

## Computer Graphics [met OpenGL]

Hier volgt een "samenvatting" in het masterwerk, soms terugkoppeland naar Bachelor-course, maar die samenvatting van mij was niet te volgen. Beter is de getypte tekst van Vincent Koeman volgen → bevat ook veel matrices.

### Naslagwerken

- [Hearn & Baker] Computer Graphics met OpenGL (bachelor / basics)
  - [Eisemann<sup>docent!</sup>] Real-time Shadows
  - [Akenine-Möller] Real-time Rendering
  - [Lengyel] Mathematics for 3D game prog. & CG
- } niet in mn bezit en niet gratis pdf. (master)

**Wist-je-dat?** In de 16<sup>e</sup> eeuw tekende Albrecht Dürer mbv. een raster. Hier keek doorheen of hij spande lijntjes richting het te tekenen voorwerp. In principe is dit ook hoe fotocamera's werken - alleen met lichtstralen door een lens.

**Wist-je-dat?** De virtuele camera alles modelleert mbv. driehoeken? Games van tegenwoordig gebruiken er > 200.000. Films zelfs > 1 Miljard, en daar moet dan over berekend worden; logisch dat bv. Despicable Me al €500.000 kostte aan elektriciteit

In deze course wordt alles bekeken vanuit het proces: de graphics pipeline. Onderdelen hiervan zijn:

- = projectie → het transformeren van de coördinaten naar het scherm
- = rasterizatie → van het model naar pixel representatie
- = "depth test" → te doen om in correctie volgorde te kunnen tekenen.

Een manier voor dit laatste is mbv. **Ray Tracing**. Vanaf het 'oogpunt' trekken een lijn (ray) en zoeken we het intersectiepunt dat het dichtstbij ligt. De gehele procedure is zeer kostbaar.

In een versimpelde vorm slaan we in elke pixel een diepte waarde op, welke geraadpleegd wordt wanneer we moeten bepalen wat te tekenen. Met deze approach wordt sorteren ontlopen. Wel kan deze approach niet omgaan met transparante voorwerpen.

Note: het vullen van de pixels kan zeer gemakkelijk geparalleliseerd worden

Meer over Visible-Surface Detection Methoden: **H&B g**

(kort: een **depth-buffer** bevat object afstanden vanaf de viewing position)



## → versimpelde graphics pipeline

Voordat de projectie en rasterisatie plaats vindt vinden alle camera- en object orientatie en bewegingen plaats. Voor transformaties wordt gebruik gemaakt van projectieve geometrie en homogeneous coördinates. Deze hom. coördinaten zijn noodzakelijk om een translatie en perspectief transformatie in matrix vorm te krijgen.

idee is: twee coördinaten zijn gelijk als er een  $a' = 0$  bestaat zo dat  $p = a = q$ .  
Op deze manier kan, wanneer je een plane vastlegt ook infinity vastgelegd worden.  
**Note:** "infinity is not altered by translation, but rotation does alter infinity."

° Ofwel, twee punten zijn gelijk als ze op een lijn door de oorsprong liggen. (één dimensie minder)

**Note:** Wanneer we bewegingen gaan combineren, is de volgorde van de operaties belangrijk. We werken in  $4 \times 4$  matrices voor 3d en matrix multiplicatie is niet commutatief, dus rotatie dan translatie  $\neq$  translatie dan rotatie

Als voorbeeld de rotatie om punt Q geeft de stappen ① transleer Q naar de oorsprong, ② roteer om de oorsprong, ③ transleer terug naar Q.

Transformaties worden behandeld in H&B5, met belangrijke matrices ook in de samenvatting.

Toevoegingen t.o.v. de bachelor is het concateneren van operaties om relaties te beschrijven; dit gaat mbv. een **matrix stack**. Let op dat je in dit geval van de wortel start (bv de schouder als je een complete arm "beweegt") en translatie voor rotatie gaat.

