertexlist^.pre:=e;
   end;
end;

procedure LineTo3{xm,ym,zm:real};
{A line is drawn from the point in the last MoveTo3
LineTo3 to the one given here.}
var xs1r,ys1r,xs2r,ys2r:integer;
    lineoutside:boolean;
begin
   Transform3(xm,ym,zm,xs1,ys1,zs1,xs2,ys2,zs2,lineoutside);
   if not(lineoutside) then
   begin
      xs1r:=round(xs1);
      ys1r:=round(ys1);
      xs2r:=round(xs2);
      ys2r:=round(ys2);
      PixelLine(xs1r,ys1r,xs2r,ys2r);
```

```
   end;
end;

procedure MoveTo3(xm,ym,zm:real);
{The next line should begin at this point.}
begin
   MultiplyVectMat(xm,ym,zm,1,K,xh,yh,zh,wh);
   xh1:=xh;
   yh1:=yh;
   zh1:=zh;
   wh1:=wh;
end;

procedure StartPolygon(xm,ym,zm:real);
{Initiates the list and performs the first steps (those
before the clipping), in the transformation from
model-coordinates to screen-coordinates. The
clipping-algorithm must have both the end-points of a line.}
begin
   new(firstvertexlist);
   firstvertexlist:=nil;
   orderno:=0;
   MultiplyVectMat(xm,ym,zm,1,K,xh,yh,zh,wh);
   xh1:=xh;
   yh1:=yh;
   zh1:=zh;
   wh1:=wh;
   xs2pre:=1000;
   ys2pre:=1000;
   zs2pre:=1000;
end;

procedure NextVertex(xm,ym,zm:real);
{Puts one (or sometimes two) point(s) into the list after
transformation. The coordinates are translated and scaled as
to be visible on the screen. The y-coordinates are also
corrected as not to be shown upside down.}
label 7;
var e:pointer;
    noline:boolean;
begin
   Transform3(xm,ym,zm,xs1,ys1,zs1,xs2,ys2,zs2,noline);
   if noline then goto 7;

   if (xs1<>xs2pre) or (ys1<>ys2pre) or (zs1<>zs2pre) then
  {If the startingpoint of the new line isn't the same as the
  last point in the previous line, the startingpoint is
  inserted.}
   begin
      new(e);
      with e^ do
      begin
         x:=xs1;
         y:=ys1;
         z:=zs1;
```

```
      end;
      insert(e);
      orderno:=orderno+1;
   end;
   new(e);
   with e^ do
   begin
      x:=xs2;
      y:=ys2;
      z:=zs2;
   end;
   insert(e);
   orderno:=orderno+1;
   xs2pre:=xs2;
   ys2pre:=ys2;
   zs2pre:=zs2;
 7:   ;
end;

procedure LastVertex(xm,ym,zm:real);
{Puts the last vertex into the list and calls HiddenSurface.}
begin
   NextVertex(xm,ym,zm);
   HiddenSurface;
end;

procedure Translate3(Tx,Ty,Tz:real);
{Translates the model coordinate-system.}
var T:matrix;
    i,j:integer;
begin
   T:=UNIT;
   T[4,1]:=Tx;
   T[4,2]:=Ty;
   T[4,3]:=Tz;
   MultiplyMatMat(T,M,M);
   MultMVP;
end;

procedure Rotate3x(fi:real);
{The model coordinate-system is rotated around the x-axis.}
var T:matrix;
    i,j:integer;
    cosfi,sinfi:real;
begin
   fi:=3.141592654*fi/180;
   cosfi:=cos(fi);
   sinfi:=sin(fi);
   T:=ZERO;
   T[1,1]:=1;
   T[2,2]:=cosfi;
   T[2,3]:=-sinfi;
   T[3,2]:=sinfi;
   T[3,3]:=cosfi;
   T[4,4]:=1;
```

