Assignment 1: Scankonvertering af linjer

Marie Elkjær Rødsgaard - dck495
Department of Computer Science

November 29, 2022

Contents

1 Introduktion

Denne opgave er den første opgave i Grafik og omhandler midtpunktsalgoritmen til skankonvertering af rette linjer. I denne opgave skal man forstå, forklare og implementere midtpunktsalgoritmen. Denne rapport vil gå i dybden med hvordan problemet af de forskellige punkter kan blive løst.

2 Overblik

For at kunne snakke om midtpunktsalgoritmen skal vi først forstå hvad skankonvertering af rette linjer er. Skankonvertering af rette linjer er at få repræsenteret en ret linje så præcist som muligt ved brug af pixels.

3 Midpunktsalgoritmen

3.1 Definition på en linje

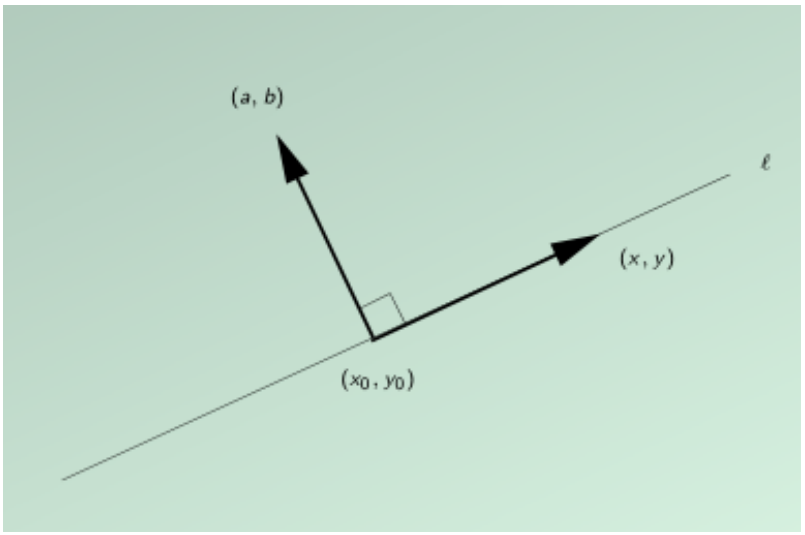

Figure 1: Linje l og dens normal vektor (a,b)

På figuren over ses en linje l og dens normalvektor (a,b). l kan defineres som

$$l : \{(x, y)|(a, b) \cdot (x - x_0, y - y_0) = 0\}$$

Nu kan vi regne den homogene ligning af den rette linje

$$(a, b) \cdot (x - x_0, y - y_0) = 0$$

$$ax + by - (ax_0 + by_0) = 0$$

Den sidste del kan vi nu kalde c

$$ax + by + c = 0$$

Nu har vi fundet linjens homogene ligning.

$$l = \{ax + by + c = 0\}$$

Nu kan vi så kigge på distancen til den rette linje. Dette kalder vi $d(x, y) = \frac{(a,b)}{\sqrt{a^2+b^2}} \cdot (x - x_0, y - y_0)$ Vi kan nu gange dette udtryk ud. I tælleren får vi det samme som vores l.

$$d(x, y) = \frac{ax + by + c}{\sqrt{a^2 + b^2}}$$

Tallet som d giver os er ikke et heltal. Vi kan dog bruge d på den måde at vi kun behøver at vide hvilket fortegn det returnerer.

$$d(x, y) = \begin{cases} > 0, \text{ over the line} \\ = 0, \text{ on the line} \\ < 0, \text{ under the line} \end{cases}$$

Da kvardratroden altid giver et positivt tal kan vi lade vær med at kigge på den. Vi får derfor en funktion

$$F(x, y) = ax + by + c$$

3.2 Teori

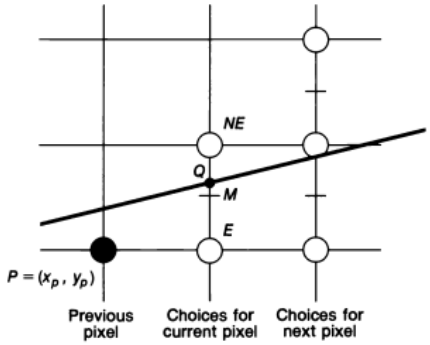

Fig. 3.7 The pixel grid for the midpoint line algorithm, showing the midpoint *M*, and the *E* and *NE* pixels to choose between.

