Fundations of Computer Graphics
Ugeopgave 3

Mads Ohm Larsen

22. februar 2010

1 Introduktion

I denne uge vil vi gerne tegne trekanter. Trekanterne skal helst være fyldte trekanter. Grunden til at vi er så vilde med trekanter, er at alle polygoner kan brydes ned til trekanter, det vil sige de kan blive trianguleret. Det betyder at vi kan tegne alle former og figurer ved hjælp af trekanter, og derfor behøver vi ikke kunne tegne andet, da, hvis vi for eksempel, gerne vil have en firkant, bare kan tegne to trekanter.

Når vi tegner trekanter vil vi følge en vis konvention, nemlig at højre- og topkanter ikke bliver tegnet. Dette gøres simpelthen fordi at så står pixelsne ikke og flimre mellem to forskellige farver, hvis vi tegner to trekanter ved siden af hinanden i forskellige farver. Det betyder selvfølgelig at alle trekanter bliver en lille smule mindre end de burde være, men de bliver tilgengæld også det pænere at kigge på.

2 TreKANTER

Hvad er det præcis en trekant er? En trekant er en figur, med tre kanter. Skal vi tegne den, kan vi altså tegne de tre kanter og alle pixels inden i denne konvekse figur. Vi skal dog have i bagtankerne at vi ikke skal tegne højre- og topkanter.

Den nemmeste måde at tegne trekanter på, er således ved at tegne såkaldte skanlinier. Med skanlinier menes en vandret linie, som skær både venstre- og højrekanterne. Dette kan selvfølgelig gøres *dumt* ved faktisk at beregne skæringerne i flydende tal og derefter afrundet resultatet.

Vi vil, ligesom vores line rasterizer fra sidste uge, lave en inkrementiel algoritme til at tegne disse trekanter. Det betyder imidlertid at vi først må vide hvordan en kant ser ud, vi laver altså en `edge_rasterizer` først, og med denne kan vi lave en `triangle_rasterizer`.

3 Kanter

3.1 Hvad er en kant?

En meget kort beskrivelse af en kant er en linie, fra (x_{start}, y_{start}) til (x_{stop}, y_{stop}).

Det er sådan man normalt kender kanter i en f.eks. en trekant, men vi vil gerne lidt mere. Vi vil gerne kunne sige at flere linie stykker danner en kant. Vi

tænker derfor ikke på en kant, som en faktisk kant i en f.eks. en trekant, men derimod som et sæt af liniestykker, som går fra et punkt til et andet, eventuelt gennem et eller flere midterpunkter.

Hvor om alting er, så kigger vi kun på de kanter som består af et liniestykke, da vi kan kæde disse sammen på en smart måde, som jeg vil komme ind på senere. En kant kan beskrives som en linie med følgende ligning:

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

som vi allerede kender så godt fra vores line rasterizer. Dette er selvfølgelig blot den normale ligning for en linie, hvor dy er ændring i y og dx er ændring i x. Da vi skankonvertere langs y-aksen, skriver vi denne ligning om til

$$\begin{aligned} x &= \frac{dx}{dy}(y - \beta) \\ &= \frac{dx}{dy}y - \frac{dx}{dy}\beta \end{aligned}$$

En ændring på x-aksen, vil således være

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

Denne kan igen skrives om til

$$\begin{aligned} x_{i+1} &= x_i + \frac{dx}{dy}(y_{i+1} - y_i) \\ &= x_i + \frac{dx}{dy}\Delta y \end{aligned}$$

Da vi gerne vil regne pixels, kan vi antage at $\Delta y = 1$, og vi får da at i en given skanlinie (y_{i+p}) er den tilhørende x-værdi:

$$\begin{aligned} x_{i+p} &= x_i + \sum_{j=1}^{p} \frac{dx}{dy} \\ &= x_i + \frac{\sum_{j=1}^{p} dx}{dy} \end{aligned} \tag{1}$$

Vi har således delt vores værdi op i en heltals og en brøk del. Hvis brøken giver mere end 1 skal vi tælle vores heltal det antal ned (vi starter med at lave kanter med negativ hældning), ellers skal vi ikke bruge brøken til noget. Brøken er kun større end 1 hvis tælleren er større end nævneren, så derfor kan vi lave en sammenligning i stedet for en division og en sammenligning.

