C# Programming Techniques
Advanced Applied Programming
2017 -2018

Katja Verbeeck
Joris Maervoet
Tim Vermeulen

3D Graphics

Inhoud

3D Toepassingen

3D Graphics Pipeline

Geometrische bewerkingen

Rastering - Rendering

Belichting

Wiskunde transformaties

TOEPASSINGEN van 3D

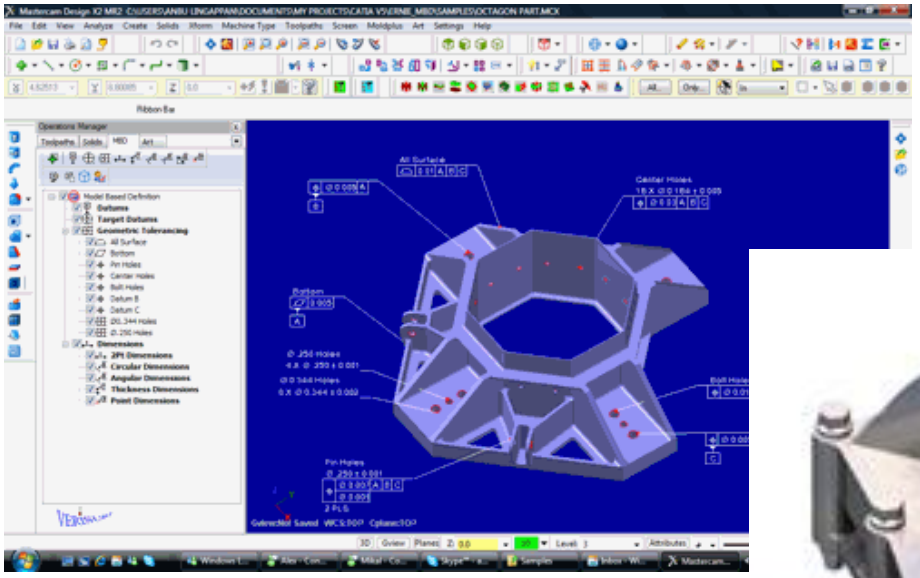
animatiefilms

1986: eerste academy award voor Pixar Animation Studios

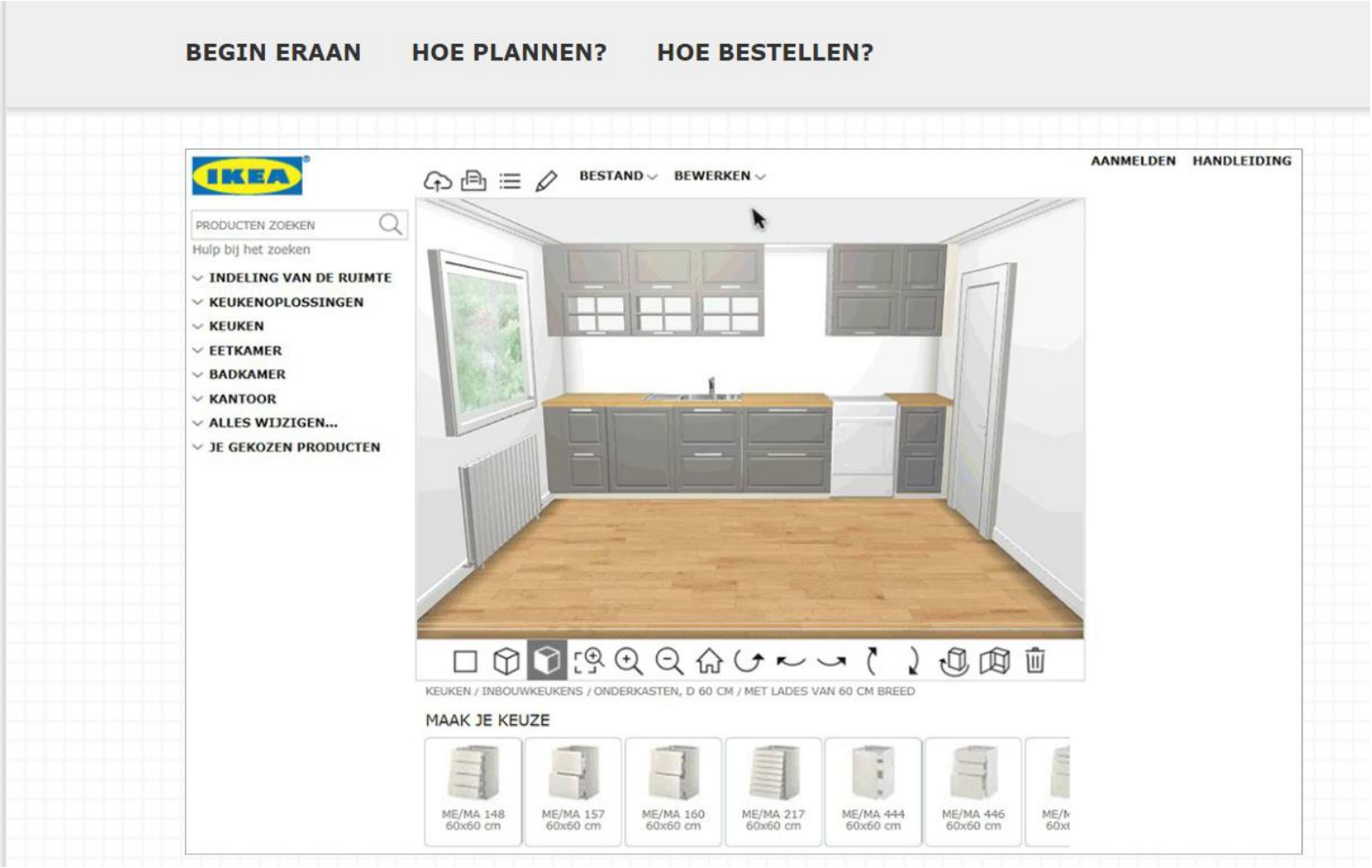
Luxo, Jr. is de eerste computeranimatiefilm gemaakt door Pixar Animation Studios, sinds de oprichting als onafhankelijke filmstudio. Het filmpje van 2 minuten en 18 seconden werd in 1986 gemaakt, en diende als demonstratie van waar Pixar toe in staat was.

When John Lasseter was learning how to make models, he chose the nearest, easiest subject: an architect's lamp sitting on his desk. He started moving it around in the animation system like it was alive and it eventually became another short film by Pixar that was nominated for an Academy Award®.

