Toepassingen van Meetkunde in de informatica
Project: Snijdende cirkels

Spring 2017

Willem Verheyen (r0624869)
Sjaan Vandebeek (r0624482)

1. Inleiding

We beschrijven kort de drie algoritmes en hun complexiteit. Voor elk algoritme gebruiken we dezelfde manier om de snijpunten tussen twee cirkels te berekenen. We berekenen de afstand tussen de middelpunten van de twee cirkels, indien deze afstand groter is dan de som van de stralen van de cirkels zijn er geen snijpunten. Bij twee gelijke cirkels worden er geen snijpunten berekend.

Beschouw cirkel een $c1$en cirkel twee $c2$:

$$x1 = x - coördinaat\ middelpunt\ c1$$
$$y1 = y - coördinaat\ middelpunt\ c1$$
$$x2 = x - coördinaat\ middelpunt\ c2$$
$$y2 = y - coördinaat\ middelpunt\ c2$$

Afstand tussen de cirkels:

$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$
$$d1 = \frac{(c1.straal)^2 - (c2.straal)^2 + d^2}{2d}$$
$$h = \sqrt{(c1.straal)^2 - (c2.straal)^2}$$

hulppunten:

$$x3 = x1 + \frac{d1(x2 - x1)}{d}$$
$$y3 = y1 + \frac{d1(y2 - y1)}{d}$$

Eerste snijpunt:

$$x4 = x3 + \frac{h(y2 - y1)}{d}$$
$$y4 = y3 - \frac{h(x2 - x1)}{d}$$

Tweede snijpunt enkel indien de som van $c1$en $c2$niet gelijk is aan de afstand tussen hun middelpunten:

$$x5 = x3 - \frac{h(y2 - y1)}{d}$$
$$y5 = y3 + \frac{h(x2 - x1)}{d}$$

We hebben hiervoor volgende bron gebruikt:
https://math.stackexchange.com/questions/256100/how-can-i-find-the-points-at-which-two-circles-intersect

2. Algoritmen

2.1. Algoritme 1

Algorithm 1 brute force

```
1: Input: cirkels: lijst met n cirkels
2: for elke cirkel1 in cirkels do
3:     for elke volgende cirkel2 in cirkels do
4:         bereken de snijpunten tussen cirkel1 en cirkel2
5:     end for
6: end for
```

beschrijving: Elke cirkel in de lijst *cirkels* wordt vergeleken met elke andere cirkel in *cirkels*, hiervan worden dan de onderlinge snijpunten berekend. Omdat we elke cirkel met elke andere cirkel vergelijken voeren we $\frac{n(n-1)}{2}$ vergelijkingen uit. Dit komt overeen met O(n^2).

2.2. Algoritme 2

Algorithm 2 doorlooplijn

```
1: Input: events: gesorteerde lijst van de x-coordinaten van begin en eindpun-
   ten van cirkels
2: T ← ∅
3: for iedere event in events do
4:     if event is de x-coordinaat van het linkereindpunt van cirkel S then
5:         for iedere cirkel in T do
6:             bereken de snijpunten tussen de cirkel die hoort bij event en cirkel
7:         end for
8:         VoegToe(T,S)
9:     else
10:        Verwijder(T,S)
11:    end if
12: end for
```

beschrijving: Voor het tweede algoritme maken we gebruik van een *sweepline* die van links naar rechts loopt. Wanneer de lijn aan het begin van een cirkel komt, (Het meest linkse punt van die cirkel) dan wordt de cirkel vergeleken met alle cirkels die op dat moment gesneden worden door de *sweepline,* m.a.w. Al de cirkels die zich in *T* bevinden, een lijst van actieve cirkels. Na het vergelijken wordt de cirkel ook toegevoegd aan *T*. Wanneer de *sweepline* aan het einde van een cirkel komt, (het meest rechtse punt) dan wordt de cirkel verwijderd uit *T*.

