Graphics and Game Technologie

Samenvatting hoorcolleges voor deeltoets 2

Bas Loyen

October 18, 2020

Contents

7 Solid Modeling

7.1 Object Representatie Methoden

Solid Modeling is het representeren van voorwerpen op een computer. Deze verschillende representaties worden beschreven met de volgende eigenschappen:

1. **Volledig**, elk denkbaar object kan worden gerepresenteerd.
2. **Ondubbelzinnig**, object kan maar op één manier geïnterpreteerd worden.
3. **Uniek**, een object kan maar op één manier gemaakt worden.
4. **Correct**, het is niet mogelijk een incorrect object te maken.
5. **Nauwkeurig**, het object is exact en niet een benadering.
6. **Gesloten**, een object is na transformaties nog steeds correct.
7. **Compact**, het object kan compact gerepresenteerd worden.
8. **Efficiënt**, het object kan snel worden gemanipuleerd en geïnspecteerd.

7.1.1 Triangle Mesh

Dit is een verzameling van verbonden driehoeken/polygonen gedefinieerd door opgeslagen punten. Binnen en buitenkant van een driehoek wordt bepaald door de volgorde waarmee de punten zijn opgeslagen.

Voordelen, het is een makkelijke manier op objecten op te slaan (*Compact*), bijvoorbeeld in een triangle strip. Ook zijn driehoeken (op zichzelf) *efficiënt* te renderen.

Nadelen, binnen en buitenkant van objecten aangeven is ambigu (niet altijd *ondubbelzinnig*). Objecten zijn moeilijk te combineren ($\cap, \cup, -$) en complexe modellen hebben veel driehoeken nodig wat ten kosten gaat van *efficiëntie*.

7.1.2 Sweeps

Bij een sweep definieert men een 2D object en een pad waar het voorwerp loodrecht overheen wordt bewogen. Doormiddel van *extrusion* en *revolvement* over dit pad wordt het 3D object gedefineert.

Voordelen, het is conceptueel eenvoudig en *compact*.

Nadelen, moeilijk *efficiënt* te implementeren, kan *incorrecte* resultaten geven, de *nauwkeurigheid* is afhankelijk van de resolutie van het 2D object en de sweeps zijn niet *gesloten* onder reguliere boolse operatoren ($\cap, \cup, -$).

7.1.3 Implicit Quadric Surfaces

Objecten worden gerepresenteerd door middel van kwadratische vergelijkingen.

Voordelen, zeer *nauwkeurig* en zeer *compact*.

Nadelen, objecten zijn impliciet gedefinieerd, je moet objecten nog berekenen en renderen (*inefficiënt*). Het heeft gelimiteerde toepasbaarheid. Lastig om onregelmatige objecten mee te modelleren.

7.1.4 Constructieve Methoden

Complexe objecten worden gemodelleerd door primitieven te combineren. Primitieven zijn blokken, bollen, cilinders, etc. Het combineren gebeurd aan de hand van reguliere boolse operatoren ($\cap, \cup, -$).

Voordelen, complexe objecten kunnen makkelijk gerepresenteerd worden door een boom van primitieven en boolse operaties.

Nadelen, niet altijd *compact* of *efficiënt* voor complexe objecten. Ook niet altijd *ondubbelzinnig* aangezien de boolse operatoren soms worden aangepast om bijvoorbeeld te bepalen wanneer een rand van een object wel of niet in het resulataat terecht komt.

7.1.5 Boundary Representations (B-reps)

Objecten worden beschreven in termen die hun omhullende beschrijven: punten, randen en zijdes. De omhullende begrenst het interieur en exterieur van het object. Het kan soms echter lastig zijn om te bepalen wat een zijde is.

Veel B-rep systemen zijn **2-manifold**, dit houdt in dat een object door 2D vlakken beschreven kan worden. Als een object *2-manifold* is dan kan je op elke willekeurige plek op dat object een 'bolletje' plaatsen en dan op de doorsnede van dat bolletje en het vlak het object uit kan vouwen tot een 2D vak. Dit geeft de volgende eigenschappen waar *2-manifold* aan meot voldoen:

1. Elke kant wordt door één of twee driehoeken gebruikt.
2. Elk punt is verbonden met één enkele verzameling van driehoeken die aan de rand verbonden zijn.