De volgende stap is projectie. Het complete camera model dat de image, projection én orientation/location bevat wordt vaak genoteerd als:

$$P = \begin{bmatrix} a_x & 0 & x_0 \\ 0 & a_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \end{bmatrix}$$

↗ de extrinsic/external parameters (= move camera)  
↘ intrinsic/internal parameters (= change settings)

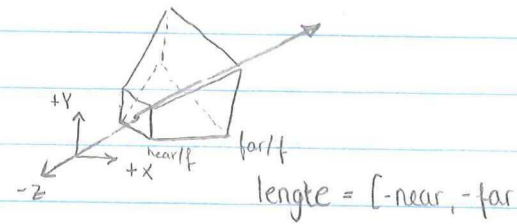
Let wel dat deze oplossing geen rekening houdt met diepte. Dit wordt opgelost door een near en far clipping plane toe te voegen (X-mapping)



screen mapping

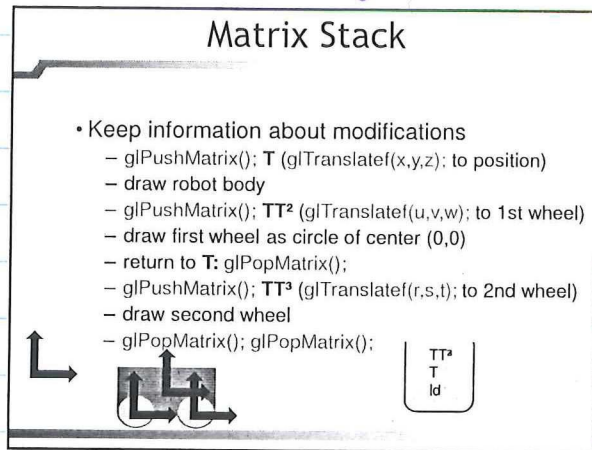
z-mapping

$$\begin{bmatrix} f/aspect & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{near+far}{near-far} & \frac{z \times near \times far}{near-far} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

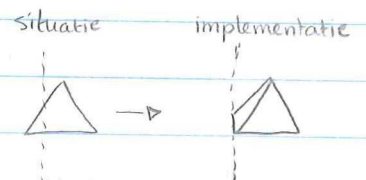


Hiermee kunnen we ook perspectie toevoegen: Er is meer precisie dichtbij en minder ver weg.

De uiteindelijke volgorde is model/view matrix  $\rightarrow$  projection matrix  $\rightarrow$  viewport matrix



Matrix stack voorbeeld



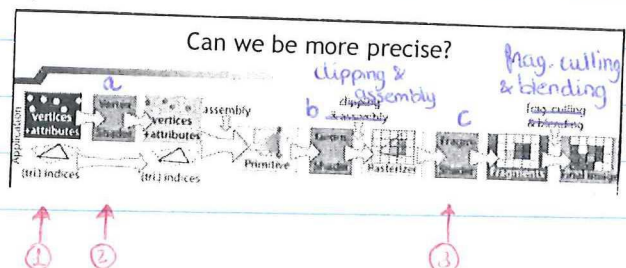
Dan een extra stap vóór rasterisatie is clipping, o.a. behandeld in H&B 6,7. De mastercourse gebruikt de approach om de objecten te testen tegen de cube van het perspective frustum; en noemt het creëren van "clipping triangles".

Vanaf OpenGL > 4.0 wordt er bewogen van de standaard pipeline vandaan: je moet zelf je matrices berekenen, de stack implementeren en doorgeven aan je shader - en shaders moet je zelf definiëren (altijd).

In de programmable graphics pipeline zijn 3 soorten shaders:

- vertex shader, verantwoordelijk voor camera projectie en object placement
- geometry shader, welke de array met vertices/attributen omzet naar meerdere primitieven (en o.a. clipping uitvoert)
- De Fragment Shader doet berekeningen over de pixels (en bepaalt dus de kleur).

Dus: tussen de geometry en fragment shader zit de rasterisatie stap!





Shading kan op drie momenten worden toegepast ① Flat shading gaat per oppervlakte, ② Gouraud per vertex (incl. interpolatie) en ③ Phong shading wordt gedaan per pixel (de nieuwe defacto standaard).

Shading en Illuminatie wordt o.a. behandeld in H&B 10-2, 10-10

Voor illuminatie hebben we de som van 3 termen:

$I$  = light property (RGB)  
 $K$  = surface property (RGB)

• Ambient  $A = I_a K_a$

emuleert het indirecte licht in een scene, zorgt ervoor dat niet alles pik zwart is.