Het algoritme zal in de meeste gevallen een sneller uitvoertijd hebben dan bij algoritme 1 omdat niet elke cirkel enkel wordt vergeleken met al de actieve cirkels. De uitvoertijd hangt dus af van de grootte van de lijst. Wanneer de lijst van actieve cirkels al de cirkels bevat dan zal het

algoritme even snel zijn als het eerste algoritme. Algoritme twee heeft dus een worst case complexiteit van O(n^2).

2.3. Algoritme 3

Algorithm 3 geavanceerde doorlooplijn

```
1: Input: events: gesorteerde lijst van de x-coordinaten van begin en eindpunten van cirkels
2: T ← ∅
3: straal: gesorteerde lijst van de cirkels in T op de straal
4: MaxStraal die de straal bijhoudt dan de cirkel met de grootste straal in T
5: for iedere event in events do
6:     if event is de x-coordinaat van het linkereindpunt van cirkel S then
7:         VoegToe(T,S)
8:         VoegToe(straal,S)
9:         for iedere cirkel in T die als y-coordinaat tussen y-coordinaat van
10:            S + MaxStraal en y-coordinaat van S + MaxStraal liggen do
11:            bereken de snijpunten tussen de cirkel die hoort bij event en cirkel
12:        end for
13:        if de straal van S groter is dan MaxStraal then
14:            MaxStraal is gelijk aan de straal van S
15:        end if
16:    else
17:        if Straal van cirkel D gelijk aan MaxStraal then
18:            MaxStraal is gelijk aan het eerste element uit straal
19:        end if
20:        Verwijder(T,S)
21:        Verwijder(straal,S)
22:    end if
23: end for
```

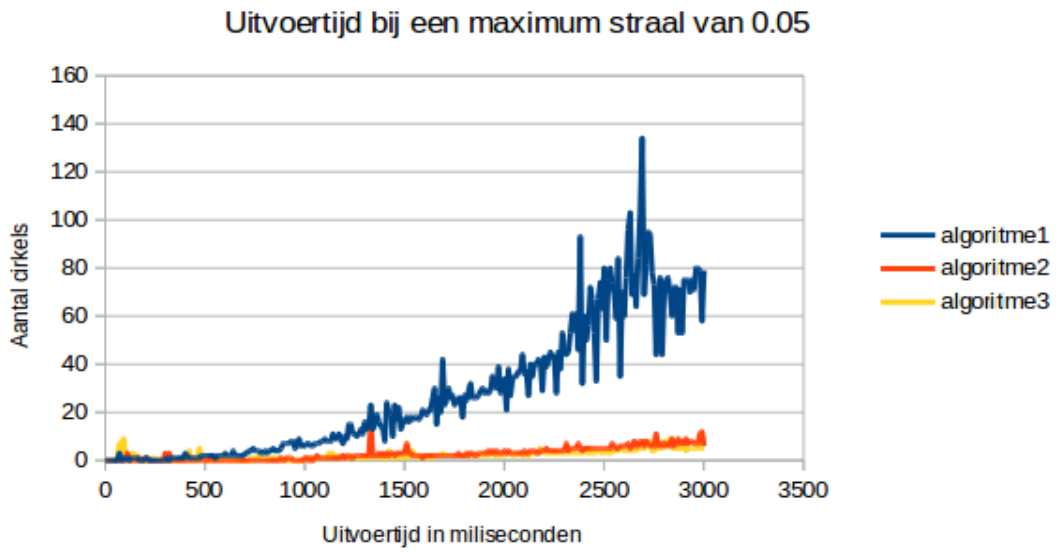
beschrijving: We doen hetzelfde als bij *algoritme 2* en lopen m.b.v. een doorlooplijn door de cirkels. We gebruiken een zoekboom die cirkels sorteert op grootte van de straal (genaamd *straal*) en een zoekboom die cirkel sorteert op de y-coördinaat van het middelpunt. (genaamd *T*) Wanneer de *sweepline* aan het begin van een cirkel komt dan voegt die deze cirkel toe aan de zoekboom *T*. Er wordt een subset van de zoekboom genomen zodat alle middelpunten van de cirkels in de subset binnen op een afstand ligt die kleiner is dan de straal van de cirkel en de grootste straal van de cirkels in de zoekboom straal. Alle cirkels in die subset worden vergeleken met de cirkels die we net hebben toegevoegd. Daarna wordt de cirkel toegevoegd aan de zoekboom die sorteert op stralen. Wanneer de *sweepline* aan het eindpunt van een cirkel komt verwijdert die de cirkel uit de zoekbomen.

