NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk

Institutt for datateknikk og informasjonsvitenskap

EKSAMEN I EMNE
TDT4195 BILDETEKNIKK
ONSDAG 25. MAI 2005
KL. 09.00 – 13.00

Løsningsforslag - grafikk

Oppgave 3: Grunnleggende begreper

a) Gjør rede for sammenhengen mellom *grafiske primitiver* og *grafiske attributter*.

For å beskrive en scene må strukturen til objektene som befinner seg i scenen og deres plassering i forhold til scenen og hverandre angis. De funksjoner i et grafisk API som benyttes til å beskrive de ulike bildekomponentene blir kalt for grafiske primitiver. Et grafisk attributt er en parameter som beskriver hvordan et grafisk primitiv skal vises. Altså:

- **Grafiske primitiver:**
  * Definisjon: Bildekomponenter som beskriver geometrien til objekter.
  * Eksempler: punkt, linjer, sirkler, andre koniske objekter, splines, polygoner ...

- **Grafiske attributter:**
  * Definisjon: En parameter som beskriver hvordan et primitiv skal vises.
  * Eksempler: Farge, størrelse, tykkelse, stiplet eller ikke, fonttype, antialiasert ...

Sammenhengen er altså at et attributt beskriver hvordan et primitiv skal vises.

b) Angi stegene i *visualiseringssamlebåndet* (viewing pipeline) for tre dimensjoner slik de er nevnt i læreboka og beskriv kort hvilke operasjoner som utføres i hvert steg. Mellom hvilke to steg bør klipping foregå, og hvorfor bør klipping foregå akkurat der?

Modelltransformasjoner: Modellene beskrevet i modellkoordinater plasseres i verdenskoordinat-systemet. Alle modelleringstransformasjoner som skal utføres på modellene utføres i dette steget, for eksempel, all animasjon. Resultatet er en scene i verdenskoordinater.

Visningstransformasjoner: Visningstransformasjonene produserer visningskoordinater (kamerakoordinater), som beskriver modellenes plassering i forhold til kameraet. Transformasjonsmatrisen for overgang til visningskoordinatene finnes ved å translere og rotere kamerakoordinatsystemet inn i verdenskoordinatsystemet. Origo i kamerakoordinatsystemet transleres til origo i verdenskoordinatsystemet og aksene i kamerakoordinatsystemet roteres slik at de sammenfaller med aksene i verdenskoordinatsystemet.

Projeksjonstransformasjoner: Et vindu i projeksjonsplanet sammen med projeksjonssenter eller projeksjonsretning (avhengig av projeksjonsmetode) og avgrensninger foran og bak definerer projeksjonsvolumet og projeksjonstransformasjonen. Dersom visningsvolumet er skjevt, rettes dette opp i dette steget. Alt som befinner seg innenfor projeksjonsvolumet, skal vises i bildeplanet (såfremt ingen objekter i scene skygger for hverandre). Projeksjonstransformasjonene resulterer i projeksjonskoordinater, som bestemmer hvor i bildeplanet entiteter skal vises.

Normaliseringstransformasjoner: Transformerer betraktningsvolumet (som kan være et rettvinklet parallellepiped eller et frustrum) til en kanonisk kube. Normaliserte koordinater forenkler prosessen med å vise det syntetiserte bildet på ulikt utstyr. Resultatet av normaliseringstransformasjoner er normaliserte koordinater.

Viewporttransformasjoner: Viewporttransformasjoner omformer normaliserte koordinater til utstyrskoordinater. Utstyrskoordinatene blir generert på basis av det aktuelle utstyret som bildet skal vises på, og vil derfor kreve kunnskap om det aktuelle utstyret.

Klipping blir som oftest utført etter normalisering. Grunnen til dette er at man da kan klippe etter å ha slått sammen alle transformasjoner som ikke er avhengige av utstyrskoordinatene. Det er også forenklende å kunne klippe mot plan som er parallelle med koordinatplanene i koordinatsystemet.