• Diffusie  $D = I_d K_d \cos \theta$

Voor Lambert Surfaces geeft dit isotropische reflectie; Het geeft informatie over de objectvorm, want het resultaat hangt af van de lichtbron locatie, niet van die van de observer

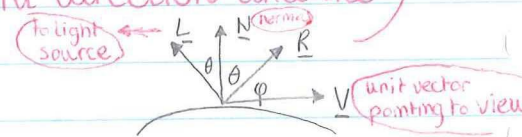
• Specular Phong  $\Rightarrow S(\varphi) = I_s K_s (\cos \varphi)^n$   
Blinn-Phong  $\Rightarrow S(\varphi) = I_s K_s \cdot \left( \frac{\vec{v} - \vec{r}}{\|\vec{v} - \vec{r}\|} \cdot \vec{n} \right)^n$

Wikipedia wil daar een +,  
maar het effect is duidelijker met -

Dit verkort de "lichtpunten" op voorwerpen.

$\theta$  = angle of ~~view~~ incidence between the incoming light direction and the surface normal

$\varphi$  = the angle <sup>viewing</sup> relative to the specular-reflection direction  $R$



Note: Cel-shading is putting a threshold on the diffuse shading

Na shading maken textures het allemaal af; het is zelfs zo dat het aantal driehoeken verminderd kan worden maar er meer detail bereikt wordt door textures\*. Textures zijn extreem bruikbaar en efficiënt, maar er zijn wel memory en some quality issues.

Textures worden o.a. gebruikt voor kleuren (ambient/diffuse), normals (mbv bump mapping) en precomputed illumination (mbv een light map), maar voor nog veel meer!

H&B 10-12

→ "Bump mapping is a technique for simulating bumps and wrinkles on the surface of an object; it is achieved by perturbing the surface moments of the object and using these perturbed normals during lighting calculations."

Note: Bump mapping does not actually modify the shape of the underlying object

\* Bv. stoeptegels in Markarth (Skyrim) is simpelweg textures! :)



Een texture kan gezien worden als een discreet scalar field op een oppervlak; uitdaging is het bepalen/vastleggen van de texture coördinates.

Een andere toepassing is environment maps; hier zijn de texture coördinates een weerspiegelende lijn en zit de environment map op een sphere om het object.

Een van de problemen van textures is Aliasing, in twee soorten:

1. Oversampling: een pixel is kleiner dan een texel (= texture element). Dit kan "tegengewerkt" worden mbv. bilineair interpolation. (opv. nearest neighbor)
2. Undersampling: Hier komt een pixel niet persee overeen met een texel

Voor textures zijn MipMaps de absolute standaard. Zowel de memory costs als performance costs zijn laag. Een mipmap is een hiërarchische texture waar op elke niveau de resolutie is gehalveerd. In deze structuur is een enkele ~~mipmap~~ lookup voldoende. Over de mipmap kan filtering uitgevoerd worden (bilineair of trilineair), waarbij het populairste anisotropische filtering is. "AF" is a method of enhancing the image quality of textures on surfaces that are at oblique viewing angles w.r.t. the camera where the projection of the texture appears to be non-orthogonal. It improves on BF and TF by reducing blur and preserving detail at extreme viewing angles." Tegenwoordig zijn anisotropische mipmaps niet uncommon.

Next step: Shadows!

Binnen shadowing heb je verschillende approaches. Hard shadows worden geconstrueerd met gebruik van 1 lichtpunt en we zien dat "point lights do not create penumbras." Soft shadows zijn resultaat van meerdere lichtpunten - of ~~het~~ het benaderen van de lichtbron als een verzameling lichtpunten.

Dese kunnen berekend worden mbv Ray Tracing, maar bij bv. 512 samples duurt deze berekening minuten! ↳ H&B 10-11

Oplossing is Shadow Mapping, waar gebruik gemaakt wordt van de depth buffer (al eerder genoemd), hier een shadow map genoemd.

Een tweede stap in dit process is het renderen vanaf het viewpoint (geïmplementeerd in een fragment shader). Voor elke pixel wordt 'in afstand vergeleken met de shadow map → equal = pixel is lit / farther = in shadow. Problemen/uitdagingen hier zijn discretizatie en een depth bias, en staircase artifacts en oversampling.


↳ major focus: bv met SM partitioning (increase resolution where needed)



**Note:** Depth is niet uniform! i.a.w. de depth values in de depth buffer are not uniformly ~~partitioned~~ partitioned! Kijk maar naar de z-mapping in ④.