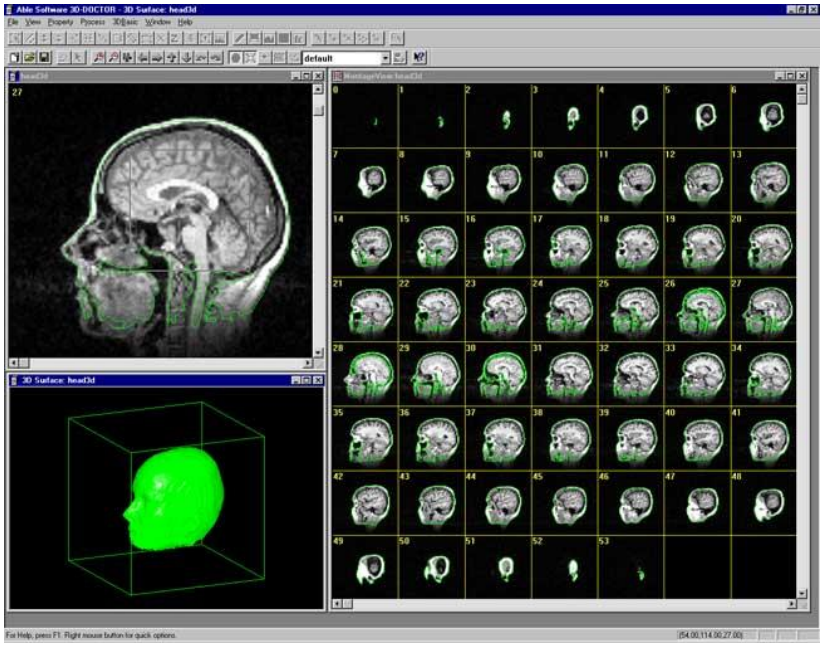
Simulatie / Prototyping

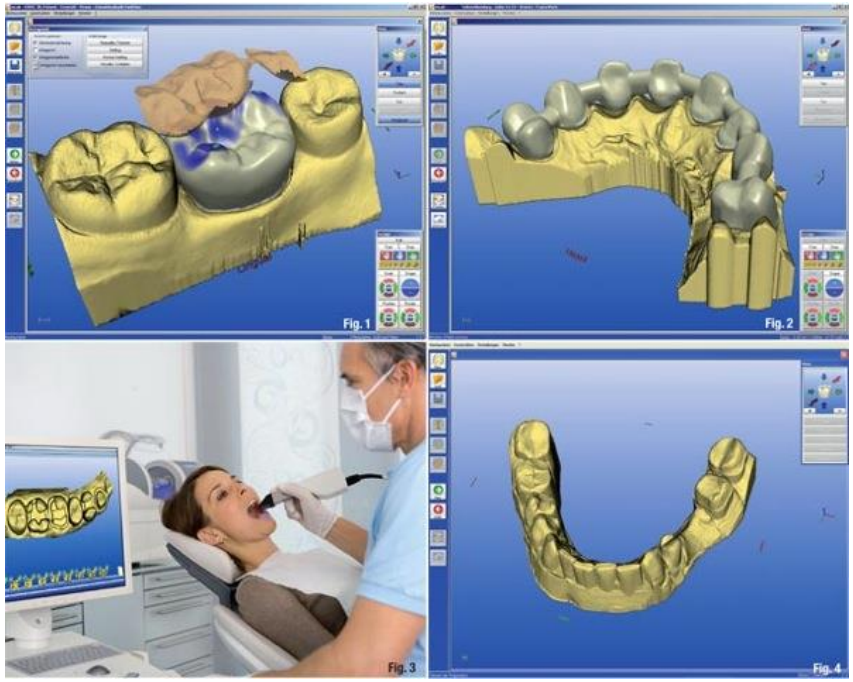


3D ontwerptools



Medische Beeldvorming



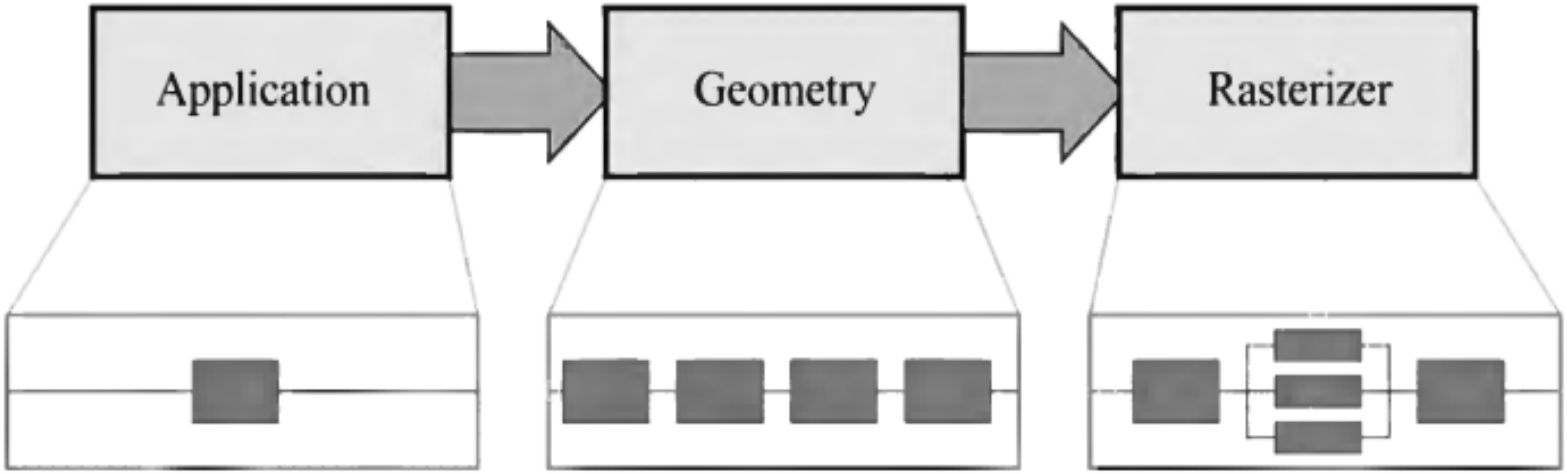


Game Industry

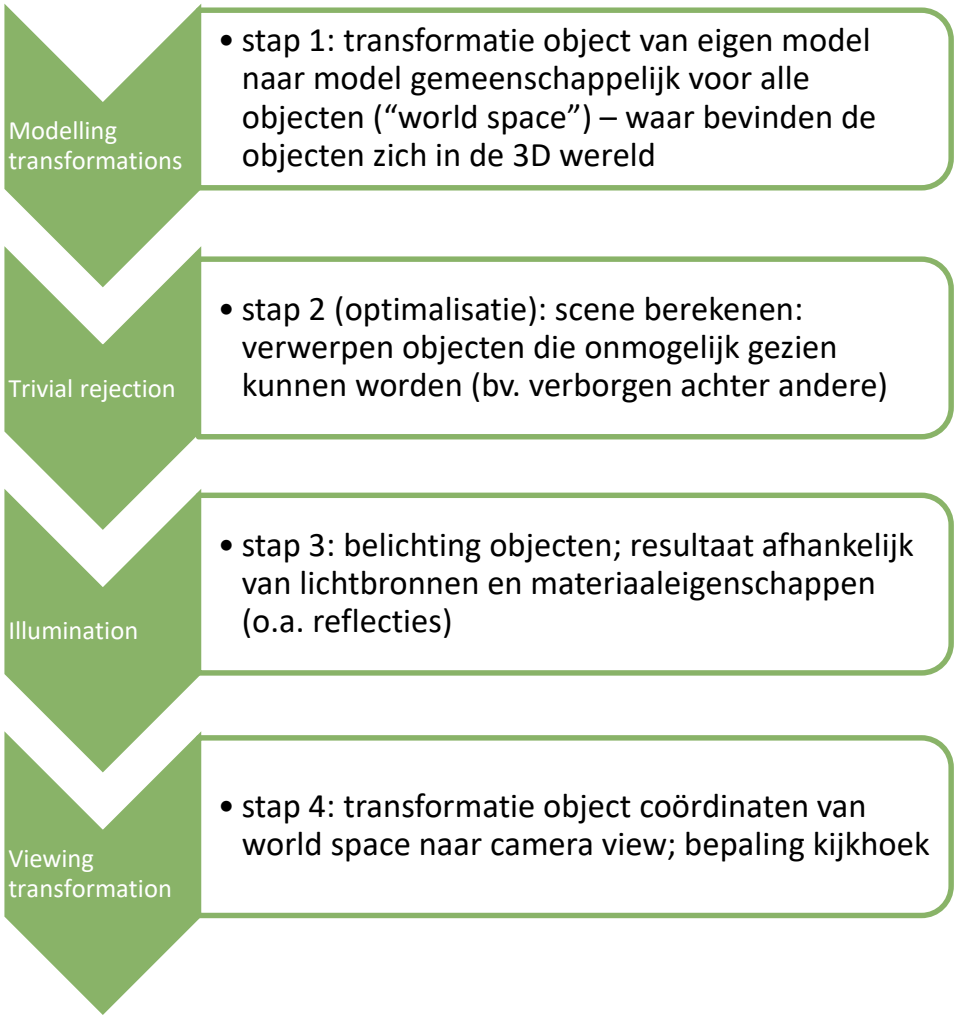
VR-bril

3D Graphics Pipeline

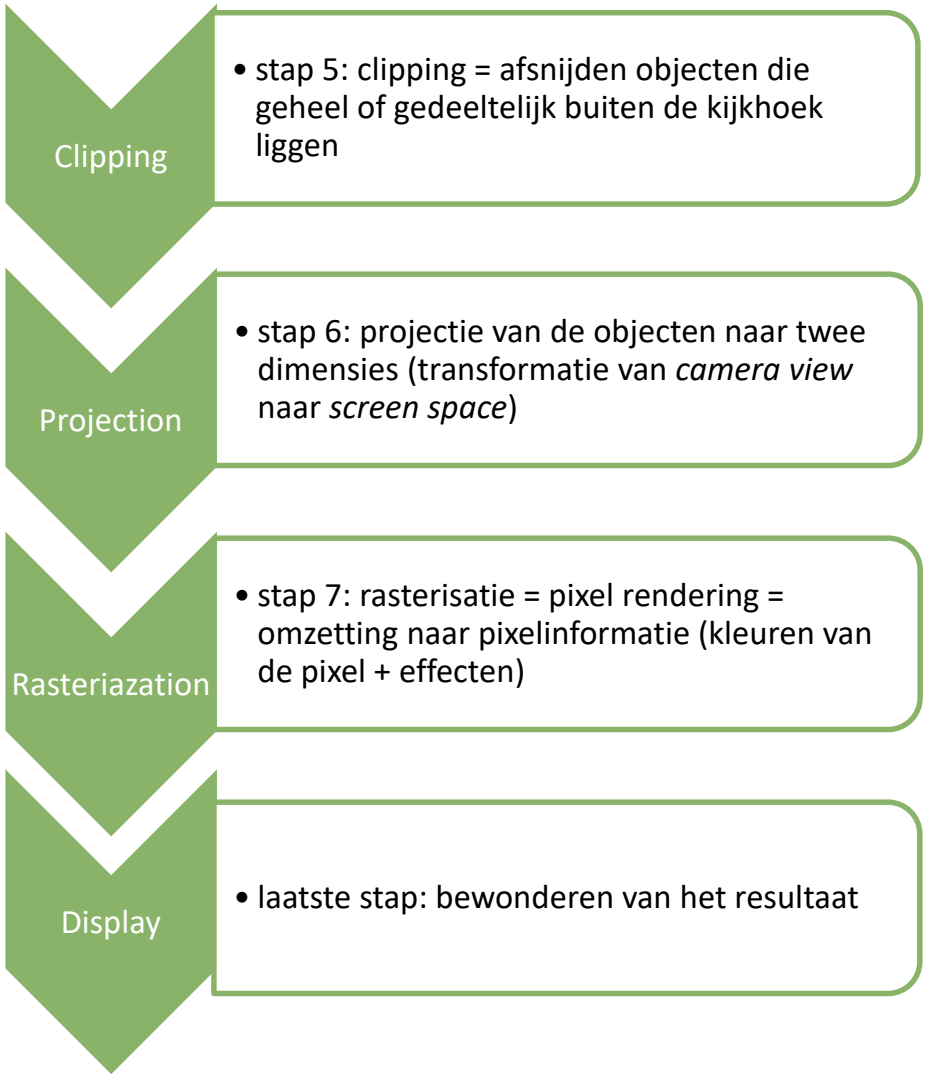
3D-graphics genereert tweedimensionale afbeeldingen uit driedimensionale representaties van geometrische objecten.



Geometrische bewerkingen



Geometrische bewerkingen



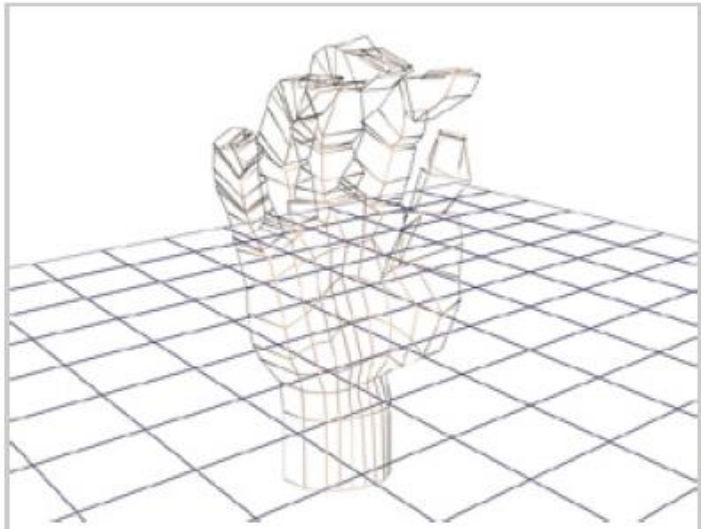
Geometrische / 3D begrippen
Vertices

complexe objecten worden opgesplitst in kleine veelhoeken of polygonen (meestal driehoeken)

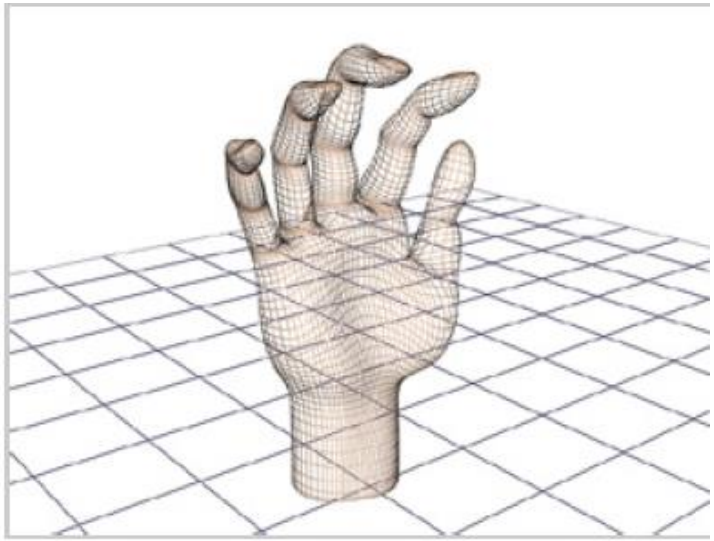
Opgeslagen via de coördinaten van de hoekpunten

Benadering van reëel object

Wiskundig eenvoudiger en dus minder rekenintensief



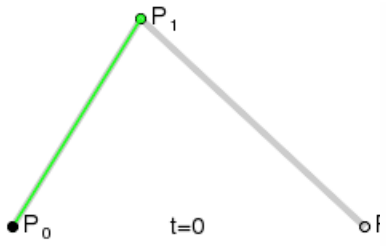