De tredimensjonale (homogene) koordinatene transformeres ikke om til todimensjonale bildekoordinater før etter normaliseringstransformasjonen. Perspektivtransformasjoner er en inherent del av normaliseringstransformasjonen. **En besvarelse som kommenterer dette må anses som *meget* god.**

Oppgave 4: Diverse metoder

a) Funksjonen gluLookAt spesifiserer visningsparametere i OpenGL. Som parametere har funksjonen to punkter og en vektor. De to punktene er henholdsvis øyepunktet og referansepunktet. Vektoren er view-upvektoren. OpenGL benytter disse parametrene til å definere visningskoordinatsystemet (kamerakoordinatsystemet, viewing reference frame). Vis hvordan parametrene kan brukes til å sette opp et høyrehånds visningskoordinatsystem.

$\boldsymbol{N} = P_{COP} - P_{ref}$
$|\boldsymbol{N}| = sqrt(N_x^2 + N_y^2 + N_z^2)$
$\boldsymbol{n} = \boldsymbol{N}/|\boldsymbol{N}| = (n_x, n_y, n_z)$

$\boldsymbol{V_{up}}$ vil generelt ikke stå vinkelrett på $\boldsymbol{N}$. Dette må kommenteres for en perfekt besvarelse.

$\boldsymbol{V} = \boldsymbol{V}_{up} - (\boldsymbol{V_{up}} \cdot \boldsymbol{N}/|\boldsymbol{N}|^2)\boldsymbol{N}$
$\boldsymbol{v} = \boldsymbol{V}/|\boldsymbol{V}| = (v_x, v_y, v_z)$

Vektoren $\boldsymbol{u}$, som står vinkelrett både på $\boldsymbol{v}$ og $\boldsymbol{n}$, finnes ved et kryssprodukt av $\boldsymbol{v}$ og $\boldsymbol{n}$:

$\boldsymbol{u} = \boldsymbol{v} \times \boldsymbol{n} = (u_x, u_y, u_z)$
Ettersom $\boldsymbol{v}$ og $\boldsymbol{n}$ er normaliserte, blir også $\boldsymbol{u}$ det.

Alternativt kan metoden benyttet i boken brukes:

$\boldsymbol{u} = (\boldsymbol{V_{up}} \times \boldsymbol{n})/|\boldsymbol{V_{up}} \times \boldsymbol{n}|$

$\boldsymbol{v} = \boldsymbol{n} \times \boldsymbol{u}$

Enhetsvektorene $\boldsymbol{u}$, $\boldsymbol{v}$ og $\boldsymbol{n}$ danner et ortogonalt, høyrehånds koordinatsystem.

b) Skriv pseudokode for flatefyllingsalgoritmene *boundary-fill* og *flood-fill*. I hvilke tilfelle vil det lønne seg å bruke den ene framfor den andre?

Det bør komme frem at naboer besøkes, og det bør kommenteres om naboene er *four-connected* eller *eight-connected*. Under følger algoritmer som besøker naboer som er *four-connected*. Skannutvidet *boundary-fill* godtas også, men er ikke nødvendig. Dersom pseudokoden ikke er selvforklarende, bør en forklaring gis i tillegg.

```
//Boundary Fill
boundaryFill4(int, x, int y, int fillColor, int borderColor){
        int interiorColor;
        getPixel(x, y, interiorColor);
        if((interiorColor != borderColor) && (interiorColor != fillColor)){
                setPixel(x,y);
                boundaryFill4(x+1, y, fillColor, borderColor);
                boundaryFill4(x-1, y, fillColor, borderColor);
                boundaryFill4(x, y+1, fillColor, borderColor);
                boundaryFill4(x, y-1, fillColor, borderColor);
        }
}
```

```
//Flood fill
floodFill4(int x, int y, int fillColor, int interiorColor){
        int color;
        getPixel(x, y, color);
        if (color == interiorColor) {
                setPixel(x,y, fillColor);
                floodFill4(x+1, y, fillColor, interiorColor);
                floodFill4(x-1, y, fillColor, interiorColor);
                floodFill4(x, y+1, fillColor, interiorColor);
                floodFill4(x, y-1, fillColor, interiorColor);
        }
}
```

Boundary-fill brukes dersom man skal fylle en flate som har en grense definert av en farge, mens flood-fill fyller en flate som har en definert farge (skifter farge på flaten). Flood-fill kan dermed fylle flater med grenser definert av mer enn en farge. Flood-fill tåler ikke forurensning i flaten som skal fylles – alle piksler som definerer flaten må ha en farge og samme farge. Boundary-fill tåler derimot at flaten som skal fylles kan bestå av flere farger så lenge grensefargen ikke er en av disse.

c) Angi de nødvendige stegene i rett rekkefølge for å rotere et objekt 30 grader rundt linjen som går gjennom punktene (1,1,1) og (1,1,2). Still opp transformasjonsmatrisene med utregnede matriseelementer. Det kreves ikke at matrisene blir konkatenert.