```
   MultiplyMatMat(T,M,M);
   MultMVP;
end;

procedure Rotate3y{fi:real};
{Rotation of the model coordinatesystem around the y-axis.}
var T:matrix;
     i,j:integer;
     cosfi,sinfi:real;
begin
   fi:=3.141592654*fi/180;
   cosfi:=cos(fi);
   sinfi:=sin(fi);
   T:=ZERO;
   T[1,1]:=cosfi;
   T[1,3]:=sinfi;
   T[2,2]:=1;
   T[3,1]:=-sinfi;
   T[3,3]:=cosfi;
   T[4,4]:=1;
   MultiplyMatMat(T,M,M);
   MultMVP;
end;

procedure Rotate3z{fi:real};
{Rotates the model coordinate-system around the z-axis.}
var T:matrix;
     i,j:integer;
     cosfi,sinfi:real;
begin
   fi:=3.141592654*fi/180;
   cosfi:=cos(fi);
   sinfi:=sin(fi);
   T:=ZERO;
   T[1,1]:=cosfi;
   T[1,2]:=-sinfi;
   T[2,1]:=sinfi;
   T[2,2]:=cosfi;
   T[3,3]:=1;
   T[4,4]:=1;
   MultiplyMatMat(T,M,M);
   MultMVP;
end;

procedure SetViewBox(SP,DP,FP:real);
begin
   P[3,3]:=SP/(DP*(1-DP/FP));
   P[3,4]:=SP/DP;
   P[4,3]:=-SP/(1-DP/FP);
   MultMVP;
end;

procedure Pan3{theta:real};
{The viewer rotates around his vertical axis.}
var thetarad,Csin,Ccos:real; H:matrix;
```

```
begin
   thetarad:=3.141592654*theta/180;
   Csin:=sin(thetarad);
   Ccos:=cos(thetarad);
   H:=UNIT;
   H[1,1]:=Ccos;
   H[1,3]:=Csin;
   H[3,1]:=-Csin;
   H[3,3]:=Ccos;
   MultiplyMatMat(V,H,V);
   MultMVP;
end;

procedure Tilt3{fi:real};
{The viewer rotates around the axis parallell
                                   to the horizon.}
var firad,Ksin,Kcos:real; i:integer; H:matrix;
begin
   firad:=-3.141592654*fi/180;
   Ksin:=sin(firad);
   Kcos:=cos(firad);
   H:=UNIT;
   H[2,2]:=Kcos;
   H[2,3]:=-Ksin;
   H[3,2]:=Ksin;
   H[3,3]:=Kcos;
   MultiplyMatMat(V,H,V);
   MultMVP;
end;

procedure Roll3{Zchange:real};
{The viewer moves forwards or backwards.}
var H:matrix;
begin
   H:=UNIT;
   H[4,3]:=-Zchange;
   MultiplyMAtMat(V,H,V);
   MultMVP;
end;

procedure Zoom3{Dchange:real};
begin
   DP:=DP+Dchange;
   SetViewBox(SP,DP,FP);
end;

procedure NewViewPort3{vxl,vxr,vyb,vyt:real};
begin
   S[1,1]:=(vxr-vxl)/2*xpixelscale3;
   S[2,2]:=(vyt-vyb)/2*ypixelscale3;
   S[4,1]:=(vxl+vxr)/2+xpixeloffset3;
   S[4,2]:=(vyb+vyt)/2+ypixeloffset3;
end;

procedure FrameColor{colno:integer};
```

```
begin
   framecol:=colno;
end;

procedure InternalColor(colno:integer);
begin
   internalcol:=colno;
end;

procedure Shade(shadeon: boolean);
begin
   shading := shadeon;
   if shadeon then BlueShade
              else TwoPlanes;
end;

procedure BackFaceRemoval(bfrem:boolean);
begin
   backfacerem:=bfrem;
end;

procedure Reset3;
begin
   M:=UNIT;
   MultMVP;
end;

procedure Setdepth;
var i,j:integer;
begin
   for i:=0 to 511 do depth[0,i]:=10;
   for i:=1 to 511 do depth[i]:=depth[0];
end;

procedure DefineScreen3;
  const
  CmiXsize = 511;  CmiYsize = 511;
  screenwidth = 280;  screenheight = 200;

  begin
  xpixelscale3 := (CmiXSize + 1) / ScreenWidth;
  ypixelscale3 := (CmiYSize + 1) / ScreenHeight;

  xpixeloffset3 := CmiXSize / 2;
  ypixeloffset3 := CmiYSize / 2;
  end;

procedure InitThreeDim;
var i,j:integer;
begin
   for i:=1 to 4 do
   for j:=1 to 4 do
   begin
      ZERO[i,j]:=0;
   end;
```

```
    UNIT:=ZERO;
    for i:=1 to 4 do UNIT[i,i]:=1;
    M:=UNIT;
    V:=ZERO;
    P:=ZERO;
    S:=ZERO;
    SP:=75;
    DP:=300;
    FP:=800;
    V[1,1]:=1;
    V[2,3]:=1;
    V[3,2]:=1;
    V[4,4]:=1;
    Roll3(-400);
    P[1,1]:=1;
    P[2,2]:=1;
    SetViewBox(SP,DP,FP);
    DefineScreen3;
    S[3,3]:=1;
    S[4,4]:=1;
    NewViewPort3(-75,75,-75,75);
    MultMVP;
    shading:=false;
    backfacerem:=false;
    internalcol:=0;
    framecol:=1;
    CmiErase;
end;