Een **polyhedron** (veelvlak) is een volume dat is begrensd door een verzameling polygonen waarvan elke rand een even aantal polygonen heeft. Om een object hiermee te beschrijven moet het aan het volgende voldoen:

- Elke rand verbindt twee punten en wordt door precies twee zijden gedeeld
- Tenminste drie randen komen samen in elk punt
- Zijden mogen elkaar niet snijden

Dit voldoet aan de volgende relatie: $V - E + F = 2$, met punten V, randen E en zijden F.
Met gaten: $V - E + f - H = 2(C - G)$, met gaten in zijden H, gaten in object G en componenten C.

7.2 Datastructuren

7.2.1 Polygoon Meshes

Deze datastructuur bevat twee lijsten, een lijst van punten (geometrie) en een lijst met zijden (topologie). Hierbij is de volgorde van de punten belangrijk om aan te geven wat de binnen en buitenkant is.

7.2.2 Winged-Edge

De *winged-edge* datastructuur beschrijft de geometrie en topologie van randen, punten en zijden. Het bestaat uit drie tabellen. In de tabel met randen staat per rand de twee aanliggende punten, de twee aangrenzende zijdes, de linker en rechter voorgaande randen en de linker en rechter volgende randen. De andere twee tabellen slaan voor alle punten en zijdes de randen op die er aan grenzen. Hierdoor is het erg makkelijk om door een object heen te itereren.

7.2.3 Spatial-Partitioning Representaties

Object wordt beschreven door aangrenzende niet overlappende volumes. Dus alleen ($\cup$)

Bij **cell-decomposition** wordt een verzameling primitieven aan elkaar gelijmd tot complexe objecten. Elke twee primitieven delen dus een punt, rand of zijde. Deze representatie is *ondubbelzinnig* maar niet *uniek*.

Bij **constructive solid geometry (CSG)** wordt het object als een boom opgeslagen waarbij de bladeren primitieven bevatten en elke knoop een boolse operatie en transformatie bevat op de twee kinderen van die knoop. *Cell-decomposition* is een vorm van *CSG* met alleen de $\cup$ operatie.

Net als **spatial-occupance enumeration**, hierbij worden objecten weergegeven door identieke blokken/voxels in een regelmatig rooster. De *nauwkeurigheid* is afhankelijk van de cel grootte, maar het is wel *uniek* en *ondubbelzinnig*.

Quadtrees/octrees is voor 2D en 3D respectievelijk. Bij een *quadtree* wordt het vlak verdeelt in kwadranten. Per kwadrant wordt gekeken of het leeg of vol is, zo ja dan is het klaar, als het gemengd is verdeel je de kwadrant weer en ga je recursief door tot alles vol of leeg is. Hetzelfde geld voor een *octree* maar dan met een kubus die verdeelt wordt in octanten. Het is *efficiënt* te verwerken en eenvoudig om boolse operatoren op toe te passen.

	Accuracy	Domain	Uniqueness	Validity checking	Closure
Primitive instancing	Yes	Limited	Possible	Easy	No
Sweeps	Yes	Limited	Possible	Easy	No
b-reps poly	Approximation	Limited	No	Difficult	Yes
b-reps non-poly	Yes	Any	No	Difficult	Yes
Cell decomposition	Approximation	Any	No	Difficult	Yes
Spatial occupancy enumeration	Approximation	Any	Yes	Easy	Yes
Octrees	Approximation	Any	Yes	Easy	Yes
BSP trees	Approximation	Any	No	Easy	Yes
CSG	Yes	Any	No	Easy	Yes

Figure 1: Vergelijking

Een **binary space-partioning (BSP) tree** verdeelt een object hiërarchisch recursief op in twee delen aan weer zijden van een vlak. Elke knoop in de *BSP tree* representeert een vlak, het linker kind van elke knoop is de achterkant en het rechter kind de voorkant van dat vlak. Als een kind niet verder opgedeeld kan worden is het ofwel binnen of buiten (in of out).