Her gjelder det å oppdage at linjen som det skal roteres rundt er parallell med z-aksen. Stegene som må utføres er som følger:

1) Translasjon av et av punktene som definerer linjen det skal roteres rundt til origo.
2) Rotasjon med 30 grader rundt z-aksen.
3) Invers av 1)

1: Translasjon av linje til origo.
Tar utgangspunkt i punktet (1, 1, 1) og translerer det til origo.

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

2: Rotasjon rundt z-aksen
Roterer 30 grader rundt z-aksen.

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & 0 & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3: Invers av det første steget.
Translerer tilbake til utgangspunktet.

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

Transformasjonsmatrisen M for å rotere et objekt rundt linjen som beskrevet over er:

$$M = T^{-1}RT$$

Viktig at rekkefølgen blir korrekt!

Oppgave 5: Antialiasering

a) Forklar kort hva *antialiasering* er.

Grafiske primitiv som blir generert av rasteralgoritmer får taggete kanter. Dette kommer av at rasteralgoritmen diskretiserer kontinuerlig informasjon - det blir tatt for få prøver i forhold til frekvensinnholdet i informasjonen som skal beskrives (undersampling). Dette fenomenet kalles aliasering. Tiltak som i større eller mindre grad kompenserer for aliaseringseffekter, kalles antialiasering.

b) Forklar i korte trekk hvordan *supersampling*, *subpikselvektmasker* (subpixel weighting masks) og *filtrering* kan benyttes til å antialiasere tynne rette linjer. Hvilken av de nevnte metodene tror du vil gi best visuelt resultat og hvorfor?

Supersampling: Et bilde blir tegnet med større oppløsning enn hva som er nødvendig ved oppdeling av en piksel i subpiksler. Fargeintensiteten til en piksel i resultatbildet blir bestemt ved summering og midling av fargene til subpikslene. Alle subpiksler blir tillagt lik vekt.

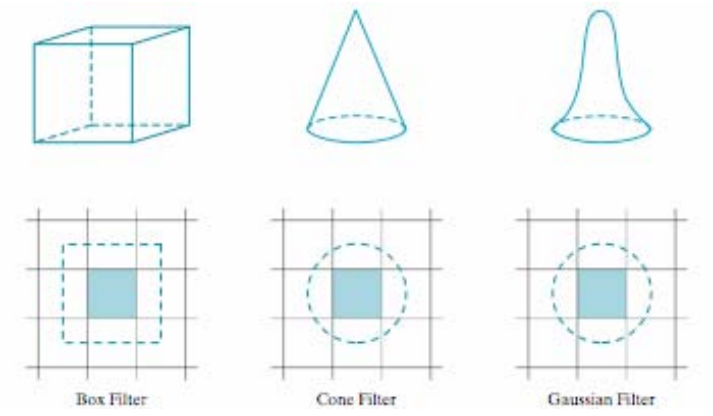
Vi kan dele opp hver piksel i et sett med subpiksler, og tegner linjen opp med, for eksempel, Bresenhams linjealgoritme. Vi teller deretter antall subpiksler som linjen treffer. Dersom vi deler hver piksel inn i ni subpiksler, vil hver piksel kunne ha tre forskjellige intensitetsnivåer, ettersom Bresenhams linjealgoritme maksimalt vil velge tre av subpikslene. En linje som skjærer en piksel i tre av subpikslene vil ha sterkest intensitet (en intensitet på tre), mens en linje som skjærer en piksel i ett av subpikslene vil ha lavest intensitet (en intensitet på en).