This illustration shows the wireframe of a hand made from relatively few polygons -- 862 total.
How StuffWorks.com

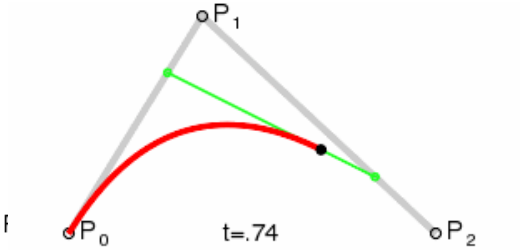


The outline of the wireframe can be made to look more natural and rounded, but many more polygons -- 3,444 -- are required.
How StuffWorks.com

3D theepot, “hello World” voor de grafische community

Utah teapot of **Newell teapot**





Quadratische Bezier curve

De 3D theepot is het eerste object geweest dat niet werd opgebouwd via een set van polygonen. Hierna is de theepot lange tijd als benchmark (snelheidstest) voor renderingsoftware gebruikt. De originele theepot is nog te bezichtigen in het Computermuseum in Boston.

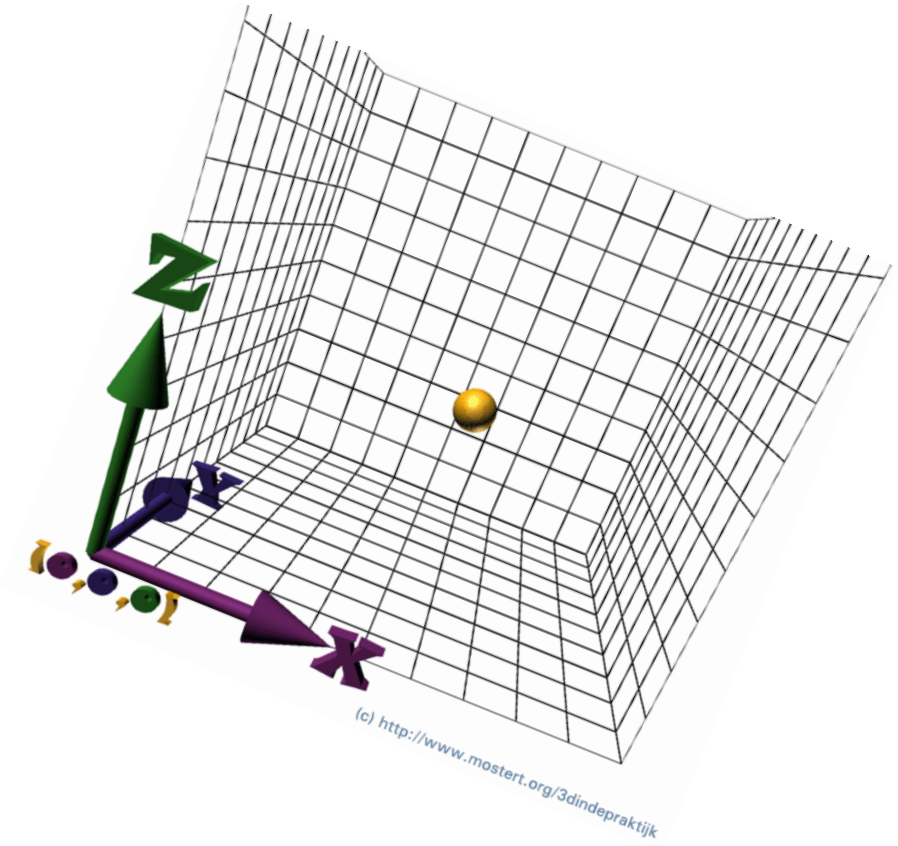
De Camera of Frustum

Het cameramodel veronderstelt dat een camera op de 3D scene gericht wordt – wat de camera ziet wordt op het scherm getoond – zowel de wereld als de camera kan gemanipuleerd worden