3. Experimenten

We voeren on programma uit op een stijgende invoer van cirkels. We beginnen bij 1 cirkel en verhogen het aantal per 10 tot 3000 cirkels. De resultaten stellen we voor op een grafiek waarbij we als x-as de uitvoertijd en als y-as het aantal cirkels gebruiken. Er worden 3 experimenten opgesteld, bij elk experiment wordt de maximale straal verhoogd, deze nemen respectievelijk een maximale waarde aan van 0.05, 0.2 en 0.5 .

3.1. Experiment straal 0.05

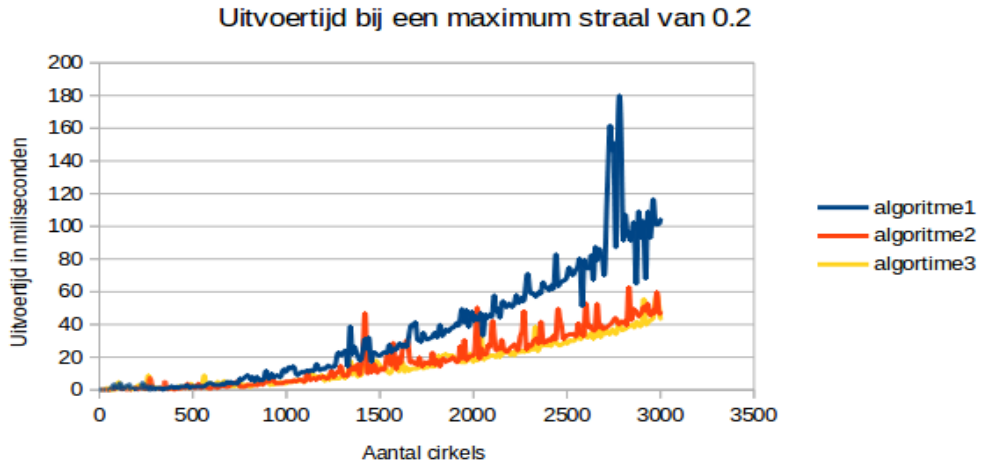
We zeggen dat de straal een maximale waarde van 0.05 kan aannemen.



We zien dat *algoritme 2 en 3* het hier veel beter doen dan *algoritme 1*. De reden hiervoor is dat *algortime 1* elke cirkels met elkaar vergelijkt terwijl *algoritme 2 en 3* dit niet doen. *Algoritme 2* heeft een worst case complexiteit van O(n^2), maar aangezien de stralen zo klein zijn heeft dit geen invloed bij zeer kleine stralen. *Algoritme 3* is nog sneller als *algoritme 2* omdat deze ook de *y-as* in rekening houdt.

