Assignment 2: Scankonvertering af Trekanter

Marie Elkjær Rødsgaard - dck495
Department of Computer Science

November 29, 2022

Contents

1 Introduktion **2**

2 Teori **2**

3 Implementation **4**

4 Test **5**

5 Konklusion **6**

1 Introduktion

Denne opgave omhandler skankonvertering af trekanter. I denne opgave skal man forstå de to algoritmer og hvordan de sammen skankonvertere trekanter. Dette problem vil blive uddybet i rapporten.

2 Teori

Når en trekant skal skankonvertere bliver det gjort fra nederste venstre hjørne op mod højre hjørne. For ikke at få problemer med overlap, hvis eksempelvis trekanter skal kombineres til et polygon, er der nogle regler for hvilke pixels der skal med og hvilke pixels der ikke skal med. Pixels på en bundlinje af trekanten skal altid med, og det samme gælder for pixels på venstre kant samt pixels inde i selve trekanten. Pixels der ikke skal med er pixels på en top kant samt højre kant. Dette betyder også at to ens trekanter kan blive konverteret forskelligt alt fra hvordan den ligger.

En trekants kant er defineret af dens startpunkt (x_i, y_i) og slutpunkt (x_{i+k}, y_{i+k}). En kant kan også ses som et linjestykke, som kan vises på denne formel.

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

Vi skankoverterer kanter langs y-aksen så formlen kan skrives om til

$$x = \frac{dx}{dy}(y - \beta) = \frac{dx}{dy}y - \frac{dx}{dy}\beta$$

Dette betyder at den ændring vi har i x-positionen når vi flytter os fra skanlinjen er lig med y_i hen til y_{i+1}

$$x_{i+1} - x_i = \frac{dx}{dy}(y_{i+1} - y_i$$

Dette kan vi så skrive til

$$x_{i+1} = x_i\frac{dx}{dy}(y_{i+1} - y_i = x_i + \frac{dx}{dy}$$

Hvis vi kigger på beregningen for $x_{i+3} = x_i + \sum_{j=1}^{3} \frac{dx}{dy}$, kan vi kalde x_i for vores integer del. Resten kalder vi brøkdelen eller frac. Når vi skal skankonvertere på en kant er der to slags kanter at kigge på: kanter med negative

hældninger og kanter med positive hældninger.

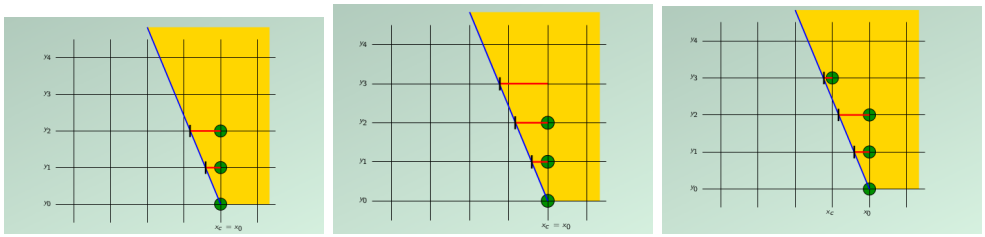

Figure 1: Negativ hældning kant eksempler

Algoritmer for kanter med negativ hældning starter med at sætte $x_c = x_0$ hvor vi sætter en pixel. Vores frac sætter vi lig med 0. Nu går vi videre og kigger på hvor den næste pixel skal være. Vi starter med først at ligge $frac = frac + |\frac{dx}{dy}|$, hvis vores $frac < 1$ så bliver vores $x_1 = x_c$ som ses på første figur i figure 1. Hvis vores $frac \geq 1$ så ville vores $x_1 = x_c = x_c - 1$ og vores frac ville blive $frac = frac - 1$ som ses på den anden og tredje figur i figure 1. Vores beslutning afhænger derfor af hvad vores frac er og det er frac der finder ud af hvad x_c skal være.

Vores frac kan også skrives op som $frac = \frac{\sum |dx|}{|dy|} \Leftrightarrow \sum |dx| \leq |dy|$. Dette kan vi også skrive som $frac = \frac{\text{Accumulator}}{\text{Denominator}} = \frac{\sum |dx|}{|dy|}$ så ville $frac = 0$ være Accumulato $= 0$. $frac \geq 1$ være det samme som Accumulator $\geq$ Denominator og $frac = frac - 1$ ville være Accumulator = Accumulator − Denominator. Denne måde at sætte det op hjælper os til at kombinere med positive hældninger senere.

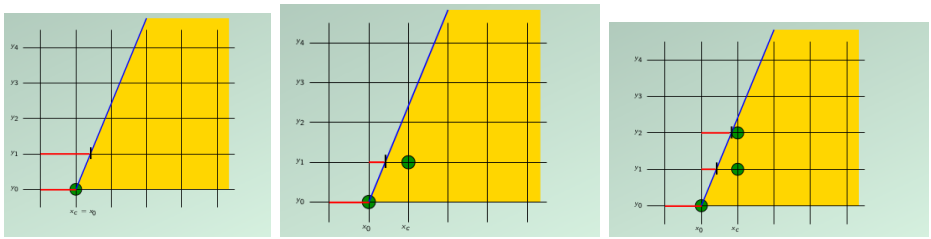

Figure 2: Positive hældning kant eksempler

Algoritme for kanter med positiv hældning starter med at sætte frac = 1 og $x_c = x_0$ som før. Vi sætter nu $frac = frac + |\frac{dx}{dy}|$ hvis $frac > 1$ så sætter vi $x_1 = x_c + 1$ og $frac = frac - 1$ som ses på de to første figurer i figure 2. Hvis $frac \leq 1$ så er $x_1 = x_c$ som ses på den sidste figur i figur 2. Vi sætter $frac = 1$ her for at få vores x_c til at inkrementere første gang.

