Computer Graphics: 3D Lijntekeningen met Cilinders en Bollen & Uitbreidingen op Texture Mapping

Jakob Struye
Tim Leys

3D Lijntekeningen met Bollen en Cilinders (0.75 punten)

1. Implementeer code die op basis van een willekeurige lijntekening een 3D-Figuur kan genereren waarin de lijnen en de punten van de lijntekening door respectievelijk cilinders en bollen worden weergegeven.
2. Pas je engine aan zodat deze, op basis van het bovenstaande, zowel platonische lichamen als willekeurige lijntekeningen kan genereren waarvan de zijden en hoekpunten zijn vervangen door respectievelijk cilinders en bollen. De deze nieuwe figuur wordt gegenereerd op basis van een niet-getrianguleerd lichaam.

Invoerformaat

Voor deze opgave introduceren we nieuwe figuurtypes:

- '*ThickLineDrawing*'
- '*ThickCube*'
- '*ThickDodecahedron*'
- '*ThickIsocahedron*'
- '*ThickOctahedron*'
- '*ThickTetrahedron*'

Als je engine het renderen van 3D L-Systemen en/of de BuckyBall ondersteunt, dan moet ook de overeenkomstige 'dikke' variant ondersteund worden:

- '*Thick3DLSystem*'
- '*ThickBuckyBall*'

Het `type` veld uit de `General` section is bij deze oefening ofwel '*Wireframe*', '*ZBufferedWireframe*', '*ZBuffering*' of '*LightedZBuffering*'.

De parameters van deze figuren zijn dezelfde als die van hun normale varianten. Elk van deze figuren bevat echter nog 3 extra parameters:

- **radius** (double): De straal van de cilinders en bollen voor schaling.
- **n** (integer): Het aantal zijvlakken van de cilinder.

- **m** (integer): Het aantal iteraties dat moet worden uitgevoerd tijdens het genereren van de bol uit een icosahedron.

Merk op dat het schalen van de figuur moet gebeuren *nadat* de figuur getekend is. Dwz. dat als de `radius` parameter 0.1 is en de schaalfactor 10, de uiteindelijke straal van de cilinders en de bollen 1 moet worden.

Ter illustratie wordt hieronder een voorbeeld van een configuratiebestand gegeven:

```
[General]
type = "LightedZBuffering"
size = 1000
eye = (100.0, 50.0, 75.0)
backgroundcolor = (0, 0, 0)
nrFigures = 1
nrLights = 1

[Figure0]
type = "ThickCube"
scale = 1.0
rotateX = 0
rotateY = 0
rotateZ = 0
radius = 0.1
center = (0,0,0)
ambientReflection = (0.0, 1.0, 0.0)
diffuseReflection = (0.0, 1.0, 0.0)
m = 5
n = 180

[Light0]
infinity = TRUE
direction = (0, -100, -100)
ambientLight = (0.25, 0.25, 0.25)
diffuseLight = (0.75, 0.75, 0.75)
```

Input van (UV-)textuurcoördinaten (0.125 punten)

Pas je engine zo aan zodat je UV-paren kan inlezen per vertex (hoekpunt) van een face. Die moeten gebruikt kunnen worden om te kunnen specificeren welk hoekpunt van een face overeenkomt met een specifieke pixel $(i, j) = (round(u \cdot w), round(v \cdot h))$ van een textuurafbeelding van w pixels breed en h pixels hoog. Voor de pixels van elke driehoek van het face dient er geïnterpoleerd te worden tussen de UV-coördinaten van de hoekpunten.
Er wordt geen ini-formaat voor deze opdracht opgelegd en worden geen voorbeeldbestanden gegeven.

Input van normaalvectoren (0.125 punten)

Pas je engine zo aan zodat je normaalvectoren kan inlezen per punt of vertex. Die normaalvectoren drukken per punt van een face uit wat de normaalvector in dat punt is. Voor de pixels van elke driehoek van het face dient er weer geïnterpoleerd te worden tussen de normaalvectoren van de hoekpunten.

Er wordt geen ini-formaat voor deze opdracht opgelegd en er worden geen voorbeeldbestanden gegeven.

Cube Mapping (0.5 punten)

Implementeer cube mapping zodat je engine ook een omgeving rond je figuur kan weergeven. Je neemt daarbij een afbeelding als invoer en omhult de figuur of figuren met een kubus bekleed met die afbeelding. De kleuren van de kubus map je vervolgens terug op de figuur.
Er wordt geen ini-formaat voor deze opdracht opgelegd en worden er geen voorbeeldbestanden gegeven.

Integratie

Om de correcte werking van bollen en cilinders volledig te kunnen evalueren, is Z-buffering met lijnen en met driehoeken nodig. Belichting is niet nodig.
Voor de uitbreidingen op texture mapping is een goed werkende texture mapping uit de vorige opdracht noodzakelijk.

Tips

- Voor het genereren van cilinders en bollen kun je de code van een vorige opdracht hergebruiken. Verder kun je een procedure schrijven die een willekeurige figuur omzet naar bollen en cilinders, je hoeft dit dus niet expliciet voor elke te ondersteunen soort figuur apart te implementeren.
- Genereer de boven- en ondervlakken van je cilinders niet. Deze zijn volledig omschreven door de bollen en dus niet zichtbaar, op mogelijke artefacten na.
- Je kan voor bollen en cilinders voorbeeldbestanden vinden op Blackboard. De andere opdrachten hebben, net zoals texture mapping, geen inputformaat en dus ook geen voorbeeldbestanden. Bij die opdrachten moet je er zelf voor zorgen dat je voldoende ini-files mee instuurt zodat we de werking van deze fuctionaliteit kunnen testen.

Input van UV-coördinaten en normaalvectoren

Om de invoer van normaalvectoren en UV-textuurcoördinaten te vergemakkelijken, kan je gebruik maken van de `OBJParser` die je op Blackboard kan vinden. Die laat je toe om de belangrijkste zaken uit `.obj` en `.mtl` bestanden in te lezen.
Als je die gebruikt, kan je best letten op volgende zaken:

- Regels die beginnen met een `#` zijn commentaar.
- Regels die beginnen met `v`, `vt` en `vn` duiden respectievelijk punten, textuurcoördinaten en normaalvectoren aan
- Elke face (aangeduid met `f`) bestaat uit een lijst van indices, te beginnen **vanaf 1**, in 1 van de formaten *i*, *i/it*, *i//in* of *i/it/in*. Deze indices kan je vervolgens gebruiken in de lijst van punten (i), UV-coördinaten (it) of normaalvectoren (in).
- De indices van een face kunnen bij sommige `.obj` bestanden ook negatief zijn. In dat geval wordt er van achteraan geteld: index -1 is het laatste punt, -2 het voorlaatste, enzovoort.