Terug naar Shadow Mapping uitdagingen / limitaties, deze zijn naast partitioning op te lossen met:

- SM "warping" [good voor "outdoor" scenario's]  
zie paper "light space perspective shadow maps" (LSPSM)
- **z-partitioning**. Deze methode gebruikt meerdere shadow maps, waar warping er maar één gebruikt.  
→ een andere naam is cascaded shadow maps. ← er zit een sectie in het boek van Eisemann hierover!

Een andere richting van oplossingen (hierboven valt onder improved sampling) is improved reconstruction / filtering. Je zou bijvoorbeeld een hard shadow kunnen filteren met een **percentage-closer filtering** (kortweg een gemiddelde van meerdere pixels ) → woow, een methode uit 1987! (Reeves et al.)

Deze methode is simple te implementeren en verbergt discretizatie, maar is wel kostbaar in de berekeningen. → heb ik!

Vervolgens behandelt de docent **Variance Shadow Maps** [Lauritzen / Donnelly], waarvan ik maar de paper moet lezen; het lost iig shadow aliasing op!

Echter, deze approach levert soms nog 'light leaks' op.

Andere approaches zijn: **Convolution Shadow Maps** [Annen et al.] → heb ik!

heb ik ook! **Exponential Shadow Maps** [Annen et al. / Salvi]

**Note:** Gemiddelden berekenen op de GPU kan op verschillende manieren (Gow)  
① Kipmapping (Williams) ② N-buffers (Décorêt) en ③ Fast Summed Area Tables

Bovenstaande omschreven verhaal was voornamelijk van toepassing op hard shadows! Bij soft shadows zijn er o.a. problemen als de occluder fusion en single silhouette. paper heb ik!

ik! **Eisemann & Décorêt** presenteren een methode voor correcte schaduwen bij meerdere voorwerpen. Dit algoritme bestaat uit 4 stappen: ① Vind de influence regio per triangle; ② Backproject deze triangle; ③ vind uit wat de geblokkeerde samples zijn; ④ accumulate over alle triangles.

De approach wordt **View-sample Mapping** genoemd. → offline?



Interessant hier is dat ④ en ② geïmplementeerd worden in de vertex/geometrie shader, ③ in de fragment shader en ④ in de blending stap zit.

**Note:** "backprojection can be interpolated over influence region on a plane."

Het volgende onderwerp is global illumination

Point based global illumination is de standaard in de industrie. **Wist-je-dat** Disney niet aan Ray Tracing doet?

Voor off-line illuminatie wordt de rendering equation, de BRDF (bidirectional reflectance distribution function) en radiosity behandeld.

Bij radiosity wordt aangenomen dat alle oppervlakken diffuus zijn (waardoor betekent dat de zelfde energie naar alle richting verspreid). Radiosity houdt dus rekening met indirect licht (wat raytracing niet doet!). Dit levert een realistischer resultaat. Bij de radiosity berekening hoort het berekenen van form factors, welke een 'bounce' van het licht omschrijft.

→ ik heb een leuk artikel gevonden ☺ uit "the Visual Computer."

$$L(x, \theta_o, \phi_o) = L_e(x, \theta_o, \phi_o) + \int_{\Omega} \underbrace{\rho_{brd}(x, \theta_o, \phi_o, \theta, \phi)}_{=0 \text{ indien geen light source}} \underbrace{L_i(x, \theta, \phi)}_{\text{reflectance}^{\circ}} \underbrace{\cos \theta}_{\text{weerkaats + licht}}$$

**kortgezegd:**

De **lambertian** ideal diffuse caster verspreid zo dat het onafhankelijk van richtingen; dus is de reflected radiance identiek in alle richtingen. Ofwel, dit is het Phong model zonder specular term.

**Note:** In feite is de integraal een lineaire operator!

→ in principe is radiosity het oplossen van een groter lineair systeem ☺

Voor real-time applicaties moeten we sneller zijn. We zullen de depth buffer gaan hergebruiken, die hadden we toch al nodig!

Besproken is (screen-space) **ambient occlusion** en **voxels**. volume + pixel ☺

**Wist-je-dat:** Uncharted 2 SSAO gebruikt ☺

nog een beetje onderontwikkeld.  
heb ik een paper over!  
van Eisemann zelf ja.

Ambient occlusion can be defined as the soft shadow generated by a sphere of light of uniform intensity surrounding the scene. (outdoors = sky light)

(Ritschel).  
Voor real-time wordt ook gebruik gemaakt van "imperfect" shadow maps