Subpixel weighting masks: Supersampling hvor subpikslene blir vektet ulikt. Som regel blir subpiksler som ligger nærmere pikselsenteret blir mer vektlagt. En mulig subpikselvektmaske for ni subpiksler er gjengitt under.:

1	2	1
2	4	2
1	2	1

Masken tillegger mest vekt til subpikslen som ligger i sentrum. Sentrumssubpikselen er vektet dobbelt av de subpikslene som ligger rett over, rett under, til høyre side og til venstre side, mens det er vektet firedobbelt av subpikslene som ligger i hjørnene.

Filtrering: Filtreringsteknikker baserer seg på å la en kontinuerlig vektfunksjon dekke pikselen. Boken omtaler filterteknikker på subpiksler, men det er også vanlig at filteret dekker en piksel og i tillegg brer seg inn over nabopikslene. Figuren under illustrerer ulike vektfunksjoner.

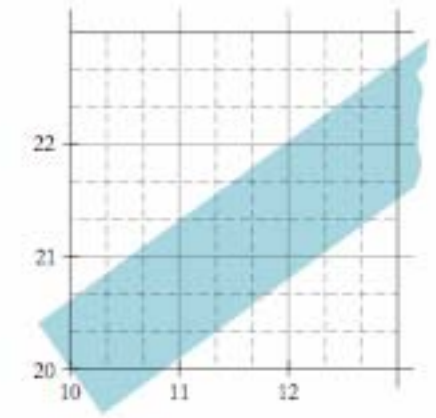


Volumet under hvert filter blir vektet til 1.0 og høyden under filteret angir den relative vekten for posisjonen i pikselen. Man finner den vektede pikselintensiteten ved å integrere langs linjens vei gjennom pikselen over pikseloverflaten. Integrasjonen foregår ved oppslag i en forhåndsberegnet tabell der oppslagsparameteren er linjens avstand fra pikselsenteret.

Filterteknikker gir best resultat ettersom vekten bestemmes med større oppløsning enn tilfellet er med de to andre teknikkene.

c) Forklar i korte trekk hvordan teknikken *arealsampling* (area sampling) benyttes ved antialiasering av linjer med tykkelse.

Arealsampling av linjer med tykkelse foregår ved at man setter intensiteten til en piksel basert på hvor mye av pikselen som dekkes av linjen. Linjen kan bli behandlet som et rektangel. Deler av linjen som ligger mellom to horisontale (eller vertikale) piksellinjer som ligger ved siden av hverandre vil ha form som en trapes. Vi kan regne ut hvor store deler av hver piksel som blir overlappet av trapesen.



Figuren over illustrerer en linje med tykkelse, som skjærer over et rutenett av piksler. Pikselen (10, 20) er omlag 90 prosent dekket og vil dermed få en intensitet på 9/10, mens pikselen (10, 21) er omlag 10 % dekket og vil dermed få en intensitet på 1/10.

En enkel måte å approksimere arealsampling er å telle antall subpiksler som er dekket av linjen. En subpiksel telles om for eksempel nedre venstre hjørne i subpikselen er dekket av linjen. Vi ser av figuren over at for pikselen (10,20) er etter denne regelen 8 av 9 subpiksler er dekket av linjen og intensiteten settes dermed til 8/9.

Oppgave 6: Linjetegning

Gjør rede for *midtpunktsalgoritmen* for linjetegning.

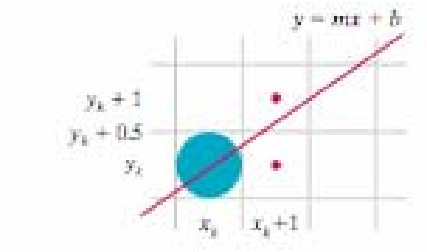
Når begrepet linje brukes uten nærmere kvalifisering, menes vanligvis en rett linje. Her kunne det likevel ha vært presisert at det gjaldt rette linjer. Studenter som spurte under eksamen ble gitt beskjed om at det gjaldt kurver generelt. De som har besvart spørsmålet og illustrerer prinsippene ved hjelp av sirkler eller ellipser, vil få full score. Under sensuren ble det tatt hensyn til eventuelle vanskeligheter som måtte ha oppstått på grunn av dette.