Het **frustum** is het gezichtsveld van een camera in 3D computer graphics.

OBJECT Space vs World Space

- Elk 3D object heeft eigen coördinatenstelsel (LCS)
- Een 3D scene heeft eigen coördinatenstelsel (WCS)
- Eerste stap in rendering pipeline = alle individuele coördinatenstelsels "mappen" op de world space via transformatie van de coördinaten van de vertices.

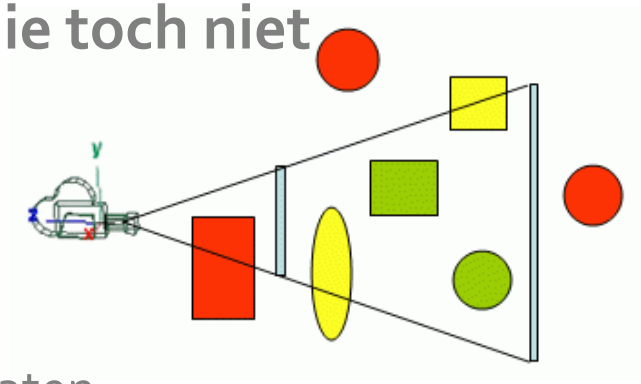


Van World space naar Cam space

- Positie in 3D scene gezien door de "camera"
- Voor de berekening van het uiteindelijke gerenderde beeld is dus ook een transformatie van world space naar camera space nodig
- Tot slot ook projectie naar 2D pixel informatie

Culling

Culling = het weglaten van veelhoeken die toch niet zichtbaar zullen zijn.

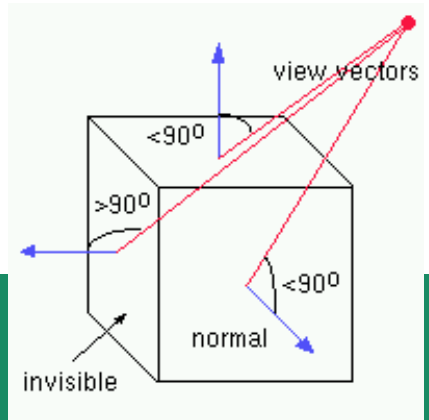


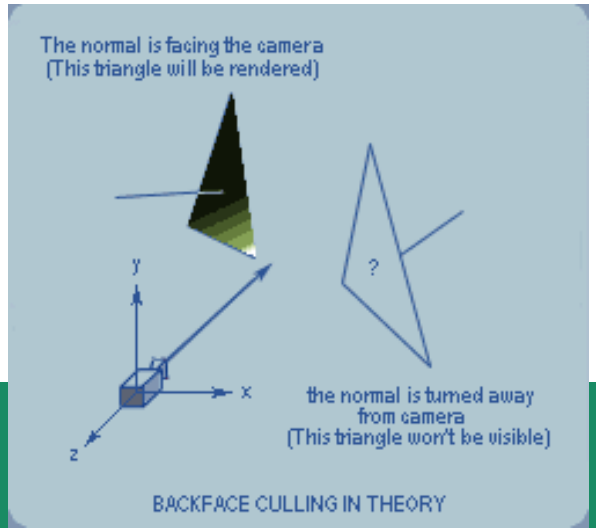
'Frustum culling':

Veelhoeken die buiten het frustum liggen weglaten.

'Backface' culling':

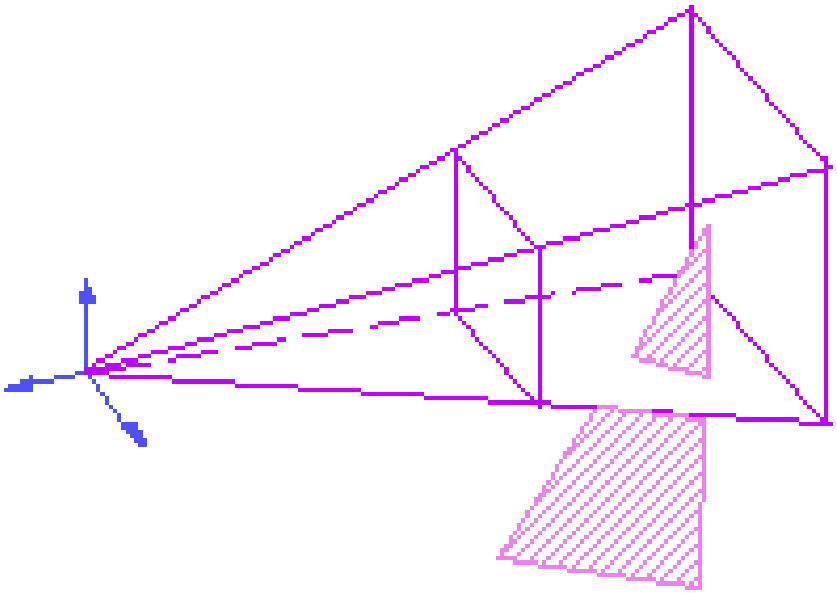
Elke veelhoek heeft een 'voorkant' en een 'achterkant'. Als alle objecten 'gesloten' zijn, zullen de veelhoeken die met hun achterkant naar de camera gericht zijn nooit zichtbaar kunnen zijn.

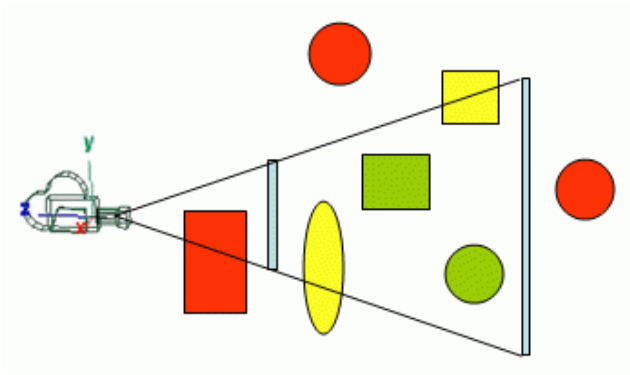




Clipping

Veelhoeken die de randen van het frustum snijden: het zichtbare deel omvormen naar nieuwe veelhoeken die volledig binnen frustum liggen.