Figure 2: Figur fra Foley kapitel 3.2.2

Midtpunktsalgoritmen er kort sagt en algoritme der vælger mellem to punkter `NE` eller `E` ud fra hvilket fortegn en værdi `d` er. Efter et punkt er valgt opdateres vores værdier forskelligt ud fra om vi valgte `E` eller `NE`.

På figuren ovenover ses de forskellige punkter der skal forstås før vi kan forstå hvordan vi vælger hvilket punkt, der er det rigtige og hvordan vi opdatere vores værdier.

Vi starter med at kigge på den sorte pixel, dette er det valgte punkt $p = (x_p, y_p)$. Det hvide punkt, som er plus én på x aksen, er punktet `E` og punktet plus én på x og y aksen er `NE`. Det er en af de to punkter som algoritmen skal vælge imellem. `Q` er det punkt hvor den linje som skal scankonverteres rammer $x = xp + 1$. I midten af vores `E` og `NE` er punktet `M`. Hvis `M` er under punktet `Q` vælges `NE` og omvendt vælges `E`.

For at regne dette ud starter vi med at repræsentere linjen med en implicit funktion med koefficienterne a, b og c. Denne funktion har vi fundet i afsnittet over.

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

Hvis vi så siger at $dy = y_1 - y_0$ og $dx = x_1 - x_0$ så kan vi skrive *slobe-intercept*

som:

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

Dette kan vi skrive om til

$$F(x, y) = dy \cdot x - dx \cdot y + B \cdot dx = 0$$

Hvor $a = dy$, $b = -dx$ og $c = B \cdot dx$ i vores implicit form.

For at bruge midtpunktskriteriet kan vi nøjes med at regne

$$F(M) = F(x_p + 1, y_p + \frac{1}{2})$$

Derefter kan vi så aflæse hvilket fortegn vi får. Dette definere vi nu som vores beslutningsvariabel

$$d = F(x_p + 1, y_p + \frac{1}{2})$$

Dette kan vi skrive som $d = a(x_p + 1) + b(y_p + \frac{1}{2}) + c$. Hvis $d > 0$ så vælger vi pixel `NE` og $d < 0$ vælger vi `E`. Og hvis $d = 0$ vælger vi bare en af dem.

Det næste trin er nu at finde ud af hvad der sker med `M` og `d` værdierne. Det er forskelligt hvad der sker alt efter om det er `E` eller `NE` der bliver valgt. Hvis `E` bliver valgt skal `M` inkrementeres med 1 i x retningen. `d` bliver nu også ændret da den er afhængig af `M`.

$$d_{new} = F(x_p + 2, y_p + \frac{1}{2}) = a(x_p + 2) + b(y_p + \frac{1}{2}) + c$$

Vores gamle d var

$$d_{old} = F(x_p + 1, y_p + \frac{1}{2}) = a(x_p + 1) + b(y_p + \frac{1}{2}) + c$$

For at finde forskellen på de to skriver vi $d_{new} = d_{old} + a$. Nu kan vi se at den inkrementerede værdi efter at have valgt `E` som vi kalder $\Delta_E = a = dy$. Dette betyder at vi ikke behøver at beregne $F(M)$ igen men bare kan tilføje Δ_E.

Hvis `NE` bliver valgt har vi at `M` bliver flyttet 1 mod x og 1 op på y. d bliver nu

$$d_{new} = F(x_p + 2, y_p + \frac{3}{2}) = a(x_p + 2) + b(y_p + \frac{3}{2}) + c$$

Forskellen her er så

$$d_{new} = d_{old} + a + b$$

Så vi kalder dette $\Delta_{NE} = a + b = dy - dx$.

Nu har vi været igennem alle trinene i algoritmen. [Foley, 1990] Denne algoritme er også hurtig fordi at den nemt kan laves om så den regner helt i `d`. Vi kan gange 2 ind i `d` da det kun er fortegnet der er vigtigt.

3.3 Implementation

Implementationen af midtpunktsalgoritmen foregår rimelig meget som forklaret i teoriafsnittet. Dog er der et par få ændringer som at få `d` til et heltal, da kun fortegnet er vigtigt, og at sørge for at algoritmen virker med alle slags hældninger.

