8 pt

a) De midpoints methode loopt over een van de assen a in stappen van 1, en kiest steeds of relatief aan de vorige stap de waarde van de andere as eentje opgehoogt moet worden. Dit impliceert ook dat bij een eenvoudige implementatie de "basis"-as altijd een stap kan plegen terwijl de andere as er twee doet. Oftewel dat $\frac{dy}{dx}$ maximaal 1 kan bedragen. Als $\frac{dy}{dx} > 1$, moet het gespiegelt worden berekent, met een loop in y en beslissing over x. (Met zo bij $\frac{dy}{dx} < -1$)

Maar:

$$4R_y = x^2$$

$$\Rightarrow \frac{dy}{dx} = \frac{1}{2R} x$$

$$\Rightarrow \max\left(\frac{dy}{dx}\right) = \frac{1}{2R} \max(x)$$

$\max(x) = R/2R$

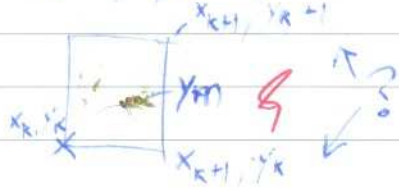
$$\Rightarrow \max\left(\frac{dy}{dx}\right) = 1$$

Met zo valt ook aan te tonen dat $-1 \leq \frac{dy}{dx}$

dus $-1 \leq dy/dx \leq 1$

In dit voorbeeld wordt alleen over x geïtereerd. 8

6 in een punt (x_k, y_k) op de rasterizatie van de parabool moet dus worden beslist of het volgende punt $(x_k + 1, y_k + 1)$ wordt of $(x_k + 1, y_k)$.



Het midpunt is dus $(x_{k+1}, y_k + \frac{1}{2})$

De vraag is dus of y_{k+1} beter benadert wordt door y_k of door y_k+1. Het midpunt nemende, is duidelijk dat als de lijn erboven verloopt, y_k+1 goed is en y_k anders. De parabool interpreterende als nulijn van $F(x,y)$ uit de opdracht, kun je zien dat als $F(x_{m}, y_m)$ positief is, de parabool onder y_p verloopt, en als $F<0$, anders om.

Verblijft nog F te berekenen.

Het eerste beslispunt is $(1, \frac{1}{2})$, waar $F = 2R - 1$.
Het volgende punt is telkens $(x+1, y+1)$ of $(x+1, y)$

Je moet het midpunt nemen

$$F(x+1, y) = 4Ry - (x^2+2x+1) = F(x,y) - 2x - 1$$

$$F(x+1, y+1) = 4R(y+1) - (x^2+2x+1) = F(x,y) + 4R - 2x - 1$$

$\Delta F_{+0} = -2x - 1$ als y niet opgehoogd wordt een
$\Delta F_{+1} = 4R - 2x - 1$ als y wel opgehoogd wordt

helaas bevat ΔF nog een referentie aan x, ook deze valt weer te werken:

$\Delta\Delta F = -2$.
Al deze berekeningen zijn gewoon geheeltallig!

Dus in pseudocode: (erg C-achtig)

```
int x=0, y=0, F=2xR-1, dF0=1, dF1=4R-1

putpixel(x,y)

while(x < 2·R) {
  if(F>0) {
    F += dF0;
  } else {
    F += dF1; y += 1;
  }
  x += 1; dF0 -= 2; dF1 -= 2;
  putpixel(x,y); putpixel(-x,y);
}
```

Let ook dat gebruik gemaakt is van de ~~symmetry~~ symmetrie rond de y-as om zowel op $-x$ als $+x$ te tekenen. ?

1b) Het algoritme as-is rekent telkens nieuwe paren $(-x, y)$ en (x, y) punten op. Hiertussen moet worden opgevuld. Dit elke ~~keer~~ doen is ~~overbodig~~ kan doen is overbodig, immers veranderd y meestal niet. We gaan dus alleen vullen als y geïncrementeerd wordt. ?

?

Voeg gewoon toe dus meteen na "y++;" en nog voor de sluit accolade:

```
for(i=-x; i<=x; i++) putpixel(i,y);
```

nu worden de eindpunten nog 2x getekend

i moet in de header als int gedeclareerd worden.

In dit ~~algo~~ algoritme is geen pixel twee keer getekend.

2) a) Voor elke pixel in ons beeld, ~~gaat~~ schiet er een ray doorheen.