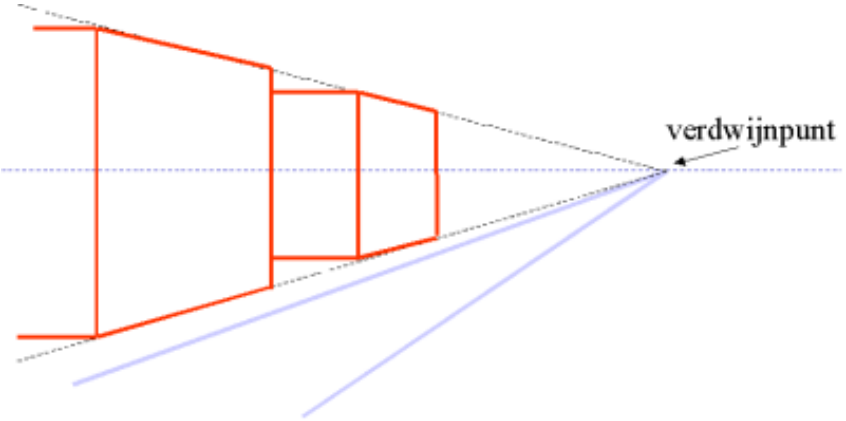
Van 3D naar 2D: projecties

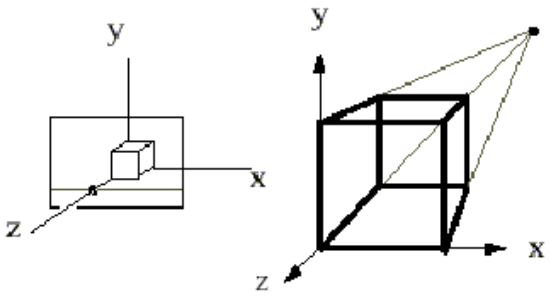
Uiteindelijk wordt je 3D wereld weergegeven op een 2D medium: op het scherm of afgedrukt op een blad papier bijvoorbeeld

Er is dus een omzetting nodig van 3D naar 2D

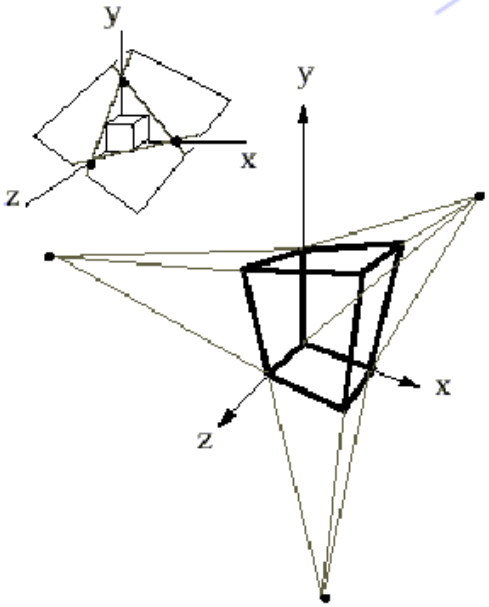
Verschillende methodes ter beschikking

Lijnperspectief

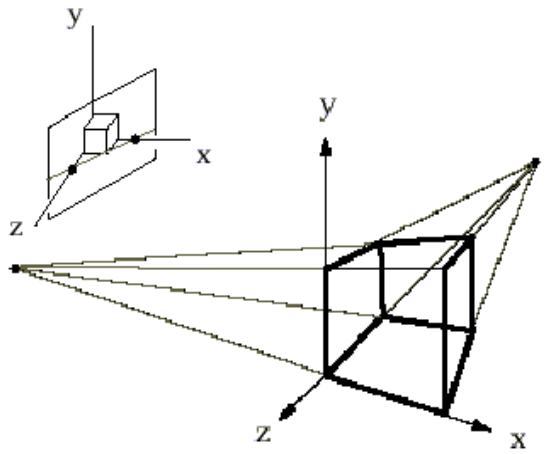




éénpunts-perspectief
(z-as vluchtpunt)



driepunts-perspectief
(x, y en z-as vluchtpunten)

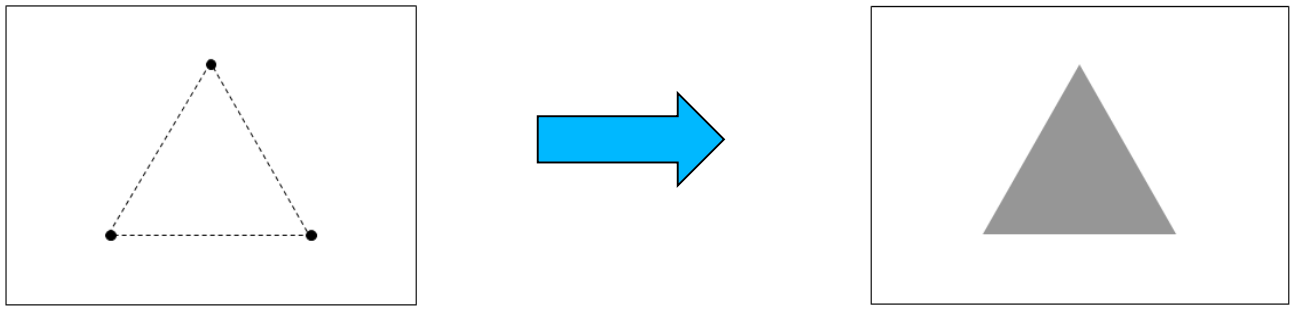


tweepunts-perspectief
(z en x-as vluchtpunten)

Lijnperspectief

Rastering

Op basis van de gebruikte projectie, zal nu voor elke polygoon een aantal pixelfragmenten berekend worden



Van coördinaten van hoekpunten --> pixelfragmenten

Rendering

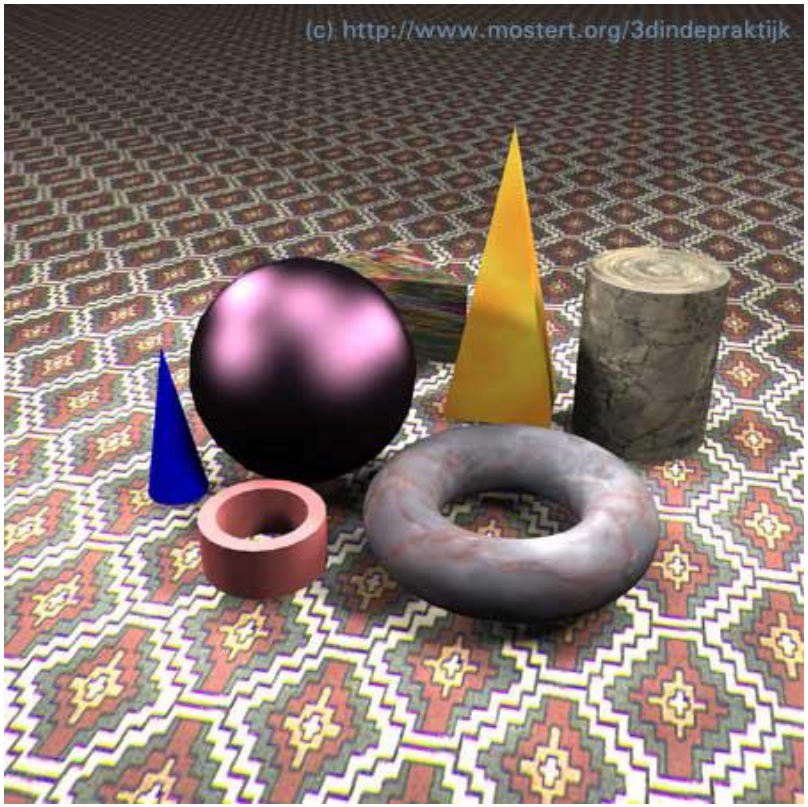
render-engine : het basisalgoritme waarmee de driedimensionale wereld op een tweedimensionaal scherm geprojecteerd wordt. De scène wordt berekend en vervolgens zeer gedetailleerd opgebouwd. Hierbij wordt bijvoorbeeld precies berekend hoe het licht op de objecten valt en welke specifieke eigenschappen het oppervlak van ieder object heeft.