.INIT
InitThreeDim;

.END
```

```
{robot.pak describes an industrial robot and
 its movement in a cartesian coordinate system}

{Author: Mikael Rignell
 Date: 1982-09-04}

{Reference:
   Paul R.:
   Robot Manipulators: Mathematics, Programming and Control
   The MIT Press}

.PROGRAM

program Robot(input,output,outfile);


.CONST
pi=3.141592654;


.VAR
outfile:text;
ch:char;
number,newnumber,Znewnumber:real;
i,j,forhelp,cno:integer;
vxl,vxr,vyb,vyt:real;
joystickvalues:joysticktype;
px,py,pz,nx,ny,nz:real;
pxold,pyold,pzold, nxold,nyold,nzold:real;
a1,a2,a3,a4,a5,z0:real;
theta1,theta2,theta3,theta4,theta5,theta6:real;
model,notjumpout,notmoving:boolean;
shad:boolean;
bafare:boolean;


.PROCEDURE

function arctan2(y,x:real):real;
var fi:real;
begin
   if x=0 then
   begin
      if y>0 then fi:=pi/2
             else fi:=-pi/2;
   end
   else
   begin
      fi:=arctan(y/x);
      if (x<0) and (y>0) then fi:=fi+pi;
      if (x<0) and (y<0) then fi:=fi-pi;
   end;
   arctan2:=fi;
end;
```

```
procedure box(x2,y2,z2:real);
begin
    MoveTo3(0,0,0);
    LineTo3(0,0,z2);
    LineTo3(x2,0,z2);
    LineTo3(x2,0,0);
    LineTo3(0,0,0);
    LineTo3(0,y2,0);
    LineTo3(0,y2,z2);
    LineTo3(x2,y2,z2);
    LineTo3(x2,y2,0);
    LineTo3(0,y2,0);
    MoveTo3(0,y2,z2);
    LineTo3(0,0,z2);
    MoveTo3(x2,0,z2);
    LineTo3(x2,y2,z2);
    MoveTo3(x2,y2,0);
    LineTo3(x2,0,0);
end;

procedure Rectanglex(xb1,yb1,zb1,yb3,zb3:real);
begin
    StartPolygon(xb1,yb1,zb1);
    NextVertex(xb1,yb3,zb1);
    NextVertex(xb1,yb3,zb3);
    LastVertex(xb1,yb1,zb3);
end;

procedure Rectangley(xb1,yb1,zb1,xb3,zb3:real);
begin
    StartPolygon(xb1,yb1,zb1);
    NextVertex(xb1,yb1,zb3);
    NextVertex(xb3,yb1,zb3);
    LastVertex(xb3,yb1,zb1);
end;

procedure Rectanglez(xb1,yb1,zb1,xb3,yb3:real);
begin
    StartPolygon(xb1,yb1,zb1);
    NextVertex(xb3,yb1,zb1);
    NextVertex(xb3,yb3,zb1);
    LastVertex(xb1,yb3,zb1);
end;

procedure Block(length,width,height:real);
begin
    if (notmoving or bafare) then
    begin
        Rectanglex(0,0,0,width,height);
        Rectanglex(length,width,0,0,height);
        Rectangley(0,0,0,length,height);
        Rectangley(0,width,height,length,0);
        Rectanglez(0,0,0,length,width);
        Rectanglez(length,0,height,0,width);
    end
```

```
   else
   begin
      box(length,width,height);
   end;
end;

procedure NewAngles(var outsidereach:boolean);
{used when moving the robot with 6 degrees of freedom
 Inparameters : px py pz (position)
                nx ny nz (direction)
 Outparameters : theta1,theta2,theta3,theta4,theta5 }
label 5,6;
var pxp,pzp, pxpp,pzpp, pxh,pyh,pzh,nxh,nyh,nzh:real;
    norm:real;
    alfa1,alfa2,alfa3,alfa4,alfa5,alfa234,alfa23:real;
    C1,S1,C2,S2,C23,S23,C234,S234,C3,S3,S4,C4,C5,S5:real;
begin
   outsidereach:=false;
   norm:=sqrt(sqr(nx)+sqr(ny)+sqr(nz));
   nx:=nx/norm;
   ny:=ny/norm;
   nz:=nz/norm;

   alfa1:=arctan2((a5*ny-py),(px-a5*nx));
   S1:=sin(alfa1);
   C1:=cos(alfa1);
   {alfa1 is the robot's rotation around the foundation.
         -pi<= alfa1 <=+pi }

