# Samenvatting Graphics en Game Technologie

Lucas Riedstra

16 december 2018

Buiten ray-tracing is *object-order*-rendering een populaire renderingtechniek: bekijk elk object na elkaar, en vind de pixels waar dat object een effect op heeft. Het vinden van die pixels heet *rasterisatie*. De reeks operaties die nodig zijn heet de *graphics pipeline*.

### College 7: Solid Modeling

**Objectrepresentatie** Er zijn verschillende manieren om objecten te representeren:

1. De meeste modellen bestaan uit grote hoeveelheden verbonden driehoeken, ook wel *triangle meshes* genoemd. Dit generaliseert zich naar willekeurige polygonen - wij beperken ons tot driehoeken. Hierbij gelden twee simpele afspraken: Elke edge wordt gebruikt door maximaal twee driehoeken en elke knoop is verbonden aan een reeks driehoeken waarbij elk paar één kant deelt.

   Het is hier ook belangrijk om een "voorkant" en "achterkan" van een driehoek niet-amgibu te definiëren. Een voorbeeld is om de voorkant te definiëren als de kant waar de drie hoekpunten tegen de klok in staan.

   Het voordeel van triangle mesh is dat objecten er makkelijk mee te beschrijven zijn en dat driehoeken efficiënt te renderen zijn. Nadelen zijn dat het moeilijk is om objecten te combineren, dat er meerdere mogelijkheden zijn om binnen-/buitenkant aan te geven en dat er veel driehoeken nodig zijn voor complexe modellen.

2. Een andere manier van objecten weergeven is *extrusie*, ook *sweeps* genoemd: je neemt een 2D-voorwerp, definieert een pad, en beweegt het 2D-voorwerp vervolgens loodrecht over het pad. Wat je dan overhoudt is een 3D-volume. Bijvoorbeeld: om een bol te maken kun je een cirkel roteren. Er zijn twee soorten sweeps: *translational*, waarbij je een object verplaats, en *rotational*, waarbij je een object roteert.

   Een voordeel van sweeps is dat ze (conceptueel) eenvoudig zijn, een nadeel is dat het moeilijk is om efficiënt te implementeren en niet altijd correcte resultaten geeft.

3. Nog een representatiemethode is *quadric surfaces*, waarbij een object gerepresenteerd wordt met een vergelijking (bv een bol weergeven met $x^2 + y^2 + z^2 = 1$). Deze zijn zeer nauwkeurig compact, maar het is moeilijk om er onregelmatige objecten mee te modelleren, die misschien geen (mooie) vergelijking hebben.

4. Een volgende methode is *constructie*: het gebruik van combinaties van primitieve objecten (blokken, bollen, etc.) om een object mee te modelleren. Van voorwerpen gemodelleerd met constructie kan je prima de vereniging ($\cup$) of doorsnede ($\cap$) nemen. Dit gaat niet in elk model even makkelijk, en levert ook niet altijd weer een voorwerp op. Een voorwerp kan *gereguleerd* worden: dit betekent dat de afsluiting van het inwendige van het voorwerp wordt genomen. Een *regulerende* boolse operator, bv $\cup^*$, geeft de gereguleerde versie van $A \cup B$. Bijvoorbeeld: $A \cap^* B$ is altijd een voorwerp (of leeg).

5. De laatste besproken methode is *boundary representations* of B-reps: voorwerpen worden beschreven in termen van hun punten, randen, en zijdes. Vaak worden alleen 2-manifolds toegelaten. Voor een 2-manifold geldt opnieuw (bij driehoeken) de eis dat elke kant door maximaal twee driehoeken gebruikt wordt en elk punt verbonden is met één verzameling driehoeken die allemaal een kant delen.

   Een veelvlak is een volume begrensd door een verzameling polygonen, waarbij elke rand een even aantal polygonen heeft. Elk polyhedron voldoet aan $V - E + F = 2$.

**Datastructuren** Datastructuren om polygonen te representeren:

1. Voor polygonen: een lijst van punten en een lijst van alle zijdes is genoeg, dit heet een *mesh*. De volgorde waarin de hoekpunten worden opgeschreven is belangrijk (vaak tegen de klok in).

2. De mesh geeft geen antwoord op vragen als "welke zijden delen een punt". Daarvoor is de *winged edge*-datastructuur handig: er is een tabel van edges, en voor elke edges worden de twee knopen beschreven, de twee valkken links en rechts, en de voorgangers en opvolgers links en rechts (de kanten hebben dus ook een richting). Vaak wordt ook nog een vertex-tabel bijgehouden (met de bijbehorende edges) en een vlakken-tabel met bijbehorende edges.

3. Bij spatial-partitioning-representaties wordt een voorwerp gerepresenteerd door aangrenzende, niet-overlappende volumes. Voorbeelden zijn:

   - Cell decomposition: primitieven worden aan elkaar "gelijmd". Dit is een speciaal geval van constructieve geometrie waarbij alleen $\cup$ mag worden gebruikt.
   - Spatial-occupancy: cellen worden weergegeven met voxels. Dit geeft een unieke representatie van een object.