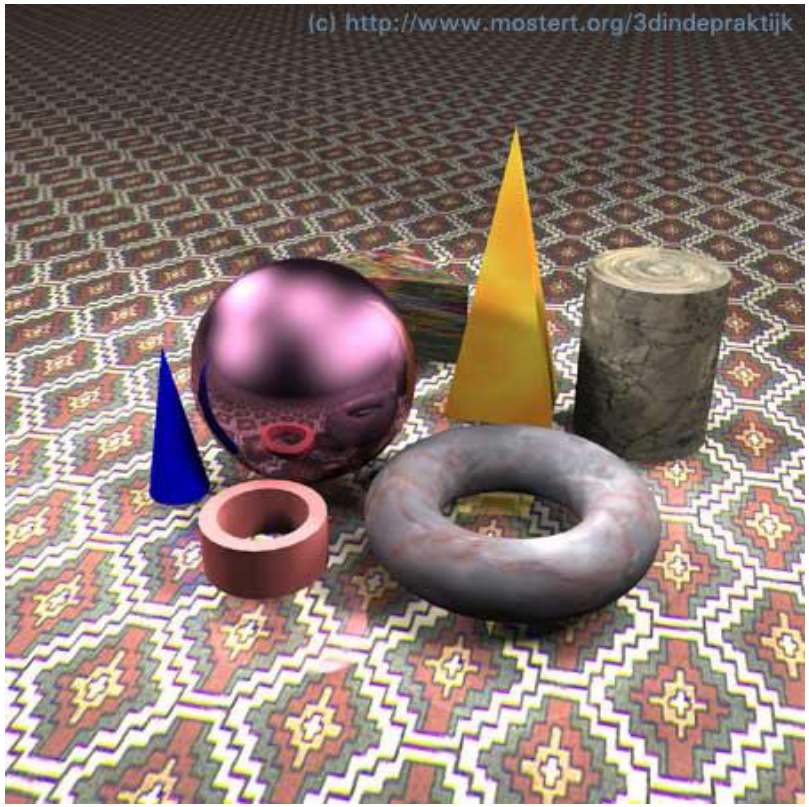
2 type technieken:

- Afbeelding-gebaseerde : voor elke pixel van de afbeelding wordt bepaald welke kleur deze heeft. Tijdsintensief, maar kwaliteitsvol

  (oa. ray-tracing)

- Object-gebaseerde: in plaats van elke pixel langs te gaan, wordt elk object langsgegaan en geprojecteerd op de afbeelding. Het is hierbij gebruikelijk om alle geometrie die afgebeeld moet worden te vertalen naar driehoeken, die vervolgens door de rasterizer op het scherm getekend worden. (dit is het geval voor huidige computergames)





Scanlijn-tracing

Ray-tracing

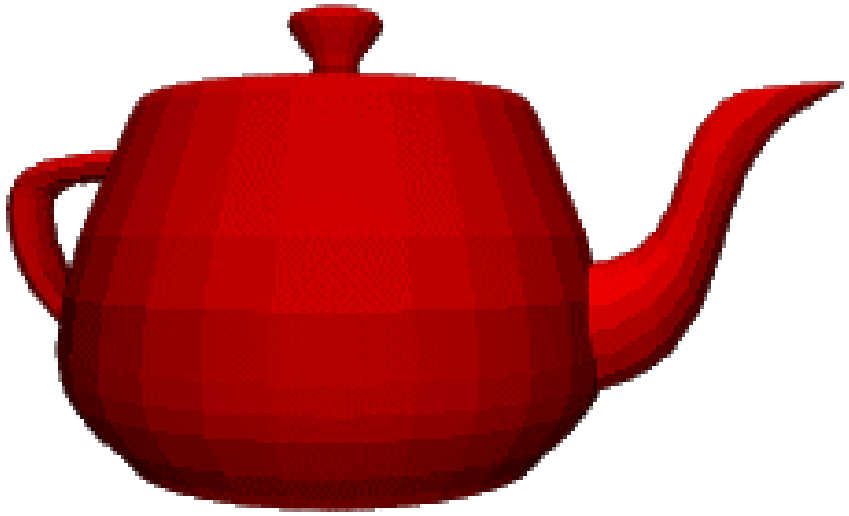
Bij scanline rendering bekijkt de render-engine de scène lijn voor lijn. De engine bepaalt welke punten zichtbaar zijn en welke niet. Vervolgens worden voor de zichtbare punten de juiste eigenschappen bepaald zoals bijvoorbeeld de uitvoering van het oppervlak. Bij ray-tracing wordt de scène pixel (punt) voor pixel bekeken en vervolgens opgebouwd. Ray-tracing zorgt dan ook voor een veel gedetailleerdere en realistischere weergave van uw scène.

Shading-algoritmes

- Bij rendering wordt vaak de term *shading* gebruikt. Omdat een virtueel object zo realistisch mogelijk moet worden, tracht de software via de polygonen van een object, dezelfde kleur, warmte en texture te krijgen als een object uit de werkelijke wereld. Dit aspect van rendering wordt aangeduid met shading.
- Shading is berekenen van kleurverandering op een object onder invloed van lichten
- Steeds afweging nodig tussen nauwkeurigheid en rekenintensiviteit

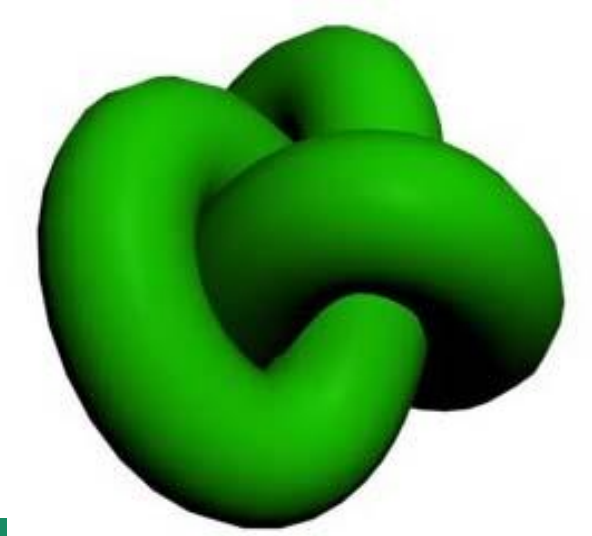
Flat shading of constant shading

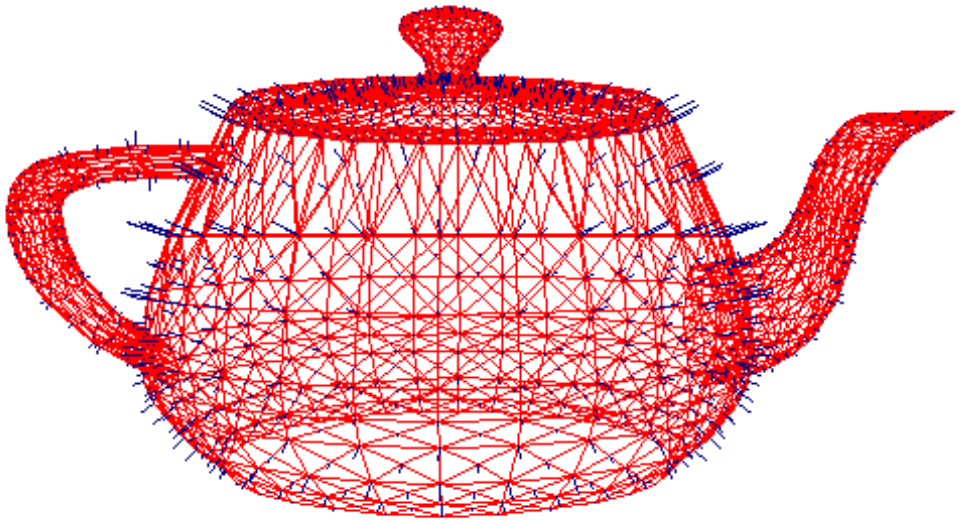
- Elk polygoon krijgt één kleur
- Zeer snel, ideaal voor previews



Gourad shading

- Kleur wordt berekend in elk hoekpunt langs de normaal
- Kleuren in driehoeken via interpolatie





Phong shading

- Hier worden eerst normalen geïnterpoleerd en dan kleur berekend langs geïnterpoleerde normalen
- Veel rekenintensiever!