   S5:=-(S1*px+C1*py)/a5;
   if abs(S5) > 1 then goto 5;
   C5:=sqrt(1-sqr(S5));
   alfa5:=arctan2(S5,C5);
   {alfa5 is the angle between the two outer arms. (Rotation
    around an z-axis).      -pi/2 <= alfa5 <= +pi/2 }

   alfa234:=arctan2(nz,C1*nx-S1*ny);
   C234:=cos(alfa234);
   S234:=sin(alfa234);
   {alfa234=alfa2+alfa3+alfa4. -pi <= alfa234 <= +pi }

   pxp:=C1*px-S1*py-(C5*a5+a4)*C234;
   pzp:=pz-z0-(C5*a5+a4)*S234-a1;
   S3:=(sqr(pxp)+sqr(pzp)-sqr(a3)-sqr(a2))/(2*a2*a3);
   if abs(S3) > 1 then goto 5;
   C3:=sqrt(1-sqr(S3));
   alfa3:=arctan2(S3,C3);
   if alfa3 < -3*pi/8 then goto 5;
   {alfa3 is the angle between the biggest parts of the arm.
   If S3>1 it means that the point cannot be reached, at
   least not with the desired direction. -3*pi/8 <= alfa3 <=
   +pi/2 . The lower limit because the outer parts of the
   arm shouldn't fold into the first part.}

   pxpp:=C234*(C1*px-S1*py)+S234*(pz-z0-a1)-C5*a5-a4;
```

```
   pzpp:=-S234*(C1*px-S1*py)+C234*(pz-z0-a1);
   alfa4:=arctan2(pxpp*a2*C3-pzpp*(a2*S3+a3),
                             pzpp*a2*C3+pxpp*(a2*S3+a3));
   if abs(alfa4) > 9*pi/10 then goto 5;
   {alfa4 is is the angle between the second
    and the third part of the arm.
    -9*pi/10 <= alfa4 <= +9*pi/10 }

   alfa2:=alfa234-alfa4-alfa3;
   if alfa2<-pi/2 then alfa2:=alfa2+pi;
   if alfa2>pi/2 then alfa2:=alfa2-pi;
   S2:=sin(alfa2);
   C2:=cos(alfa2);
   {alfa2 is the angle between the body
    and the first part of the arm.
    -pi/2 <= alfa2 <= +pi/2 }

   C23:=cos(alfa2+alfa3);
   S23:=sin(alfa2+alfa3);
   {Calculate the exact expressions for the coordinates
                                      and the directions.}
   pxh:=C5*a5*C234*C1+a4*C234*C1+a3*C23*C1-a5*S5*S1-a2*S2*C1;
   pyh:=-C5*a5*C234*S1-a4*C234*S1-a3*C23*S1-a5*S5*C1+a2*S2*S1;
   pzh:=C5*a5*S234+a4*S234+a3*S23+a2*C2+a1+z0;
   nxh:=C5*C234*C1-S5*S1;
   nyh:=-C5*C234*S1-S5*C1;
   nzh:=C5*S234;

   if (abs(px-pxh) > 0.1) or (abs(py-pyh) > 0.1) or
      (abs(pz-pzh) > 0.1) or (abs(nx-nxh) > 0.05) or
      (abs(ny-nyh) > 0.05) or (abs(nz-nzh) > 0.05)
                                               then goto 5;

   theta1:=180*alfa1/pi;
   theta2:=180*alfa2/pi;
   theta3:=180*alfa3/pi;
   theta4:=180*alfa4/pi;
   theta5:=180*alfa5/pi;
   goto 6;
 5:outsidereach:=true;
   {If we reach point 5 it means that the desired point
   cannot be reached. Then we must give back px,py,pz or
   nx,ny,nz values corresponding to a point that can be
   reached, as not to be trapped.}

 6:  ;
end;

procedure InitPos;
begin
   nx:=1;
   ny:=0;
   nz:=0;
   px:=64;
   py:=0;
```

```
   pz:=40;
   z0:=-40;
   a1:=40;
   a2:=40;
   a3:=30;
   a4:=18;
   a5:=16;
   theta1:=0;
   theta2:=0;
   theta3:=0;
   theta4:=0;
   theta5:=0;
   theta6:=0;
end;

procedure Robot6(fi1,fi2,fi3,fi4,fi5,fi6:real);
begin
    Reset3;
    Translate3(-21,-21,z0-13);
    Block(42,42,13);  {foundation}