- Quadtrees: het vlak wordt verdeeld in kwadranten, en als een in een kwadrant zowel deels wel als deels niet getekend moet worden wordt dat kwadrant weer in vieren verdeeld.
- BSP-trees: op recursieve wijze wordt een voorwerp in delen verdeeld aan verschillende kanten v/e vlak. Bij elke knoop bevat één tak elementen "voor" de snijlijn, en de andere elementen "achter".

**Scenegraphs** Scenegraphs definiëren een scène als een *directed acyclic graph*: een graaf zonder cykels. Door een DAG te gebruiken kunnen objecten deel zijn van meer dan één groep (i.t.t. bij een boom). Een *matrixstack* geeft een manier om matrices toe te voegen/weg te halen uit een matrixproduct. Bij *frustrum culling* worden alleen zichtbare objecten getekend.

### College 8: Texture mapping

Omdat normale belichtingsmodellen alleen voor 'saaie' oppervlakken zorgen, wordt *texture mapping* gebruikt - hiermee kunnen we objecten realistischer maken en complexe geometrieën benaderen. In de rasterisatiefase wordt de kleur van een pixel bepaald door zowel het plaatje als de belichting. We onderscheiden *oppervlaktetextures* (2D) en *volumetrische textures* (3D). Texturing bestaat uit vier dingen: het plaatje, de *mapping* van geometrie naar *image*-coördinaten, het *sampling*-mechanisme, en de *toepassing*:

- Voor de mapping moet er voor elk punt $(x, y, z, w)$ in het object een texture-coördinaat $(s, t)$ in het plaatje zijn.
- Meestal normaliseren we de coördinaten zodat de texturecoördinaten in $[0, 1]^2$ liggen. Bij overflow zijn er meerdere mogelijkheden, zoals *clamping* of *wrapping*.
- Er zijn natuurlijk meerdere manieren om een texture uiteindelijk toe te passen op een oppervlak. We kunnen bijvoorbeeld de belichtingscomponent compleet vergeten (en dan wel of niet transparantheid behouden), of moduleren met de huidige oppervlaktekleur.

De volledige OpenGL-pipeline bestaat uit twee delen: de *pixel pipeline* en de *vertex pipeline*. De pixel-pipeline bestaat uit de fases *assembly, operations*, en *pack*. Assembly betekent het 'uitpakken' van het plaatje uit het geheugen naar een bepaald format. De operaties zijn dingen als offset en kleurbepaling. De vertex-pipeline bestaat uit de fases *assemply, operations, primitive assembly, primitive operations, rasterization, fragment operations, framebuffer, display*. Rasterisatie is de fase van objecten in de goede volgorde op het beeld zetten (hierbij wordt de homogene coördinaat gebruikt).Voor het uitvoeren van *fragment operations* is informatie uit het *texture memory* nodig. Bij texture mapping moeten we ook interpoleren om perspectief goed zichtbaar te maken, dit heet *perspective division*.

Een punt in een texture heet een *texel*. Als een pixel veel kleiner is dan een texel moet er geïnterpoleerd worden (of gefilterd). Als pixels veel groter zijn dan een texel zijn er allerlei vormen van interpolatie mogelijk, waarbij ook *mipmaps* gebruikt kunnen worden. Bij een mipmap slaan we ook verkleinde versies van een texture op, en kiezen we op een bepaalde plek waar we welke image nodig hebben. Voor het opslaan van een MIPmap is slechts 4/3 van het geheugen van het oorspronkelijke texture nodig.

Er zijn verschillende manieren om *texture mapping* te doen:

- Bij *bump mapping* wordt de normaalvector lokaal beïnvloed om "ruwe" oppervlakken weer te geven;
- Bij *displacement mapping* wordt ook de structuur van een oppervlakte veranderd;
- Bij *environment mapping* wordt een plaatje van de omgeving gemaakt om die te gebruiken als "reflectie", simpelweg door het plaatje op een reflecterend object te plakken als een texture;
- Bij *sphere mapping* wordt een texture vervormd en gemapt d.m.v. polaie coördinaten;
- Bij *cube mapping* is een texture de zes zijdes van een kubus, en bij het mappen bepaalt de richting van de reflectievector welke zijde van de kubus wordt gebruikt.

**College 9: Datavisualisatie**

Datavisualisatie is het transformeren van grote hoeveelheden gegevens naar plaatjes. Dit is belangrijk omdat de hoeveelheid gegevens in een plaatje explosief toeneemt. We onderscheiden twee soorten visualisatie: wetenschappelijk (data met een natuurlijke structuur, bv. metingen) of informatief (data met een abstracte structuur, bv. relaties). Data visualiseren is interdisciplinair: je hebt bv. zowel mensen nodig die iets weten van de data als mensen die iets weten van graphics. Er zijn verschillende soorten visualisatie:

- Bij *glyph*-visualisatie transformeren we een object op een of andere manier aan de hand van de input-data. Een simpel voorbeeld is een scatterplot. Soms kan informatie ambigu zijn in interpretatie (vaak bij 3D tekenen), hiervoor zijn oplossingen zoals het toevoegen van *depth cues* (die een idee van diepte geven, bv. schaduwen).
- Bij *contour*-visualisatie tekenen we een bepaalde niveaukromme door de data heen. In 2D bestaat hiervoor het *marching squares*-algoritme, in 3D het *marching cubes*- of *marching tetrahedra*-algoritme. Bij het marching squares-algoritme nemen we aan dat er in het *midden* van elke pixel een waarde zit. Vervolgens zijn er 16 mogelijkheden voor configuraties van of een hoekpunt een waarde $\leq$ of $>$ de contourwaarde heeft. Op basis

daarvan tekenen we lijnstukjes, eventueel lineair geïnterpoleerd. Het marching squares- en cubes-algoritme hebben allebei een ambiguïteit die een verschil kan maken tussen één of meerdere samenhangende delen.

We kunnen de marching cubes-algoritme verbetren door gradiënten te benaderen met behulp van afgeleides.

**College 10: VR en AR**

Virtual reality wordt gebruikt voor simulatie, design, visualisatie, en kunst. Het grootste deel van onze waarnemingen bestaan uit zicht. Voor het waarnemen van afstanden zijn er drie soorten waarnemingen: absoluut (twee meter), relatief (twee keer zo groot), of ordinaal (groter). Voor het waarnemen van diepte zijn er verschillende *depth cues*. Vier ervan zijn bereikbaar met graphics:

- Occlusie is het verbergen van objecten die verder weg moeten staan;
- Perspectiefvertekening is het verbuigen of op andere manier transformeren van lijnen om een idee van perspectief te creëren;
- Atmosferische cues zorgen ervoor dat objecten die verder weg staan minder contrast hebben.
- Belichting en shaduwen kunnen ook zorgen voor een idee van vorm of afastand.

Drie ervan zijn bereikbaar met tracking:

- *Binocular disparity* is gebruikmaken van het verschil tussen wat het linker- en rechteroog ziet om afstand duidelijk te maken (objecten die dichtbij staan zullen voor de twee ogen verder weg van elkaar staan).
- Bewegingsparallax is het idee dat objecten die dichterbij staan sneller lijken te bewegen dan objecten die ver weg staan;
- Convergentie is het idee dat objecten die ver weg staan bij elkaar lijken te komen.

En *accomodatie* is het focussen op één bepaald deel van het plaatje, dat dan scherper wordt en dus dichterbij lijkt te staan.

Bij *stereographics* worden twee beelden gegenereerd: één voor het linker- en een voor het rechteroog. Er is een verkeerde manier om dit te doen: als beide camera's naar één punt gericht staan (dus niet parallel). Er zijn verschillende methodes om het juiste beeld in het juiste oog te krijgen:

- Bij *anaglyphic stereo* krijgt het linkerplaatje een rode tint en het rechterplaatje een blauwe tint; Er zijn meerdere manieren om dit te doen. Een voorbeeld is simpelweg door het linkerplaatje in de R-component en het rechterplaatje in de B-component van een leeg RGB-plaatje te plaatsen. Een andere manier is door links/R in R te zetten, links/G in G, en rechts/B in B.

Bij VR wordt vaak gebruik gemaakt van tracking om bijvoorbeeld het beeld te laten afhangen van de positie van het hoofd, of om interactie met objecten met je handen te simuleren.

Bij *augmented reality* is het idee om een device over een object te houden, en dat op het scherm dan relevante data bij dat object verschijnt.

### Graphics-hardware

GPU's bevatten tegenwoordig meer transistoren dan CPU's - deze worden gebruikt om berekeningen mee te doen. Om graphics real-time te genereren is hoge *throughput* nodig, om het ook interactief te maken is lage *latency* nodig. Er zijn meerdere optimalisaties gedaan in het verleden om graphics te versnellen:

- In hardware zijn aparte onderdelen gebouwd voor graphics;
- De 'graphics state machine' is een manier om niet meer bepaalde dingen per element te hoeven definiëren: er wordt simpelweg gekeken wat er op het moment in de state machine zit. Dit kan nuttig zijn om bijvoorbeeld veel vertices met dezelfde kleur te definiëren.
- Strips en fans worden gebruikt om minder punten nodig te hebben voor het definiëren van een geometrie: voor objecten die naast elkaar liggen hoef je minder punten te gebruiken.
- De *display list* is simpelweg een manier om dingen op de display bij compile-time te definiëren, voor statische geometrieën.
- Vertex arrays worden gebruikt om informatie over vertices sequentieel op te slaan aan de kant van de applicatie. Deze arrays kunnen dan ingeladen worden wanneer nodig.
- *Vertex buffer objects* lijken erg op vertex arrays, maar alle informatie wordt op de grafische kaart zelf opgeslagen. Dit zorgt voor snellere rendering.

De laatste optimalisatie is het uitbuiten van parallelisme. Ten eerste kan data-parallelisme uitgebuit worden (SIMD). Ten tweede zitten in de vertex pipeline weinig dependencies, dus kan ook d.m.v. multithreading instructieparallelisme worden uitgebuit.