Voor elke ray, ~~gaat~~ bereken de dicht-bij zijnde snijpunt met de scene. De licht waarde van de ray is de emissie van dit object in de scene, plus de som van alle zichtbare lichtbronnen in diffuse belichting ~~gaat~~ (dwz ook spec. maar afhankelijk van je belichtings model), plus de licht-termen van alle subrays. De hele optelling is gewogen afhankelijk van van je belichtings model, wat te maken heeft met het materiaal van het geraakte object. Er kunnen ook termen in zitten voor allemaal andere uitbreidingen; bijv zou je een distance attenuation kunnen doen van alle rays om fog te simuleren. subrays kunnen weer voor vanalles zijn zoals reflecties en

diffractie voornamelijk. Dus in pseudo code:

foreach pixel:
    calc: ray through eye and pixel, find closest intersection
    pixel = ray.shoot();

ray.shoot(): don't process if recursed "too far"
    color = weighted color model average of:
        - surface emission
        - diffuse reflection (foreach light source)
        - ambient lighting
        - sub-ray colors for: (so calc. sub-ray directions + intersections)
            - reflection (possibly multiple)
            - diffraction
    return color; (maybe attenuated by distance / atmospheric term)

de recursie is niet evident in deze pseudocode

Er is veel flexibiliteit voor uitbreiding in dit recursief model. Je zou bv voor beeld van elke subray voor je em zelf evalueerd, zijn coefficient uit rekenen en op basis daarvan en niet recursie diepte beslissen wel of niet verder te rekenen.

b) Arbitraire algebraische oppervlakken kunnen niet zondermeer in een gesloten expressie (dus zonder trial and error zoektocht) worden opgelost. Dit is wel mogelijk als de graad laag is. Substitutie van y en z bijv. levert dan een oplossingen set. Omdat lijnen (uiteraard) lineair zijn, is substitutie altijd mogelijk en blijft de graad van het resultaat ook gelijk. ?

5

Normalen zijn berekenbaar dmv. partiele differentiatie.
Er zijn formules om lage graads polynomen op te lossen, deze zijn gewoon opzoekbaar.
Als je bijv. de part. dif met resp. tot x en tot y uitrekent, krijg je twee vectoren die allebij parallel aan de oppervlakte zijn maar onderling orthogonaal. Met een kruisproduct kun je dan de normaal vinden. Het is niet de bedoeling om in de code part. diffs. enzo uit te rekenen, dit kunje van tevoren zodat al deze stappen gewoon reduceren tot het berekenen van een polynoom.

$\underline{n} = \left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}\right)$

Dus in pseudo code:

Studentnummer: 1148044
Adres:
Studierichting:
Postcode en
Woonplaats:
Jaar van eerste inschrijving:

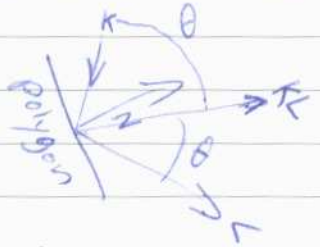
c) Bounding boxes 8 bereken alleen die snijpunten die überhaupt in de buurt komen van je object, en depth-sorting als je object volledig achter een ander object zit, hoef je dat snijpunt ook niet uit te rekenen - terwijl normaliter je alle snijpunten vindt en dan de meest dichtbijzijnde kiest. Werkt niet bij transp. objecten

3

d) Radiosity 8 Deze methode splijt de scene op in vlaktes in negros, en berekent hoe goed elke oppervlak elke andere kan "zien". Met deze informatie, en een vrij simpel belichtingsmodel met absorptie en emmissie en een "zien"-gebaseerde diffuse reflectie, kun je een heleboel simultane vergelijkingen maken. Deze worden meestal gewoon iteratief opgelost, en je hebt per oppervlak een diffuse licht waarde.
1e deel vraag niet beantwoord.

2

3. a) Gouraud en Phong verschillen subtiel. Beiden proberen de gereflecteerde lichtintensiteit te berekenen door de normaal van de oppervlakte van het polygoon en het semidelde tussen kijk en licht te nemen en hiervan een inproduct te nemen en dit tot een macht nemen. dus: $(KL \cdot N)^p$ met KL en N genormaliseerd.



θ moet gelijk θ maar het diagram is scheef.

4

Maar Gouraud maakt deze berekening per vertex en interpoleert de pixelkleurwaardes dan, en Phong interpoleert de vectors en berekent dan pas het resultaat. 9

b) Als de highlite in het inwendige van het polygoon valt zullen alle vertexes donkerder dan de echte highlite. Maar Gouraud-shading interpoleert kleuren en zal dan ook een (vaak veel-) te donker resultaat

10

Omdat het highlite zich vaak weleens tussen vertex-nabijheid en centrum van een polygoon beweegt en je dan (vooral bij een hoge macht p) een irritant flitsen ziet (flicker).

c) Om de berekening goed uit te voeren heb je de licht vector en de kijk vector nodig. Bij gouraud-shading maakt het dus niet echt uit wanneer je dit doet (maar intern moet je wel al een camera positie en licht positie hebben), maar Phong is per pixel en moet dus perse tijdens de rasterisatie, en dus na de kijk transformatie. 10