Een **scene graph** is een hiërarchische beschrijving van één of meerdere objecten in een 'scene' door middel van een **directed acyclic graph (DAG)**. Elke knoop in de graaf bevat een object en een transformatie matrix om er een geheel van te maken. **Frustum culling** is een techniek die alleen de objecten in zicht weergeeft voor de *efficiëntie*.

8 Texture Mapping

Voor **texture mapping** heb je een object, texture, mapping van object naar texture coördinaten, een sampeling mechanisme en een toepassing van de resulterende waarden nodig. De mapping doe je door voor elk punt in het object een coördinaat (u, v), in bereik [0,1], te definiëren op de texture. Alle tussenliggende punten worden door middel van interpolatie berekend.

Perspective foreshortening is het kleiner weergeven van objecten die verder weg zijn. Dit willen we ook met textures maar dit lukt niet met interpolatie over de scherm-coördinaten. Hiervoor gebruiken we de barycentrische coördinaten, hierdoor kun je alsnog de diepte afleiden die je door de projectie op het scherm was kwijt geraakt.

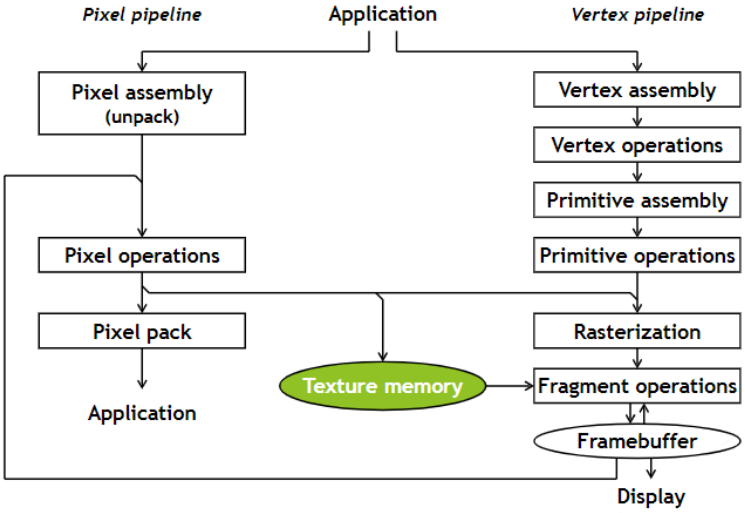

Figure 2: OpenGL Pipeline

Als de texture coördinaten buiten de interval [0,1] vallen kun je gebruik maken van **tiling** of **clamping**. Bij *tiling* wordt het getal voor de komma genegeerd, de texture herhaald zich dan. Bij *clamping* wordt de laatste waarde van de rand gebruikt. Als een pixel veel kleiner is dan een texel van de texture wordt **magnification filtering** gebruikt, dat is "upsampling" met nearest neighbour lineaire interpolatie. Andersom wordt er **minification filtering** gebruikt, ook wel "downsampling".

MIPmapping maakt gebruik van een texture en vooraf berekende verkleinde versier van de texture om artifacts op lange afstand te voorkomen. De opslag is erg efficiënt, alles staat in 1 blok geheugen en er is maar $\frac{1}{3}$ meer opslag nodig.

8.1 Toepassingen van textures

Bump mapping is het nabootsen van een ruwe texture door de normaal vectoren op het oppervlak van een object aan te passen.

Bij **displacement mapping** worden delen van het oppervlak een stukje naar voren verschoven om diepte te creëren. Het moet wel mogelijk zijn om het oppervlak op te delen.

Shadow maps is het vooraf renderen van schaduwen en ze dan als texture op objecten plaatsen. Dit wordt gedaan omdat het erg kostbaar is om in real-time schaduwen te berekenen. **Environment mapping** gebruikt hetzelfde principe als *shadow mapping* maar dan voor reflecties.

