Antwoorden Tentamen Geometrische Algoritmen, 25 oktober 2001

1. HoogsteVertex(face-object f)

```
1. if Outercomponent(f) = NULL
2. then for all e in list InnerComponents(f)
3.         do hoogste = Origin(e)
4.            ego = next e
5.            while ego ≠ e
6.            do if origin(ego) hoger dan hoogste then hoogste = origin(ego)
7.               endif
8.            ego = next(ego)
9.            endwhile
10.        endfor
11. else e = OuterComponent(f)
12.        hoogste = Origin(e)
13.        ego = next e
14.        while ego ≠ e
15.           do if origin(ego) hoger dan hoogste then hoogste = origin(ego)
16.              endif
17.           ego = next(ego)
18.        endwhile
19. return hoogste
```

Studenten die voor de onbegrensde face hebben gezegd dat NULL teruggegeven wordt, maar regels 11-19 goed hebben, hebben toch het volle punt gekregen. De rest heeft hoogstens 0.5 punt, en bij meerdere slordigheden of fouten of inefficienties 0 punten.

2. *De opgave is 2.13 uit het boek, die ik ook opgegeven heb als te oefenen opgave.*

Ik ben vergeten in de opgave te vermelden dat de te berekenen $n-1$ lijnstukken ook de driehoeken niet mogen snijden. Bij studenten die (tegen mijn bedoeling) de opgave hebben beantwoord zoals die gesteld is heb ik niets afgetrokken, behalve als hun antwoord niet correct was bij die probleemstelling.

(a.) De statusstructuur is een gebalanceerder binaire zoekboom die voor elke positie van de sweepline, alle doorsneden edges van de driehoeken op volgorde van snijpunt in de bladeren heeft. Bij elke edge is een helper opgeslagen, die een vertex is en wel de laagste vertex boven de sweepline die via een horizontaal lijnstuk met de edge verbonden kan worden zonder enige driehoek te snijden. Als zo'n vertex niet bestaat, dan is de helper de bovenste vertex van de opgeslagen edge. Dit is vrijwel hetzelfde als in hoofdstuk 3 van het boek: polygoon triangulatie. Alleen hebben zowel de linker als de rechter edges van driehoeken een helper. Een linker edge links ervan, en een rechter edge rechts ervan.

Als de statusstructuur leeg is, wordt de laagste gepasseerde vertex in een variabele bijgehouden.

Ik verwacht minstens dat je aangeeft wat voor datastructuur de statusstructuur is, dus een binaire zoekboom, lijst, etc. Veel studenten zijn dit vergeten.

(b.) Er zijn drie typen events.

Bij een bovenste vertex v van een driehoek zoeken we in de statusstructuur naar de edges direct links en rechts van v. Bestaan deze niet (de status is dan leeg), dan genereren

1

we een lijnstuk van v naar de laatst gepasseerde vertex. Bestaat deze niet, dan is v de allereerste vertex en genereren we geen lijnstuk. Bestaat er wel een edge links of rechts van v, dan kiezen we er een van, en genereren we een lijnstuk van v naar de helper van die edge. Verder passen we de statusstructuur aan: twee invoegingen van edges, met v als hun helper, en verder zetten we de helpers van de edges direct links en rechts van v op v.

Bij een middelste vertex v van een driehoek bepalen we de edge direct links of rechts (van een andere driehoek), en zetten de helper van deze edge op v.

Bij een onderste vertex v van een driehoek bepalen we de edges direct links en direct rechts van v, en zetten hun helpers op v. Bestaan beide niet, dan wordt de status straks leeg en maken de v tot laatst gepasseerde vertex. Verder verwijderen we de twee edges die in v eindigen uit de statusstructuur.

(c.) De events zijn alle vertices van boven naar beneden gesorteerd, dit kost $O(n \log n)$ tijd voor sorteren. Het verwijderen van de hoogste event uit de event list kost $O(1)$ tijd als we een lijst gebruiken, of $O(\log n)$ tijd als we een gebalanceerde zoekboom of heap gebruiken.

Elke event geeft aanleiding tot hooguit een constant aantal zoekacties, invoegingen en verwijderingen uit de statusstructuur. Daarom kost elke event $O(\log n)$ tijd om af te handelen.

De tijdgrens van $O(n \log n)$ volgt, want er zijn $3n$ events.

3. *Dit is het bewijs op bladzijde 79 van het boek, en ook precies behandeld op het college.*

   De kans is $\frac{2}{i-2}$ als er precies twee lijnen van $\ell_3, \ldots, \ell_i$ door p_i lopen.