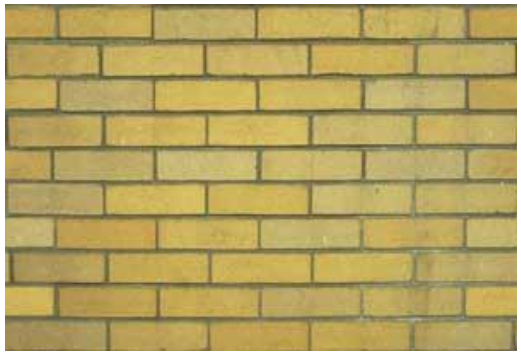
Andere

- Bvb metal shading en ink shading

Rasterbewerkingen: Textures

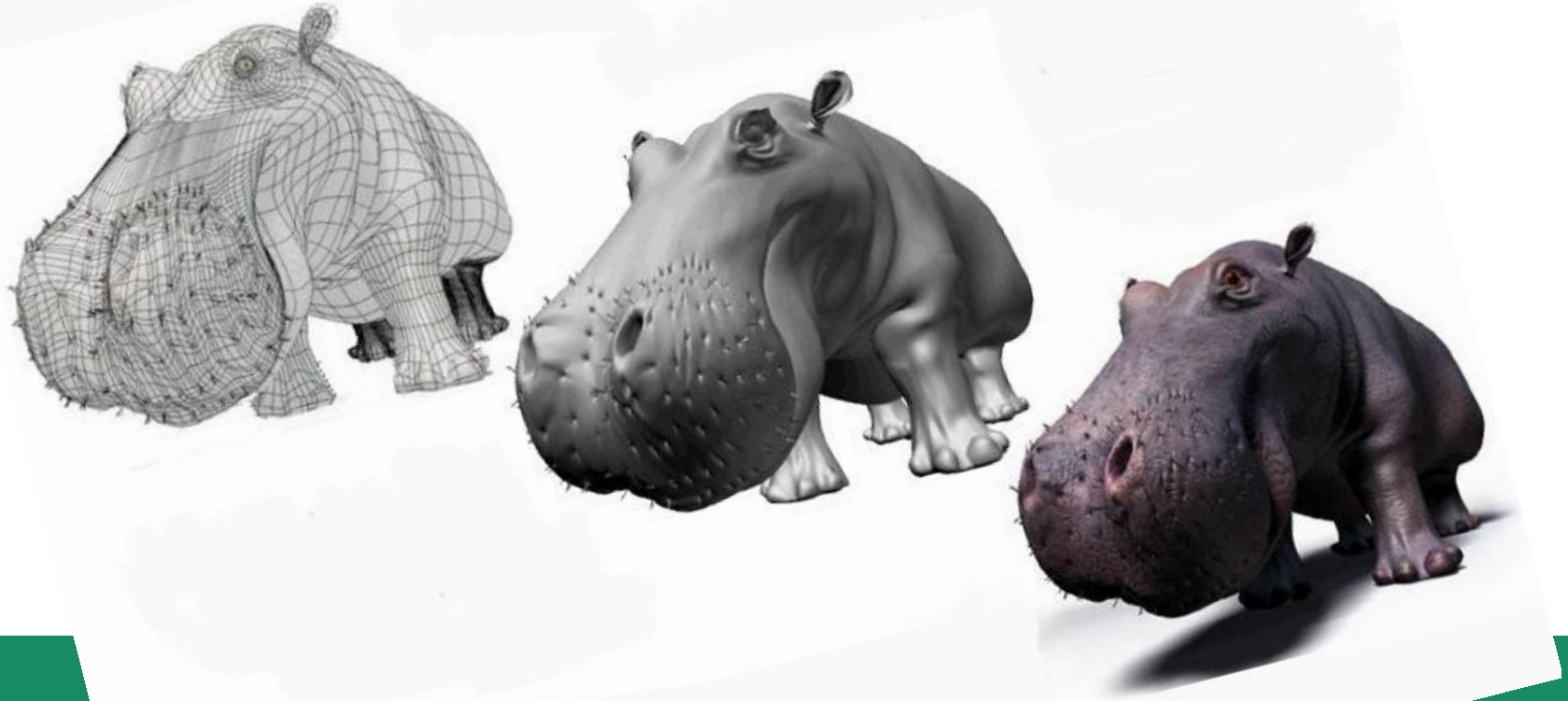
om een object een realistisch uiterlijk te geven maakt men gebruik van een *texture* (ook wel *maps* of *texture maps* genoemd)

Een texture is een 2D-afbeelding van bijvoorbeeld een bepaald materiaal, dat op het oppervlak van een object wordt geplakt.



Waarom texture mapping?

- Shading en reflection alleen volstaan niet voor realistisch beeld:



Texture mapping: het principe

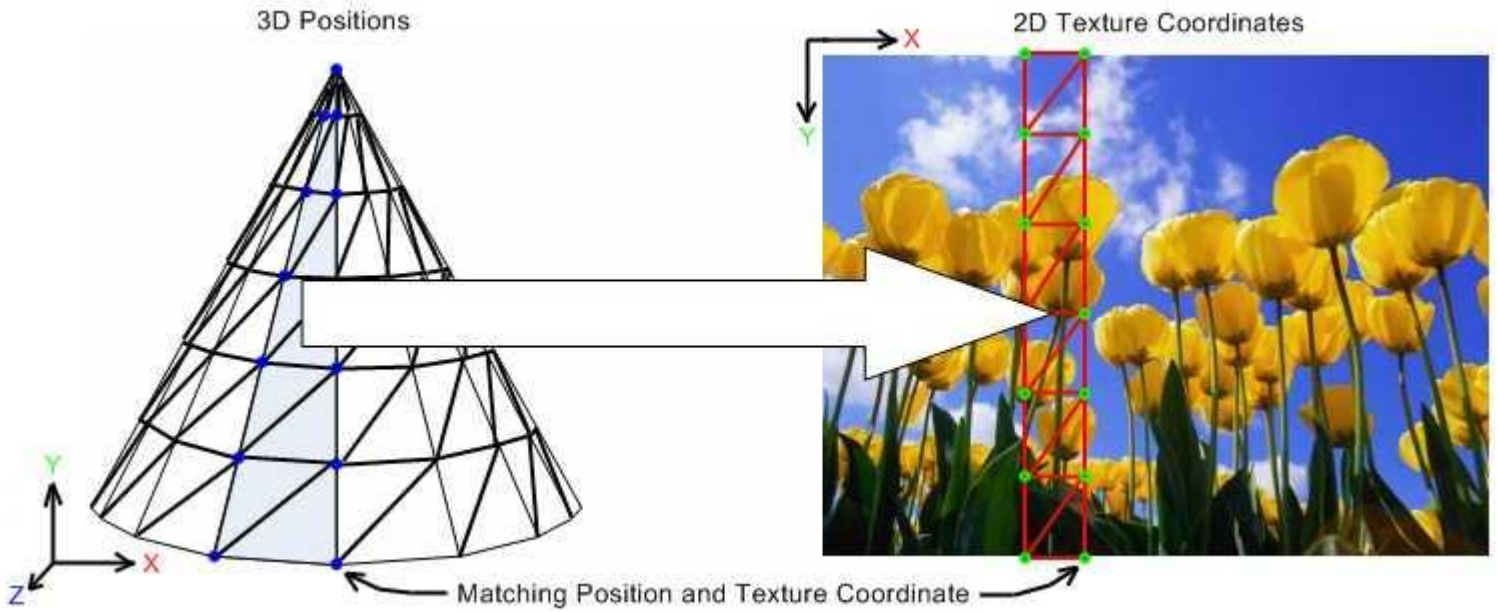
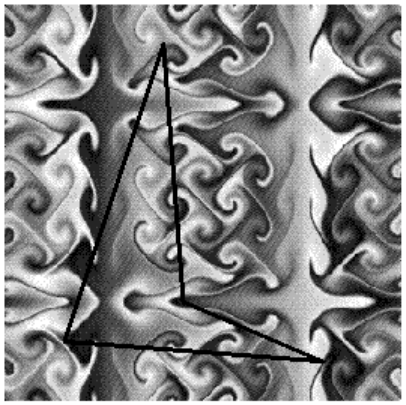


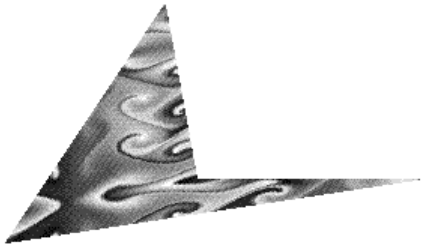
Figure from https://blogs.msdn.microsoft.com/danlehen/2005/11/06/3d-for-the-rest-of-us-texture-coordinates/

Een bitmap (foto,…) 'plakken' op elke polygoon

```
<MeshGeometry3D
   Positions="-1,1,0 -1,-1,0 1,-1,0"
   TextureCoordinates="0,0 0,1 1,1"
   TriangleIndices="0,1,2" />
```

Textuurfiltering





Probleem: originele bitmap heeft verschillende resolutie dan uiteindelijk beeld van de polygoon. ->

Alle pixels van het 3D object moeten een kleur krijgen afgeleid uit de pixels (texels - texture elements) van de texture

Simpele oplossing: point sampling
pixel krijgt de kleur van dichtstbijzijnde texel