9 Graphics Hardware

9.1 Hardware support towards real-time/interactive graphics

Voor real-time graphics is het belangrijk om goede **frame rate** en **refresh rate** te hebben. Hoe hoger beide zijn hoe fijner het is om naar te kijken. Voor interactieve doeleinden is een lage **latency** ook belangrijk. Dit is de tijd tussen actie van de gebruiker en reactie op het scherm. Alle drie zijn voor games van belang.

9.1.1 Graphics in the PC architecture

Vroeger werd gebruik gemaakt van de PCI en AGP bus voor de communicatie tussen CPU (host) en GPU (device), tegenwoordig wordt dat door PCI Express (PCIe) bus gedaan. Dit zorgt voor een theoretische snelheid van 128GB/s (op 16 banen).

9.1.2 Graphics state machine

De **graphics state** is een optimalisatie om te zorgen dat niet altijd alles meegegeven hoeft te worden aan elk element. Als er iets leeg wordt gelaten is de *graphics state* er om het aan te vullen. State changes wil je dus ook zo veel mogelijk voorkomen, die maken het renderen langzamer en kunnen zorgen voor inconsistenties. Door gelijke geometrie zoveel mogelijk samen te renderen kan het aantal state changes worden teruggedrongen.

9.1.3 Strips and fans

Voor storage overhead en memory access coherence wordt er vaak gebruik gemaakt van **triangle strip/fans**. Met deze vorm van opslag kun je met N punten $N-2$ driehoeken definiëren omdat elke opeenvolgende driehoek de twee laatste punten van de vorige driehoek gebruikt. Dit is veel efficiënter dan een **triangle list**, hier zijn N punten nodig voor $N/3$ driehoeken.

9.1.4 Display lists

Dit is een optimalisatie methode waarbij teken commando 's gecompileerd worden en in een compact formaat worden opgeslagen op de GPU. Dit kan echter niet met elk commando en is alleen nuttig voor statische geometrie aangezien aanpassingen er voor zorgen dat de hele lijst opnieuw gecompileerd moet worden. Worden tegenwoordig niet meer gebruikt.

9.1.5 Vertex arrays

Dit is wederom een optimalisatie methode waarbij geometrie niet expliciet geprogrammeerd wordt maar in een array terechtkomt die aan de GPU gegeven wordt. Hierin zitten niet alleen punten maar ook dingen zoals normaal vectoren. Ook dit wordt tegenwoordig niet meer gebruikt.

9.1.6 Vertex Buffer Objects (VBO)

Hetzelfde principe als *vertex arrays* alleen dan op de GPU zelf. Een *VBOs* houden de geometrie en state bij wat zorgt voor minder communicatie tussen CPU en GPU via de bus en significant kortere render tijd. Later werd het ook mogelijk om een mapping te maken van het geheugen van de GPU naar het werkgeheugen waar de CPU gebruik van maakt. *VBOs* worden tegenwoordig nog steeds gebruikt.

9.2 Data and task parallelism

Veel van wat wordt gedaan in graphics is goed te paralleliseren. Er zijn twee soorten parallellisme die voorkomen in de OpenGL vertex pipeline.

Data parallellisme, dit is gelijktijdig hetzelfde doen met gelijke data, zoals het transformeren van punten. Wat variantie in de gelijke data is mogelijk. Dit valt onder **Single Instruction Multiple Data (SIMD)** parallellisme.

Task paralellisme, dit is tegelijkertijd verschillende dingen doen, zoals de verschillende stadia in de OpenGL vertex pipeline.

Multithreading wordt gebruikt om *latency* te verminderen. Alle processen in een streaming processor worden in een thread geplaatst. Een thread is een datastructuur met daarin onder andere de gebruikte program counter. Threads die externe informatie nodig hebben worden in de blocked threads lijst gezet totdat deze externe conditie behaald is.

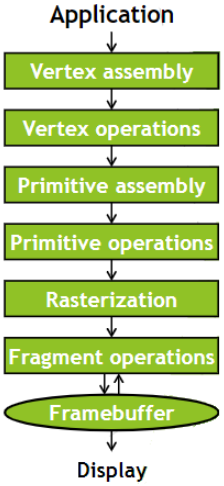

Figure 3: OpenGL vertex pipeline

10 Data Visualisatie

10.1 Visualisatie taxonomie

Bij **wetenschappelijke visualisatie (SciVis)** gaat het om de visualisatie van numerieke gegevens afkomstig van metingen of simulaties. Het is uniform, heeft een duidelijk referentie kader en is er om inzicht te creëren, denk hierbij aan een röntgenfoto.