3.2. experiment straal 0.2

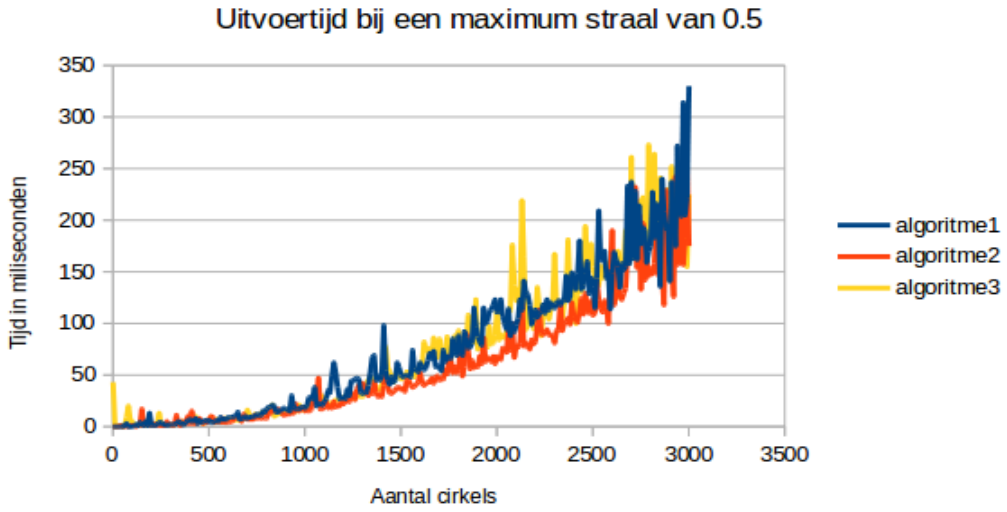
We herhalen het experiment voor een maximum straal van 0.2



We merken op dat de uitvoertijd van de drie algoritmes hoger is. De reden hiervoor is het grotere aantal snijpunten. Het bereken van de snijpunten duurt namelijk langer dan het controleren of twee cirkels snijden. We merken ook nog op dat de curven van *algoritme 2* en *3* dichter liggen bij de curve van *algoritme 1*. Dit komt omdat er meer cirkels tegelijkertijd de doorlooplijn snijden.

3.3. experiment straal 0.5

We plotten dit nu voor een maximum straal van 0.5. We verwachten dat de curve van *algoritme 2 en 3* nog meer opschuiven richting de curve van *algoritme 1* omdat de optimalisatie minder praktisch nut zal hebben. Ook de uitvoertijd ligt hoger omdat voor bijna elk paar cirkels snijpunten berekend worden.

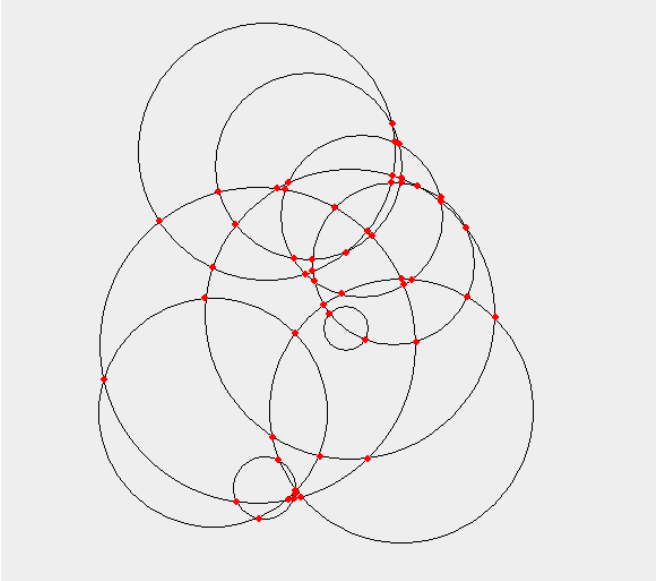


Het resultaat ligt volledig in de lijn van de verwachtingen.

We zien dat *algoritme 3* niet veel efficiënter is dan *algoritme 2*. Dit komt omdat we een grens de y-as zetten. De cirkels moeten namelijk tussen 0 en 1 liggen. Indien we deze grens groter zouden maken dan zou *algoritme 3* veel meer effect hebben.

3.4. Correctheid

We bewijzen de correctheid van de algoritmes door het principe van het bewijs door waarneming. We stellen de cirkels waarvan we de snijpunten berekenen voor op een figuur, vervolgens berekenen we de snijpunten van deze cirkels en stellen deze ook voor op dezelfde figuur. Indien de algoritmes correct zijn, zullen de snijpunten die we berekend hebben overeen komen met de snijpunten tussen de cirkels die we zien op de figuur.



Figuur 1: voorstelling van de cirkels en hun snijpunten

Figuur 1 toont ons dat de snijpunten correct zijn berekend, deze figuur is hetzelfde voor al de drie algoritmes.