    Translate3(21,21,13);
    Rotate3z(fi1);
    Translate3(-13,-13,0);
    Block(26,26,a1+5);  {body}

    Translate3(13,13,a1);
    Rotate3y(fi2);
    Translate3(-7,-7,-5);
    Block(14,14,a2+10);  {first arm}

    Translate3(7,7,a2+5);
    Rotate3y(fi3);
    Translate3(-5,-6,-6);
    Block(a3+9,12,12);  {second arm}

    Translate3(a3+5,6,6);
    Rotate3y(fi4);
    Translate3(-4,-5,-5);
    Block(a4+6,10,10);  {third arm}

    Translate3(a4+4,5,5);
    Rotate3z(fi5);
    Translate3(-2,-3,-3);
    Block(a5-4,6,6);  {fourth arm}
    Translate3(a5-4,3,3);
    Rotate3x(fi6);
    Translate3(0,-0.5,-0.5);
    Block(6,1,1);      {tip}
    Translate3(6,0.5,0.5);
 end;

 procedure MoveRobot(cx,cy,cz,dx,dy,dz:real);
 {All motion of the robot must be done
  through this procedure.}
```

```
var oldtheta1,oldtheta2,oldtheta3,oldtheta4,
     oldtheta5,oldtheta6:real;
     outsidereachmr:boolean;
begin
   outsidereachmr:=false;
   oldtheta1:=theta1;
   oldtheta2:=theta2;
   oldtheta3:=theta3;
   oldtheta4:=theta4;
   oldtheta5:=theta5;
   px:=cx;
   py:=cy;
   pz:=cz;
   nx:=dx;
   ny:=dy;
   nz:=dz;
   NewAngles(outsidereachmr);
   if outsidereachmr then
   begin
      theta1:=oldtheta1;
      theta2:=oldtheta2;
      theta3:=oldtheta3;
      theta4:=oldtheta4;
      theta5:=oldtheta5;
      px:=pxold;
      py:=pyold;
      pz:=pzold;
      nx:=nxold;
      ny:=nyold;
      nz:=nzold;
   end;
   CmiErase;
   Robot6(theta1,theta2,theta3,theta4,theta5,theta6);
   SwapPlane;
end;

procedure Weld;
var deltax,deltay:array[1..4] of real;
     i,j:integer;
     cx,cy,cz1,cz2,dx,dy,dz:real;
begin
   deltax[1]:=0;
   deltax[2]:=-4;
   deltax[3]:=0;
   deltax[4]:=4;
   deltay[1]:=4;
   deltay[2]:=0;
   deltay[3]:=-4;
   deltay[4]:=0;
   cx:=40;
   cy:=-40;
   cz1:=-40;
   cz2:=-53;
   dx:=0.00;
   dy:=0.00;
```

```
    dz:=-1;
    for i:=1 to 4 do
    begin
       for j:=1 to 20 do
       begin
          cx:=cx+deltax[i];
          cy:=cy+deltay[i];
          MoveRobot(cx,cy,cz1,dx,dy,dz);
          MoveRobot(cx,cy,cz2,dx,dy,dz);
          MoveRobot(cx,cy,cz1,dx,dy,dz);
       end;
    end;
    MoveRobot(64,0,40,1,0,0);
 end;

.MAIN
    begin
       TwoPlanes;
       InitPos;
       forhelp:=10;
       bafare:=false;
       shad:=false;
       notjumpout:=false;
       notmoving:=false;
       model:=true;
       CmiCol(1);
       DrawBackground;
       MoveRobot(px,py,pz,nx,ny,nz);
       while true do
       begin
          write('>');
          read(ch);
          if ch='l' then
          begin
             writeln('All commands are in the form : ',
                          'small character, real number');
             writeln('a : hardcopy');
             writeln('b : backfaceremoval');
             write('c : changed mode, the joysticks  ',
                          'changes the orientation');
             write('instead of the movement ',
                      'in the cartesian coordinate');
             writeln('system or vice versa');
             writeln('f : perform the hidden-surface ',
                                        'algorithm');
             writeln('h : the viewer moves horisontally');
             writeln('i : change the colour of the model');
             writeln('j : change the colour of the edges  ',
                                      'of the model');
             writeln('k : number of snapshots');
             writeln('l : gives the list of commands');
             writeln('m : the robot is controlled  ',
                                   'by the joysticks');
             write('o : the viewer rotates around the axis  ',
                                    'pointed at the ');
```