Bij **informatie visualisatie (InfoVis)** gaat het om het visualiseren van zeer diverse data, zoals uit databases, netwerken, etc. Het is vaak niet uniform, heeft een minder duidelijk referentie kader en is er wederom om inzicht te verkrijgen. Denk hierbij bijvoorbeeld aan metro kaart.

10.2 The visualization pipeline

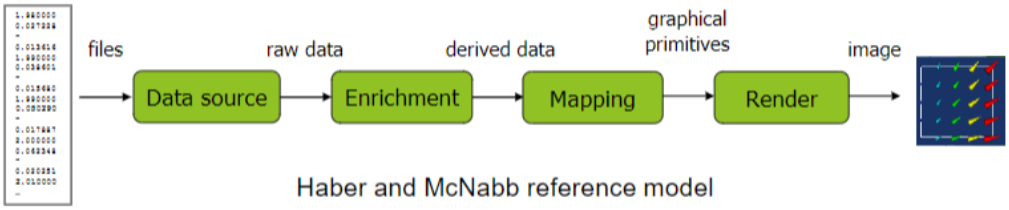

Figure 4: Visualisatie pipeline

De visualisatie pipeline werkt anders dan de graphics pipeline, we hebben het hier over een **data flow netwerk**. De verschillende filters produceren data wat weer wordt doorgegeven aan het volgende filter. De data gaat altijd 'downstream' maar de update checks zijn altijd 'upstream'.

10.3 Glyph visualization

Een **Glyph** is een object getransformeerd door input data. "De moeder van alle visualisatie methodes". Kan soms onoverzichtelijk zijn, vooral in 3D. Dit kan opgelost worden met **random subsampeling**, hierbij wordt maar een deel van een grote dataset weergegeven om patronen weer herkenbaar te maken.

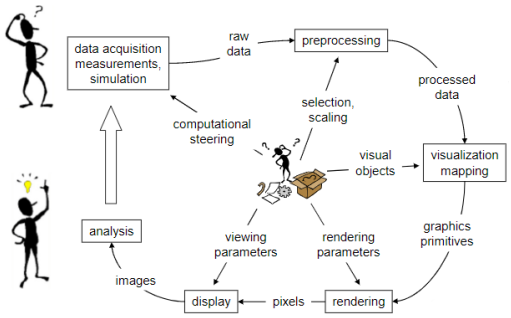

Figure 5: Visualization cycle

10.4 Contour extraction

een **contour** beschrijft de omtrek van een gebied met dezelfde of vergelijkbare waarde. *Contour extraction* kan gedaan worden met bijvoorbeeld **marching squares**. Bij *marching squares* itereer je over (bijvoorbeeld) de grijs waardes van een afbeelding en daarbij plaats je steeds een denkbeeldig vierkant over 4 van deze waardes. Vervolgens bepaal je voor elke van de vier pixels of de waarde boven of onder de contour-waarde zit. Vervolgens wordt de doorsnede van de contour door het vierkant als lijn opgeslagen. Als dit voor alle waardes is gedaan kunnen de lijnstukken getekend worden om de contour weer te geven. In 3D heet dit **marching cubes**. Deze methodes kunnen soms echter problemen veroorzaken waarbij extra informatie nodig is, een methode die dit oplost is **marching tetrahedra**. Hierbij wordt wordt elk 2x2 gebied van *voxels* opgedeeld in 6 tetrahedra waarvan de hoekpunten in de hoeken van de 2x2 kubus zitten.