Igen ville $\frac{\text{Accumulator}}{\text{Denominator}} = \frac{\sum |dx|}{|dy|}$ så ville $frac = 1$ være Accumulato $= |dy|$. $frac > 1$ være det samme som Accumulator $>$ Denominator og $frac = frac - 1$ ville være Accumulator = Accumulator − Denominator.

For at kombinere de to algoritmer kan vi sætte vores Accumulator = 1 og vælge kun at sige $>$ i stedet for $\geq$. Så ville algoritmen virke for begge slags linjer.

3 Implementation

Implementationen benytter de to algoritmer kombineret. Før vi kan bruge algoritmen skal vi dog have styr på at trekanter kan se ud på forskellige måder - der er fire cases. Vi kan have en trekant med en venstre kant og en højre kant: denne trekant vil så have enten en bundkant eller en topkant. Når vi har en af de to cases her behøver vi ikke tænke over top- og bundkanten. Dette er fordi at de pixels til de kanter vil blive tegnet med de andre kanter. De to andre cases er så at vi kan have en trekant med to højre kanter eller to venstre kanter. Hvis vi har en trekant med to venstre kanter skal vi bruge algoritmen på begge disse kanter. I min edge.cpp tjekker vi om der er top eller bundkanter og har en bool som returnere sandt eller falsk, og vi initialisere vores kanter og kan tælle op langs med vores skanlinje.

Dette bliver så brugt i min triangle.cpp som skankonvereterer til trekanter. Det sker ved først at initialisere venstre kant(er) og derefter højre kant(er). Derefter får vi edgekoordinaterne for venstre og højre kanterne. Så tjekkes der først om de venstre punkter er mindre end de højre punkter og derved om det er inden i trekanten eller at det ikke er det samme punkt. Dette resulterer i at vi får alle de pixels tegnet mellem punkterne for vores venstre og højre kant.

4 Test

For at teste at trekanterne bliver tegnet rigtigt har vi tjekket forskellige tilfælde. Og efter testerne kan vi konkludere at den tegner trekanter som ønsket og vi lever op til vores regler. Den tegner ikke højre kanter, eller top kanten. Der er dog også nogle trekanter den ikke kan tegne som har meget lille areal. Et af billederne viser en trekant hvor blot én pixel bliver tegnet, dette er ikke en god repræsentation af netop denne trekant, men det kan ikke gøre bedre for den slags.

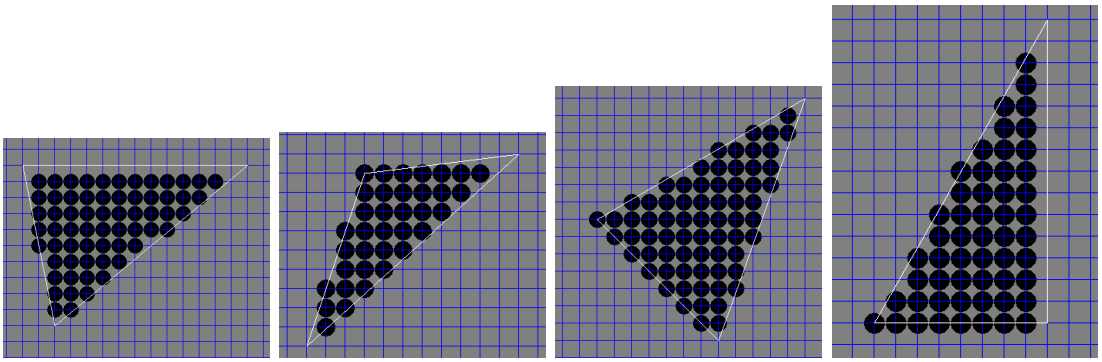

Figure 3: Forskellige tilfælde af trekanter.

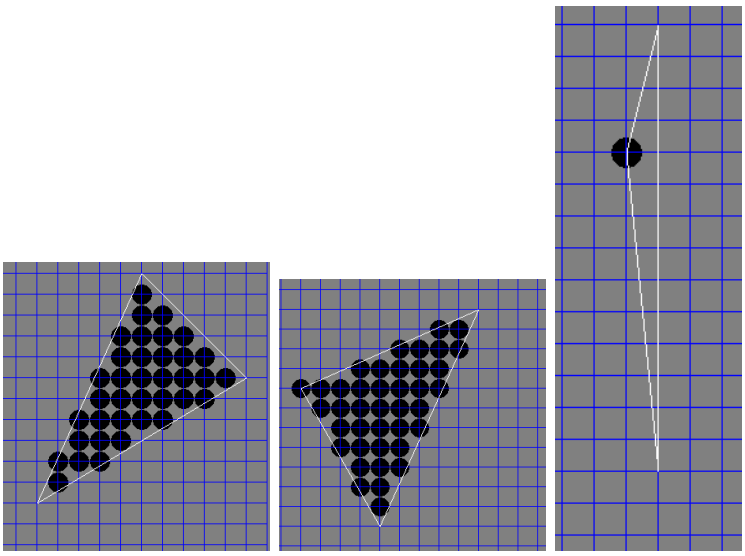

Figure 4: Flere forskellige tilfælde af trekanter.

5 Konklusion

Vi har nu lavet en implementation af skankonvertering af trekanter efter først at gennemgå hvordan man skal håndtere forskellige kanter og hvordan det kan kombineres og så bruges til at lave trekanter. Implementationen er testet for at se om den kan håndtere forskellige slags trekanter og opfylder vores ønskede krav.