```
    writeln('model, number=degrees');
    writeln('p : panning motion of the viewer, ',
                                    'number=degrees');
    writeln('r : the viewer rolls  ',
                       '(moves backwards or forwards)');
    writeln('s : shading');
    writeln('t : the viewer tilts, number=degrees');
    writeln('v : the viewer moves vertically');
    writeln('w : a πoint-welding program is called');
    writeln('z : the viewer zooms');
end
else
if ch='a' then
begin
    DrawForeground;
    PrintExactScreen;
    DrawBackground;
end
else
if ch='f' then
begin
    bafare:=false;
    BackFaceRemoval(bafare);
    notmoving:=true;
    Setdepth;
    CmiErase;
    Robot6(theta1,theta2,theta3,theta4,theta5,theta6);
    SwapPlane;
    notmoving:=false;
end
else
if ch='s' then
begin
    shad:=not(shad);
    Shade(shad);
end
else
if ch='b' then
begin
    bafare:=not(bafare);
    BackFaceRemoval(bafare);
end
else
if ch='m' then
begin
    notjumpout:=true;
    while notjumpout do
    begin
        pxold:=px;
        pyold:=py;
        pzold:=pz;
        nxold:=nx;
        nyold:=ny;
        nzold:=nz;
        JoySticks(joystickvalues);
```

```
        With joystickvalues do
        begin
           if def[0] then
           begin
              if model then px:=px+values[0]/3000
                       else nx:=nx+values[0]/10000;
           end;
           if def[1] then
           begin
              if model then py:=py+values[1]/3000
                       else ny:=ny+values[1]/10000;
           end;
           if def[2] then
           begin
              if model then pz:=pz+values[2]/3000
                       else nz:=nz+values[2]/10000;
           end;
           if def[3] then theta6:=theta6+
                                    values[3]/3000;
           if def[17] then if values[17]=1 then
                                 notjumpout:=false;
           if def[18] then if values[18]=1 then
                                 notjumpout:=false;
           if def[19] then if values[19]=1 then
                                 notjumpout:=false;
           if def[20] then if values[20]=1 then
                                 notjumpout:=false;
        end;
        MoveRobot(px,py,pz,nx,ny,nz);
     end;
  end
  else
  if ch='c' then
  begin
     model:=not model;
  end
  else
  if ch='w' then
  begin
     Weld;
  end
  else
  begin
     read(number);
     if ch='i' then
     begin
        cno:=trunc(number);
        InternalColor(cno);
     end
     else
     if ch='j' then
     begin
        cno:=trunc(number);
        FrameColor(cno);
     end
```

```
          else
          if ch='k' then
          begin
             forhelp:=trunc(number);
          end
          else
          begin
             newnumber:=number/forhelp;
             Znewnumber:=sqr(number);
             for i:=1 to forhelp do
             begin
                case ch of
                  'p' : Pan3(newnumber);
                  't' : Tilt3(newnumber);
                  'z' : Zoom3(Znewnumber);
                  'r' : Roll3(number);
                  'h' : begin
                           Pan3(90);
                           Roll3(number);
                           Pan3(-90);
                        end;
                  'v' : begin
                           Tilt3(90);
                           Roll3(number);
                           Tilt3(-90);
                        end;
                  'o' : begin
                           Tilt3(90);
                           Pan3(newnumber);
                           Tilt3(-90);
                        end;
                end;
                Clearscreen;
                Robot6(theta1,theta2,theta3,
                       theta4,theta5,theta6);
                SwapPlane;
             end;
          end;
       end;
       readln;
    end;
  end;