   De kans is $\frac{1}{i-2}$ als er meer dan twee lijnen van $\ell_3, \ldots, \ell_i$ door p_i lopen, en precies één daarvan is links begrenzend, of precies één daarvan is rechts begrenzend.

   De kans is 0 als er meerdere lijnen door p_i gaan, en er zijn minstens twee links begrenzende lijnen en minstens twee rechts begrenzende lijnen.

   Bij alle gevallen worden horizontale lijnen gerekend tot rechts begrenzende lijnen als ze hun halfvlak van onder begrenzen.

4. (a.) Een kd-boom is alleen efficient als zoekacties worden gedaan met rechthoeken, of horizontale of vertikale lijnen (de opgegeven opgave 5.5a van het boek). Dan is de zoektijd $O(\sqrt{n} + k)$, met k het aantal antwoorden. Voor lijnen met willekeurige orientaties is de zoektijd linear. Zoeken met m lijnen kost dus $\Theta(mn)$ tijd in het slechtste geval. Verder moet de kd-boom gebouwd worden, en dat zorgt voor nog eens $\Theta(n \log n)$ tijd. Als m heel klein, bijvoorbeeld constant zou zijn, dan heeft de bouwtijd de overhand in de tijdgrens. Samen dus $\Theta(mn + n \log n)$ tijd.

   Bij studenten die opmerkten dat de zoektijd $O(\sqrt{n} + k) = O(n)$ is in de worst case heb ik het antwoord goed gerekend; het eindantwoord is goed. Ook al is het voor het op te lossen probleem nogal onzinnig om door te gaan met rapporteren als je al een punt op een lijn gevonden hebt; het antwoord is immers al bekend!

   (b.) Een arrangement met m lijnen wordt gebouwd in $\Theta(m^2)$ tijd (hoofdstuk 8). Zo'n arrangement is een subdivisie bestaande uit $\Theta(m^2)$ lijnstukken. Een point location structuur daarvoor wordt gebouwd in $\Theta(m^2 \log(m^2)) = \Theta(m^2 \log m)$ tijd verwacht in het slechtste geval (hoofdstuk 6). Alle n zoekacties in deze structuur kosten per stuk verwacht $\Theta(\log m)$ tijd in het slechtste geval. Daarmee kost deze aanpak $\Theta(m^2 \log m + n \log m)$ tijd verwacht in het slechtste geval.

(c.) De oplossing uit (b.) is duur in m (kwadratisch met een log-factor) en relatief goedkoop in n (linear met een log-factor). Als we de punten en lijnen dualiseren, en dan de oplossing uit (b.) gebruiken, dan kost het bouwen van de point location structuur precies evenveel als alle zoekacties samen, namelijk $\Theta(n^2 \log n)$. Dat is dan ook het antwoord bij (c.).

Wie het woord 'dualiseren' liet vallen had dit onderdeel direct minstens half goed. Verder leverde de juiste tijdgrens doorgaans ook minstens een halve punt. Dit is volgens mij de enige opgave die enige creativiteit vraagt. Het is verder de inlossing van de belofte dat dualiteit op het tentamen voorkomt, al moest je dat wel verzinnen.

5. *Dit is opgegeven oefenopgave 10.8 uit het boek. Bovendien is het 't voldoen aan de belofte gemaakt op het laatste college dat ik iets over segmentbomen zou vragen op het tentamen.*

(a.) Neem als hoofdboom een segmentboom op de x-projecties van de rechthoeken. Bij elke knoop van de hoofdboom hoort een kanonieke verzameling van deze x-projecties; de bijbehorende rechthoeken worden in een geassocieerde structuur van deze knoop opgeslagen in een intervalboom.

(b.) Een segmentboom op n segmenten kost $O(n \log n)$ geheugen mits de geassocieerde structuren worden opgeslagen in een datastructuur die een lineaire hoeveelheid geheugen vraagt (lineair in de grootte van de kanonieke deelverzameling natuurlijk). Dat is het geval voor een intervalboom, dus $O(\log n)$ is het antwoord.

Een segmentboom als geassocieerde strucuur zou $O(n \log^2 n)$ geheugen kosten; er wordt dan een halve punt bij (a.) afgetrokken (maar niet bij (b.), als daar het juiste antwoord staat).

(c.) Daal met de x-coordinaat van het zoekpunt q af naar een blad van de hoofdboom, en voor elke knoop die je tegenkomt, zoek met de y-coordinaat van q in de geassocieerde intervalboom (code daarvoor staat in het boek, maar moet je wel expliciet geven).

De benodigde hoeveelheid tijd voor een zoekactie is $O(\log^2 n + k)$.