Implementation af algoritmen benytter sig af et if statement som tjekker om punktet er `x_dominant`. Dette vil sige at variationen er større i x-aksen end den var i y-aksen. Og omvendt når den er `y_dominant`. Inden det starter vi med at initialisere nogle værdier her: `x_step` og `y_step` som gør at vi kan steppe til højre og venstre og op og ned. Vi har så en bool `x_dominant = (abs_2dx > abs_2dy)` som betyder at variationen i x er større end i y.

I `x_dominant` statement har vi så defineret en beslutnings variable. Vi definere også en `bool left_right = (x_step > 0);`. Dette gør vi så vi kan sikre at det er den samme linje der bliver tegnet uanset om vi går fra venstre mod højre eller omvendt. Vi laver samme `bool` i `else` statement, her har vi bare *y_step* i stedet for. Punkter bliver tegnet ved brug af en glm vector. *else* statementet fungere på samme måde som `if` statementent, nu er det bare y vi tager højde for.

`linerasterizer` fungerer principielt på samme måde. Kodeopsætningen er dog lidt anerledes. Resultatet for de to funktioner er ens.

Implementationerne tager højde for alle hældninger og er ens lige meget hvilken vej den er tegnet fra. Dette gør at den virker på linjer med hældning større end 1 og mindre end 0.

3.4 Test

For at teste om implementationen tegner linjer med alle hældninger tjekker vi om den tegner linjen rigtigt i alle oktanter. Som ses på figurene nedenunder kan implementationen tegne i alle oktanter. Dette virker med begge implementationer. Der er også tjekket at den samme linje bliver tegnet af de samme pixles uanset hvilken vej man går.

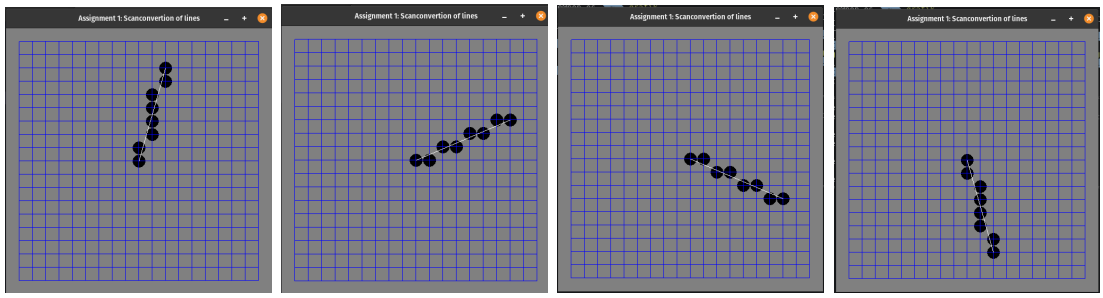

Figure 3: Linjer i de første fire oktanter

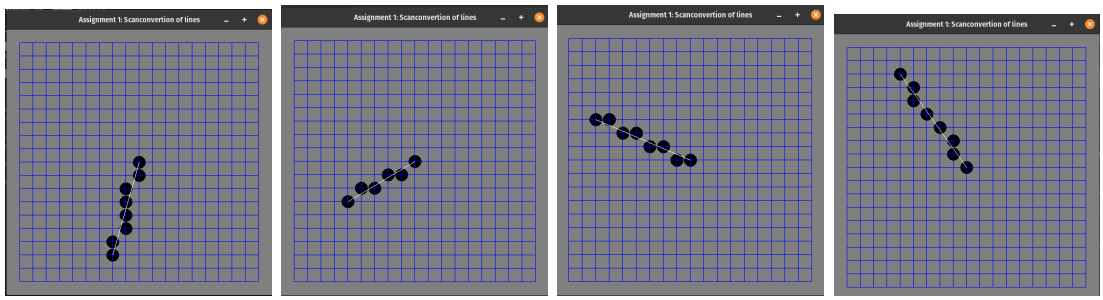

Figure 4: Linjer i de fire sidste oktanter

4 Konklusion

Vi har nu lavet en implementation af midtpunktsalgoritmen efter først at have gennemgået hvordan algoritmen fungere og derefter skrevet det til kode. Implementationen er blevet testet for at se om den kan håndtere forskellige hældninger og om den tegner samme pixels uanset om man går fra højre mod venstre eller omvendt.

References

[Foley, 1990] Foley, J. (1990). *Computer graphics: principles and practice.* The Systems Programming Series. Addison-Wesley.