Midtpunktalgoritmen for linjetegning er en algoritme for rastrering av linjer. Linjer representerte som kontinuerlige i datamaskinen skal tegnes ved hjelp av diskrete pikselverdier til en skjerm eller liknende. Dermed må det avgjøres hvilke piksler som skal tegnes for bestemte linjer. Denne avgjørelsen tas ved hjelp av en bestemmelsesparameter. Bestemmelsesparameteren baserer seg på om linjen skjærer over eller under midtpunktet mellom pikselsentrene til de to pikslene det skal velges mellom. Midtpunktalgoritmen kan tilpasses alle kjeglesnitt. Boken beskriver også midtpunktsalgoritmer for sirkler og ellipser.

Som utgangspunkt tar vi at punktet (x_k, y_k) er tegnet og at linjens stigningstall har en verdi som ligger mellom 0 og 1. Linjer med stigningstall lik 0 eller ∞ er linjer parallelle med hovedaksene.

Det er mulig å utvide algoritmen til å gjelde for alle linjer uavhengig av at stigningstallet ligger mellom 0 og 1 ved å ta i betraktning symmetri mellom de ulike kvadrantene og oktantene. For linjer med stigningstall m > 1 er det i hovedsak nok å bytte om x og y i likningene, og dersom stigningstallet er negativt, er det bare å benytte negative y-inkrementer. Dersom startpunktet til linjen ligger til høyre for endepunktet, kan man bytte om på start- og endepunkt.

Figuren under illustrerer situasjonen hvor pikselen (x_k, y_k) er tegnet. Dette er markert med en stor blå sirkel. Pikselposisjonen (x_k, y_k) betegner pikselsenteret, som er markert med små røde sirkler. Linjetegningsalgoritmen skal velge den neste pikselen som skal tegnes, og valget står mellom de to pikslene (x_{k+1}, y_k) og (x_{k+1}, y_{k+1}).



Midtpunktsalgoritmen er en heltallsalgoritme som tar utgangspunkt i likningen for vedkommende kjeglesnitt på implisitt form. Likningen for en rett linje (spesialtilfelle av kjeglesnitt) på implisitt form:

$$f(x,y) = ax + by + c = 0$$

Koeffisientene ***a***, ***b*** og ***c*** er skalerbare, Ved passende skalering gjøres de til heltall. Endepunktskoordinatene (x_1, y_1) og (x_2, y_2) forutsettes avrunder til heltall.

Vi definerer:

$$dx \equiv x_2 - x_1$$
$$dy \equiv y_2 - y_1$$

Stigningsforholdet for linjen er:

$$m \equiv \frac{dy}{dx}$$

Vi forutsetter:

$$0 \le m \le 1 \text{ og } x_2 - x_1 > 0$$

Da er dx og dy positive heltall.

Linjelikningen på eksplisitt (vanlig) form er:

$$y = \frac{dy}{dx}x + h$$

Omforming til implisitt form gir:

$$f(x, y) = 2x \cdot dy - 2y \cdot dx + 2h \cdot dx = 0$$

Uttrykk for koeffisientene $\boldsymbol{a}$, $\boldsymbol{b}$ og $\boldsymbol{c}$ blir da:

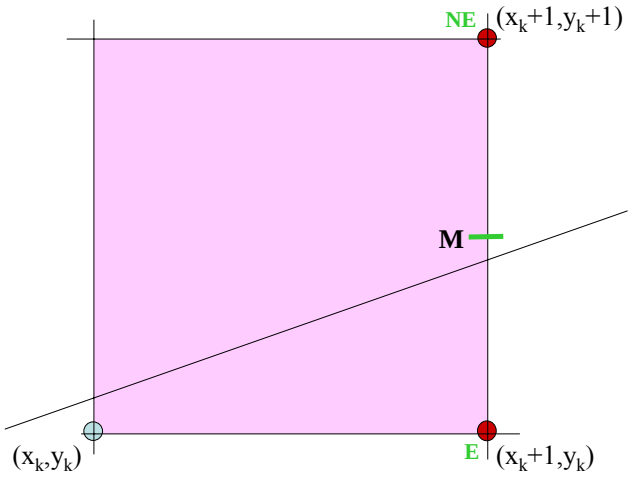
$$a = 2dy$$
$$b = -2dx$$
$$c = 2h \cdot dx$$