Når vi laver kanter med positiv hælding, skal vi sørger for at komme på den rigtige side af stregen, og vi skal derfor starte én gang længere inde. Dette vil jeg ikke komme mere ind på i denne korte gennemgang af hvordan vi tegner kanter.

3.2 Algoritmen

Når vi implemtere algoritmen, kan vi, i stedet for at regne hele summen fra (**??**), blot holde styr på den pågældende iteration.

Har vi en funktion `set_dot(x, y)`, som tegner en pixel i (x, y), kan vi gøre følgende for at tegne alle pixels på en linie

```
int dx          = x_stop - x_start;
int x_step      = ( dx < 0 ) ? -1 : 1;
int numerator   = |dx|;
int denominator = y_stop - y_start;
int accumulator = ( dx > 0 ) ? denominator : 1;
int x           = x_start;

for( int y = y_start; y < y_stop; ++y ) {
    set_dot( x, y );
    accumulator += numerator;
    while( accumulator > denominator ) {
        x           += x_step;
        accumulator -= denominator;
    }
}
```

4 Trekanterne

Der findes fire generiske trekanter. To har horisontale kanter, top henholdsvis bund kanten. De to andre har en lige kant fra det nederste venstre hjørne til det øverste venstre hjørne. De andre to kanter i disse to kan vi sætte sammen til én kant, som jeg kort var inde på tidligere. Vi kan sætte dem sammen, da vi er sikre på at de mødes i et hjørne (ellers ville det jo ikke være en kant i en trekant) og vi er sikre på at den øverste af disse ender i det øverste venstre hjørne. Det vil heller ikke påvirke skanlinierne af samme grund, og det er derfor en smart ting at gøre.

Når vi har begge kanter på plads, er det ingen problem at tegne en trekant. Her skal vi jo blot tegne alle punkterne i en skanlinie, bortset fra dem der ligger på en top- eller højrekant.

Vores algoritme er således rimelig simpel:

1. Lav en venstre `edge_rasterizer` (`left`)
2. Lav en højre `edge_rasterizer` (`right`)
3. Få koordinaterne $(x_l, y_l) =$ (`left.x()`, `left.y()`)
4. Få koordinaterne $(x_r, y_r) =$ (`right.x()`, `right.y()`)
5. Hvis der er punkter på skanlinien, altså hvis $\{(x_l, y_l) \dots (x_r - 1, y_r)\} \neq \emptyset$, tegn disse pixels
6. Lad de to `edge_rasterizer`s gå et skridt op af deres kant
7. Hvis der stadig er flere `fragments` tilbage, gå til 3, ellers returnere falsk

Vi skal altså blot vide hvordan vi klarer disse skridt og så kan vi tegne vores trekanter. De to første siger næsten sig selv, og dog, da vi lige skal finde ud af hvilken en der skal have to linier, som kant. Her må vi kigge på de fire typer jeg beskrev før. De to første, med vandrette kanter, er nemme nok, her kan vi simpelthen blot smide hele den kant væk, og kun bruge de to andre. Vi skal dog huske at den nederste kant stadig skal tegnes.

Med de to andre er det lidt sværere. Her skal vi finde ud af om det er højre eller venstre side, som har to linier. Først må vi finde ud af hvordan vi skal navngive punkterne i disse kanter. Hvis vi kalder dem `lower_left`, `upper_left` henholdsvis `the_other` og lader som om de er 3-dimensionele vektore, med z-værdi lig nul, kan vi tage krydsproduktet af de to linier, som bliver udspændt af `lower_left` og `upper_left` henholdsvis `lower_left` og `the_other`, og derved finde ud af hvilken type trekant de er. Hvis krydsproduktet er 0 betyder det at de to kanter er co-linær, og det er altså ikke en trekant vi har fat i, men derimod blot et liniestykke. Er krydsproduktet negativt betyder det, at det er den venstre kant, som er to linier og sidst, hvis krydsproduktet er positivt, betyder det at det er den højre kant, som er to linier.

Nu er det altså blot at tegne alle pixels mellem de to `edge_rasterizer`s, og så er trekanten tegnet.