.END
```

Appendix 2

Ur M-matrisen för robotarmens spets får man spetsens läge och orientering. I procedure Robot6 sker följande translationer och rotationer av modellkoordinatsystemet.

Translate3(-21,-21,z_0-13) Translate3(21,21,13)	Translate3(0,0,z_0)
Rotate3z(φ1)	
Translate3(-13,-13,0) Translate3(13,13,a1)	Translate3(0,0,a1)
Rotate3y(φ2)	
Translate3(-7,-7,-5) Translate3(7,7,a2+5)	Translate3(0,0,a2)
Rotate3y(φ3)	
Translate3(-5,-6,-6) Translate3(a3+5,6,6)	Translate3(a3,0,0)
Rotate3y(φ4)	
Translate3(-4,-5,-5) Translate3(a4+4,5,5)	Translate3(a4,0,0)
Rotate3z(φ5)	
Translate3(-2,-3,-3) Translate3(a5-4,3,3)	Translate3(a5-6,0,0)
Rotate3x(φ6)	
Translate3(0,-0.5,-0.5) Translate3(6,0.5,0.5)	Translate3(6,0,0)

Anm. De lokala variablerna i Robot6 kallas φ1-φ6 medan de globala kalla θ1-θ6.

Beteckningarna S1=sinθ1, C1=cosθ1, C234=cos(θ2+θ3+θ4) etc. används.
a1-a5 är avstånd mellan vridningsaxlar enl. fig 11.
z_0 är den första vridningspunktens z-koordinat. I och med att origo för wcs ligger på den andra axeln, blir z_0=-a1.

I matrisform ger translationerna och rotationerna:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 6 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C6 & -S6 & 0 \\ 0 & S6 & C6 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a5-6 & 0 & 0 & 1 \end{bmatrix}$$

$$
\begin{bmatrix} C5 & -S5 & 0 & 0 \\ S5 & C5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a4 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} C4 & 0 & S4 & 0 \\ 0 & 1 & 0 & 0 \\ -S4 & 0 & C4 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a3 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} C3 & 0 & S3 & 0 \\ 0 & 1 & 0 & 0 \\ -S3 & 0 & C3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & a2 & 1 \end{bmatrix}
$$

$$
\begin{bmatrix} C2 & 0 & S2 & 0 \\ 0 & 1 & 0 & 0 \\ -S2 & 0 & C2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & a1 & 1 \end{bmatrix}
\begin{bmatrix} C1 & -S1 & 0 & 0 \\ S1 & C1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & z_0 & 1 \end{bmatrix} =
$$

$$
= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C6 & -S6 & 0 \\ 0 & S6 & C6 & 0 \\ a5 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} C5 & -S5 & 0 & 0 \\ S5 & C5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a4 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} C4 & 0 & S4 & 0 \\ 0 & 1 & 0 & 0 \\ -S4 & 0 & C4 & 0 \\ a3 & 0 & 0 & 1 \end{bmatrix}
$$

$$
\begin{bmatrix} C3 & 0 & S3 & 0 \\ 0 & 1 & 0 & 0 \\ -S3 & 0 & C3 & 0 \\ 0 & 0 & a2 & 1 \end{bmatrix}
\begin{bmatrix} C2 & 0 & S2 & 0 \\ 0 & 1 & 0 & 0 \\ -S2 & 0 & C2 & 0 \\ 0 & 0 & a1 & 1 \end{bmatrix}
\begin{bmatrix} C1 & -S1 & 0 & 0 \\ S1 & C1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & z_0 & 1 \end{bmatrix} =
$$

$$
= \begin{bmatrix} C5 & -S5 & 0 & 0 \\ C6*S5 & C6*C5 & -S6 & 0 \\ S6*S5 & S6*C5 & C6 & 0 \\ a5*C5+a4 & -a5*S5 & 0 & 1 \end{bmatrix}
$$

$$
\begin{bmatrix}
C4*C3-S4*S3 & 0 & C4*S3+S4*C3 & 0 \\
0 & 1 & 0 & 0 \\
-S4*C3-C4*S3 & 0 & -S4*S3+C4*C3 & 0 \\
a3*C3 & 0 & a3*S3+a2 & 1
\end{bmatrix}
$$

$$
\begin{bmatrix}