Pikselen (x_k, y_k) er den siste pikselen som ble "slått på". Det er to kandidatpiksler for den neste å "slå på":

$$(x_k + 1, y_k)$$

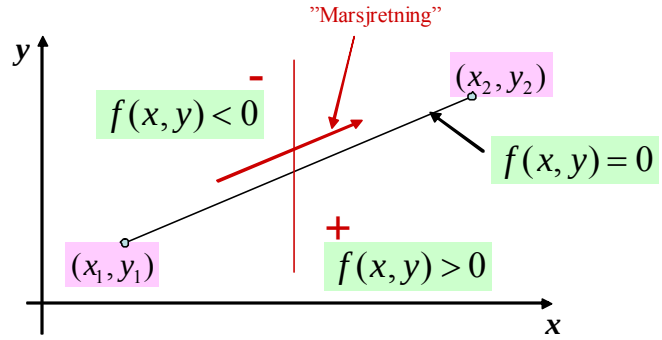
og

$$(x_k + 1, y_k + 1)$$



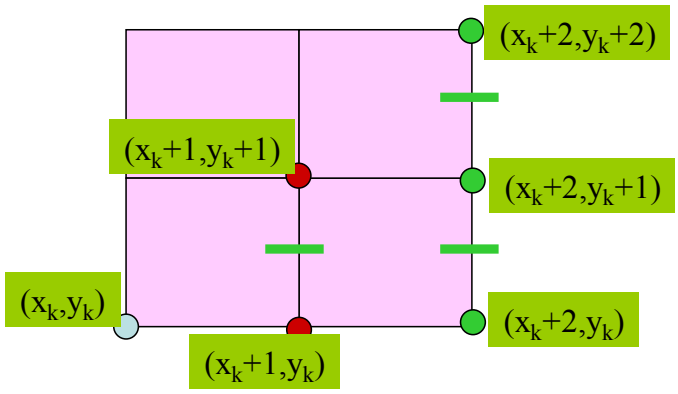
Vi definerer en desisjonsvariabel (bestemmelsesparameter):

$$d_k \equiv f(M) = f(x_k + 1, y_k + \frac{1}{2}) = a(x_k + 1) + b(y_k + \frac{1}{2}) + c$$



Dersom $d_k \geq 0$, velges NE-pikselen, dersom $d_k < 0$, velges E-pikselen.

Videre pikselvalg er avhengig av hvilket valg som ble gjort på "nettlinjen" $x_k + 1$.



Dersom $(x_k + 1, y_k)$ ble valgt, er de nye kandidatene $(x_k + 2, y_k)$ og $(x_k + 2, y_k + 1)$.

Dersom $(x_k + 1, y_k + 1)$ ble valgt, er de nye kandidatene $(x_k + 2, y_k + 1)$ og $(x_k + 2, y_k + 2)$.

E ble valgt på nettlinjen $x_k + 1$. Ny desisjonsvariabel:

$$\underline{\underline{d_{k+1}}} = f(x_k + 2, y_k + \frac{1}{2}) = a(x_k + 2) + b(y_k + \frac{1}{2}) + c = a(x_k + 1) + b(y_k + \frac{1}{2}) + c + a = d_k + a =$$

$$= \underline{\underline{d_k + 2dy}}$$

NE ble valgt på nettlinjen $x_k + 1$. Ny desisjonsvariabel:

$$\underline{\underline{d_{k+1}}} = f(x_k + 2, y_k + \frac{3}{2}) = a(x_k + 2) + b(y_k + \frac{3}{2}) + c = a(x_k + 1) + b(y_k + \frac{1}{2}) + c + a + b = d_k + a + b =$$
$$= \underline{\underline{d_k + 2dy - 2dx}}$$

Startverdi for desisjonsvariabelen ***d***:

$$\underline{\underline{d_1}} = f(x_1 + 1, y_1 + \frac{1}{2}) = a(x_1 + 1) + b(y_1 + \frac{1}{2}) + c = ax_1 + by_1 + c + a + \frac{b}{2} = f(x_1, y_1) + a + \frac{b}{2} = 0 + a + \frac{b}{2} =$$
$$= \underline{\underline{2dy - dx}}$$

Alle operasjoner i midtpunktsalgoritmen er heltallsoperasjoner.