4.1 Lidt kode at slutte af på

Til sidst vil jeg blot vise resultatet, i form af billeder, samt en lille stumpkode, navnligt `next_fragment()` og en hjælpefunktion, `SearchForNonEmptyScanline()`, som bruges til at finde den næste skanline, som indeholde pixels, som skal tegnes.

Først `next_fragment()`

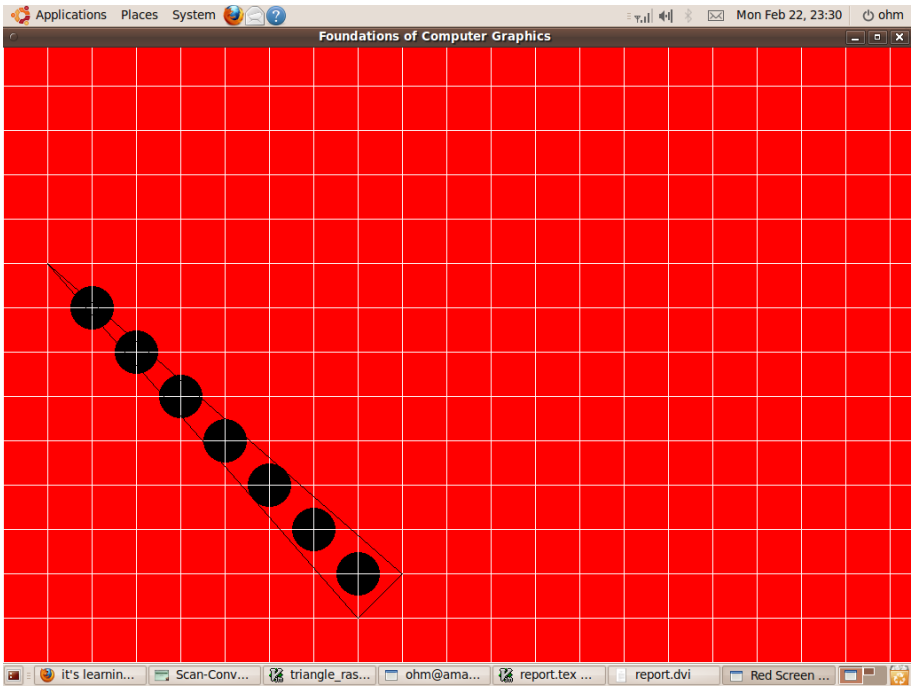
```
1  void next_fragment()
2  {
3      if( this->valid &&
4          this->left.more_fragments() &&
5          this->right.more_fragments() ) {
6
7          if( this->y_current > this->left.y() &&
8              this->y_current > this->right.y() ) {
9
10             this->valid = false;
11         } else if( this->x_current >= this->right.x() - 1 ) {
12
13             if( !this->SearchForNonEmptyScanline() ) {
14                 this->valid = false;
15             }
16
17         } else {
18             this->x_current += 1;
19         }
20     } else {
21         this->valid = false;
22     }
23 }
```

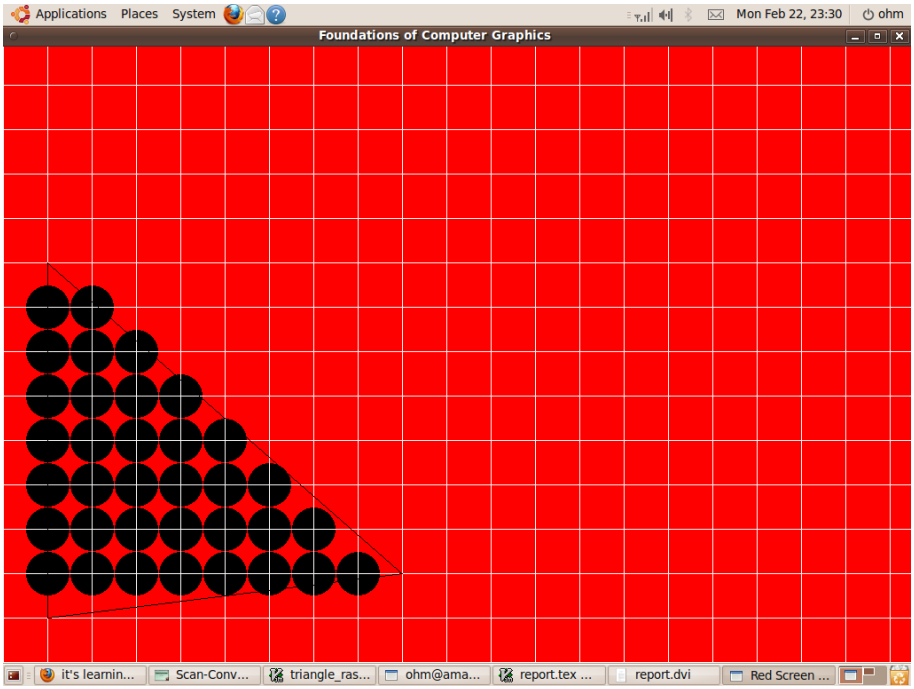
Her løber vi alle de pixels igennem, som skal tegnes, en af gangen. Jeg gør her brug af min hjælpefunktion `SearchForNonEmptyScanline()`, som ser således ud