11 Animatie

Lasseter's 12 fundamentele principes van traditionele animatie(1987):

1. Squash and stretch
2. Timing
3. Anticipation
4. Follow through and overlapping action
5. Slow-in and slow-out
6. Staging
7. Arcs
8. Secondary actions
9. Straight-ahead and pose-to-pose action
10. Exaggeration
11. Solid drawing skills
12. Appeal

11.1 Keyframing

Een **keyframe** is een waarde (of tuple van waardes) op een gegeven moment in een animatie. Dit kan gebruikt worden om belangrijke punten aan te geven en de waardes kunnen dan geïnterpoleerd worden. Dit wordt het meest gebruikt voor de positie van een object in een animatie, maar het kan voor alles gebruikt worden wat met parameters wordt vastgelegd.

11.1.1 Catmull-Rom splines

Een belangrijke eigenschap van een curve is de C^n continuïteit. Dit staat voor de hoeveelste afgeleide waarvoor de beweging continu zou moeten zijn. Om dit te behalen wordt vaak gebruik gemaakt van de **tension, continuity** en **bias (TCB)** waardes. De *tension* gaat over de scherpte van de curve, met *continuity* kun je knikken laten voorkomen in de curve en de *bias* gaat over het 'gewicht' van de punten op de curve.

11.2 Motion capture

Motion capture is een manier om realistische animatie te maken door middel van sensoren op een echt mens/object. Dit is omdat het erg lastig kan zijn hetzelfde te bereiken met *keyframing*. Denk hierbij aan Rob die gitaar speelt.

11.3 Physics-based animation

Hierbij worden de wetten van de natuur gebruikt om een realistische animatie te maken. Hierdoor is het mogelijk om uit te rekenen wat er 'in het echt' zou gebeuren in plaats van dat dit door middel van *keyframing* of *motion capture* moet. De **rigid body solver** berekent wat er met een **rigit body** gebeurd als het in contact komt met andere objecten. Het is belangrijk om hier rekening te houden met welke krachten er spelen op een object, een simulatie waarbij impuls wordt gebruikt komt wel in evenwicht in plaats van een simulatie met constante krachten waarbij alles blijft bewegen. Denk hierbij aan het voorbeeld uit het college met alle rode kubussen die vallen en nog trillen als terwijl je zou verwachten dat ze stil zouden moeten liggen.

12 Interactive Graphics

Interactieve grahics zie je vooral terug in games en simulaties voor bijvoorbeeld vliegtuigen, auto 's, etc. Hierbij is het van belang dat het realistisch en responsief is. Een opkomend gebied waarin dit wordt toegepast in **virtual reality (VR)**. *VR* kan toegepast worden in situaties die voldoen aan het **DICE** principe, *DICE: Dangerous, Impossible, Counterproductive, Expensive*. Maar vooral door games is het zo groot geworden als het nu is.

12.1 Stereographics

Wij nemen diepte waar aan de hand van de volgende eigenschappen. De linker kolom is met allen graphics te simuleren maar de rechter kolom heb je meer voor nodig.

- Occlusion (hidden surfaces)
- Perspective distortion
- Aerial/atmospheric
- Lighting, shadows
- Binocular disparity
- Motion parallax
- Convergence
- Accommodation

Stereo graphics maakt het mogelijk om de eigenschappen uit de linker kolom te simuleren. Dit gebeurd aan de hand van twee beelden, een gegenereerd voor je linker oog en de andere voor je rechter oog.

Bij **anaglyph stereo** wordt dit gedaan met een bril met een rood en blauw filter er in. De twee beelden worden dan ieder in blauwe en rode componenten omgezet zodat ieder oog één beeld ziet. Dit heeft echter het nadeel dat alles in rood en blauw tinten is. **Time frame sequential stereo** lost dit op door een beeld af te spelen wat steeds wisselt tussen links en rechts met een bril die telkens het linker en rechter oog bedekt. Dit is echter erg omslachtig en kan ook passief door middel van **Passive polarized stereo** waarbij een bril met polarisatie filters nodig is. Ook dit heeft een actieve variant waarbij er een filter voor het beeld zit die heel snel de polarisatie verwisselt om hetzelfde resultaat te bereiken. **Stereo graphics** zie je tegenwoordig terug in *VR* en **augmented reality (AR)**. In een *VR* bril is er een klein scherm voor elk oog. Bij *AR* wordt er een beeld geplaatst over de realiteit.