C2*C1 & -C2*S1 & S2 & 0 \\
S1 & C1 & 0 & 0 \\
-S2*C1 & S2*S1 & C2 & 0 \\
0 & 0 & a1+z_0 & 1
\end{bmatrix}
=
\left(
\begin{array}{l}
\text{Trigonometriska} \\
\text{regler ger:} \\
C4*C3-S4*S3=C34 \\
C4*S3+S4*C3=S34
\end{array}
\right)
$$

$$
=
\begin{bmatrix}
C5 & -S5 & 0 & 0 \\
C6*S5 & C6*C5 & -S6 & 0 \\
S6*S5 & S6*C5 & C6 & 0 \\
a5*C5+a4 & -a5*S5 & 0 & 1
\end{bmatrix}
$$

$$
\left[
\begin{array}{ll}
C34*C2*C1-S34*S2*C1 & -C34*C2*S1+S34*S2*S1 \\
S1 & C1 \\
-S34*C2*C1-C34*S2*C1 & S34*C2*S1+C34*S2*S1 \\
a3*C3*C2*C1- & -a3*C3*C2*S1+ \\
(a3*S3+a2)*S2*C1 & (a3*S3+a2)*S2*S1
\end{array}
\right.
$$

$$
\left.
\begin{array}{ll}
C34*S2+S34*C2 & 0 \\
0 & 0 \\
-S34*S2+C34*C2 & 0 \\
a3*C3*S2+ & 1 \\
(a3*S3+a2)*C2+ & \\
a1+z_0 &
\end{array}
\right]
=
$$

$$
\left[
\begin{array}{ll}
C5*C234*C1-S5*S1 & -C5*C234*S1-S5*C1 \\
C6*S5*C234*C1+C6*C5*S1+S6*S234*C1 & -C6*S5*C234*S1+C6*C5*C1-S6*S234*S1 \\
S6*S5*C234*C1+S6*C5*S1-C6*S234*C1 & -S6*S5*C234*S1+S6*C5*C1+C6*S234*S1 \\
(a5*C5+a4)*C234*C1-a5*S5*S1+ & -(a5*C5+a4)*C234*S1-a5*S5*C1- \\
+a3*C23*C1-a2*S2*C1 & -a3*C23*S1+a2*S2*S1
\end{array}
\right.
$$

$$
\left.
\begin{array}{lll}
-C5*C234*S1-S5*C1 & C5*S234 & 0 \\
-C6*S5*C234*S1+C6*C5*C1-S6*S234*S1 & C6*S5*S234-S6*C234 & 0 \\
-S6*S5*C234*S1+S6*C5*C1+C6*S234*S1 & S6*S5*S234+C6*C234 & 0 \\
-(a5*C5+a4)*C234*S1-a5*S5*C1- & (a5*C5+a4)*S234+ & 1 \\
-a3*C23*S1+a2*S2*S1 & +a3*S23+a2*C2+a1+z_0 &
\end{array}
\right]
$$

Denna matris skall vara lika med matrisen:

$$\begin{bmatrix} n_x & n_y & n_z & 0 \\ o_x & o_y & o_z & 0 \\ a_x & a_y & a_z & 0 \\ p_x & p_y & p_z & 1 \end{bmatrix}$$

Detta ger villkoren:

p_x = a5*C5*C234*C1+a4*C234*C1+a3*C23*C1-a5*S5*S1-a2*S2*C1

p_y = -a5*C5*C234*S1-a4*C234*S1-a3*C23*S1-a5*S5*C1-a2*S2*S1

p_x = a5*C5*S234+a4*S234+a3*S23+a2*C2+a1+z_0

n_x = C5*C234*C1-S5*S1

n_y = -C5*C234*S1-S5*C1

n_z = C5*S234

Härav framgår att vinkeln φ6 ej har någon inverkan på p och n.
För att förenkla fortsatta räkningar sättes φ6=0°.
Då kan M skrivas som produkten av fem matriser:

$$M = A_5 A_4 A_3 A_2 A_1$$

$$A_1 = \begin{bmatrix} C1 & -S1 & 0 & 0 \\ S1 & C1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & z_0 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} C2 & 0 & S2 & 0 \\ 0 & 1 & 0 & 0 \\ -S2 & 0 & C2 & 0 \\ 0 & 0 & a1 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} C3 & 0 & S3 & 0 \\ 0 & 1 & 0 & 0 \\ -S3 & 0 & C3 & 0 \\ 0 & 0 & a2 & 1 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} C4 & 0 & S4 & 0 \\ 0 & 1 & 0 & 0 \\ -S4 & 0 & C4 & 0 \\ a3 & 0 & 0 & 1 \end{bmatrix}$$

$$A_5 = \begin{bmatrix} C5 & -S5 & 0 & 0 \\ S5 & C5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ C5*a5+a4 & -S5*a5 & 0 & 1 \end{bmatrix}$$

Inversen av en transformationsmatris

$$T = \begin{bmatrix} n_x & n_y & n_z & 0 \\ o_x & o_y & o_z & 0 \\ a_x & a_y & a_z & 0 \\ p_x & p_y & p_z & 1 \end{bmatrix}$$

$$T^{-1} = \begin{bmatrix} n_x & o_x & a_x & -p \cdot n \\ n_y & o_y & a_y & -p \cdot o \\ n_z & o_z & a_z & -p \cdot a \\ p_x & p_y & p_z & 1 \end{bmatrix}$$

$p \cdot o$ betecknar skalärprodukten mellan vektorn p och o etc.