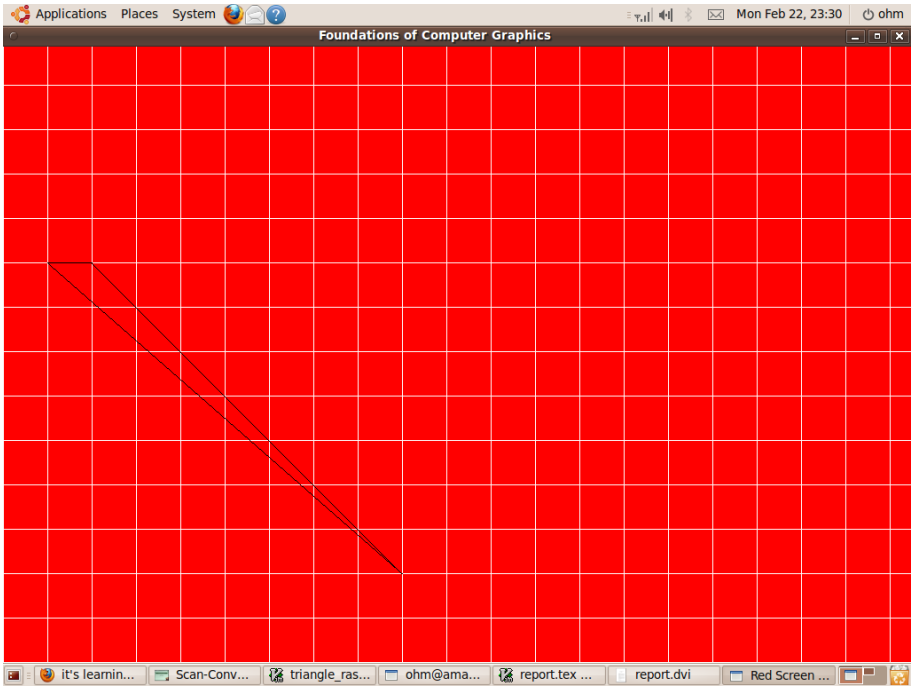
```
bool SearchForNonEmptyScanline()
{
    do {
        this->left.next_fragment();
        this->right.next_fragment();

        if( this->left.more_fragments() &&
            this->right.more_fragments() ) {

            this->x_current = this->left.x();
            this->y_current = this->left.y();
        } else {
            break;
        }
    } while( this->x_current > this->right.x() - 1 );

    return this->valid &&
           this->left.more_fragments() &&
           this->right.more_fragments();
}
```

Denne returnere sand, hvis den finder en ikke-tom skanline, samt sætter x- og y-koordinater for den pixel, som er længst til venstre på denne skanline.

Kører man `framework` og trykker `T` nogle gange, kommer man frem til følgende figurer, som også er vedlagt denne rapport i en større størrelse:




Applications
Places
System
Mon Feb 22, 23:30
ohm
Foundations of Computer Graphics
it's learnin...
Scan-Conv...
triangle_ras...
ohm@ama...
report.tex ...
report.dvi
Red Screen ...


Applications
Places
System
Mon Feb 22, 23:30
ohm
Foundations of Computer Graphics
it's learnin...
Scan-Conv...
triangle_ras...
ohm@ama...
report.tex ...
report.dvi
Red Screen ...