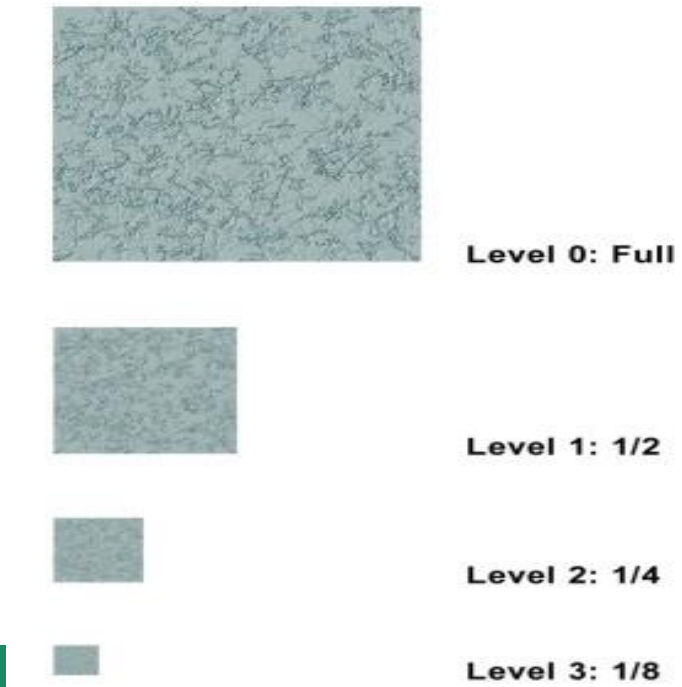
! Undersampling: resolutie textuur > resolutie polygoon
--> niet alle textuurpixels gebruikt, verlies informatie)

! Oversampling: resolutie textuur < resolutie polygoon
--> meerdere pixels gebruiken een texel: blokkerig)

voorbeeld: Doom1/2, dicht bij muur

Textuurfiltering: MIP-mapping

MIP: textuur in verschillende resoluties opslaan, best geschikte resolutie gebruiken.



Belichting

Licht geeft meer realistisch leven aan je 3D scene

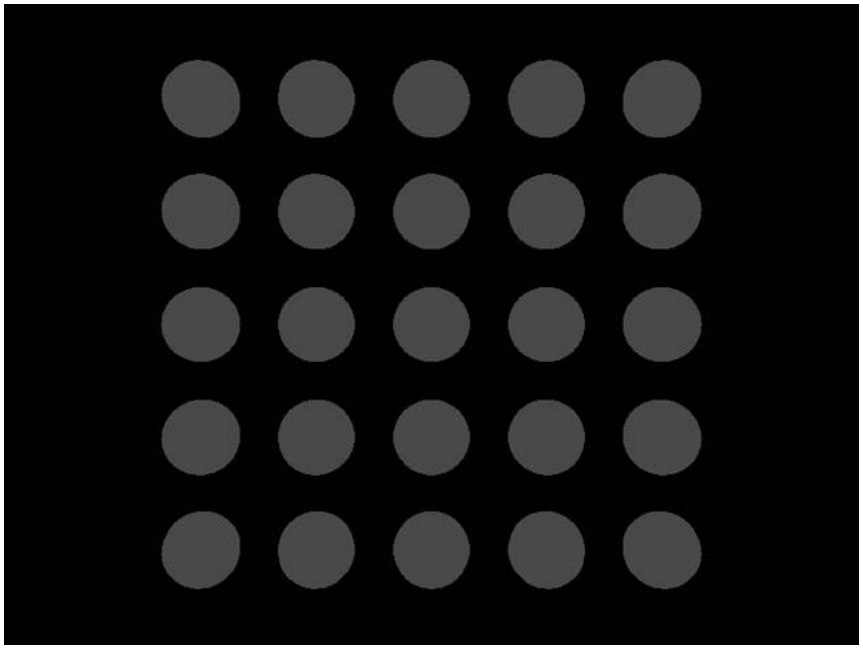
Licht betekent ook schaduw!

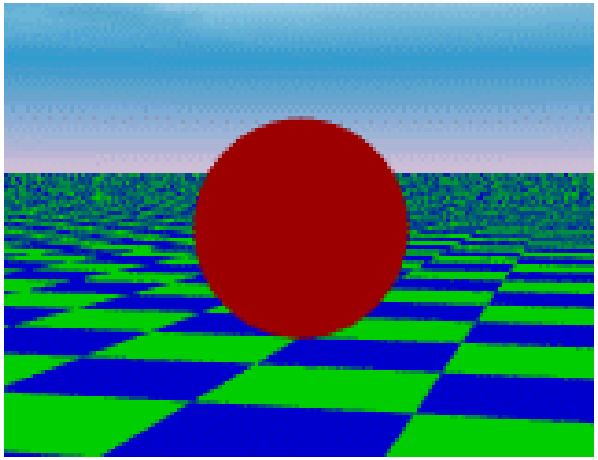
Verschillende lichtbronnen en parameters tunen : (lichtkleur, helderheid,schaduwmethoden, special effects, …)

Ambient light

Lichtbron zonder oorsprong of richting

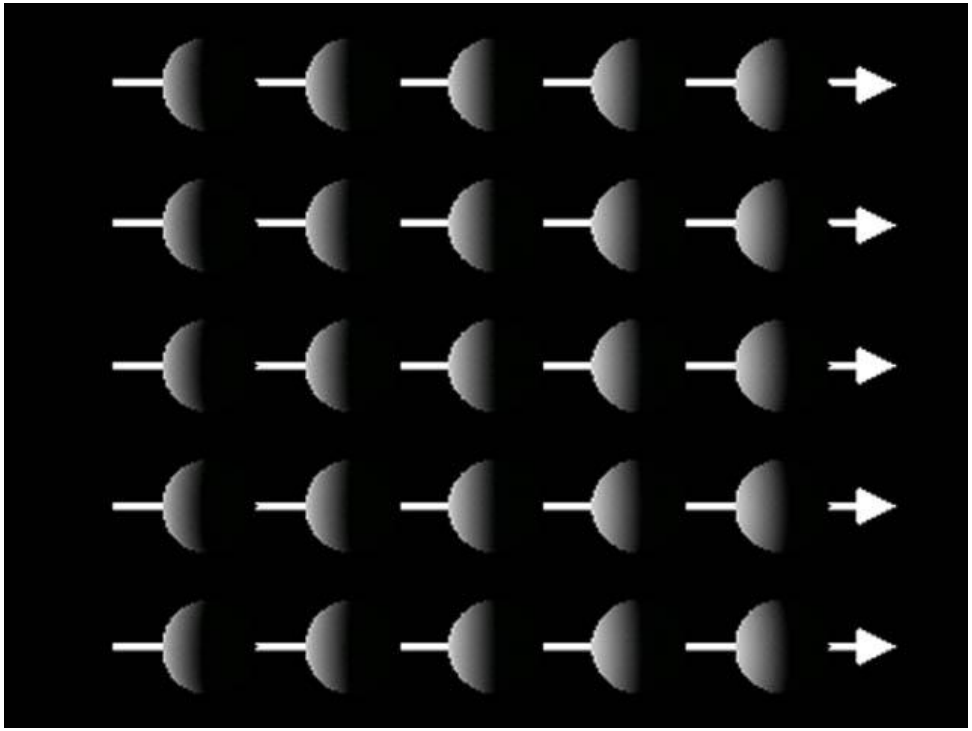
Het voordeel van een ambient light is dat je hele scène meteen verlicht is, het nadeel is echter dat er geen schaduwwerking optreedt waardoor alles een vrij steriele en "platte" indruk geeft.

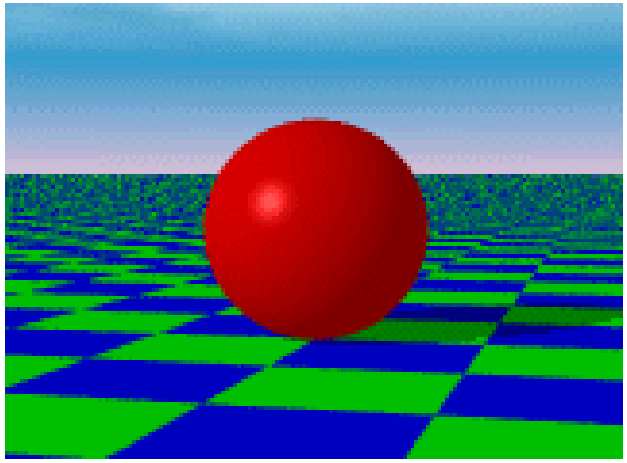




Directional light

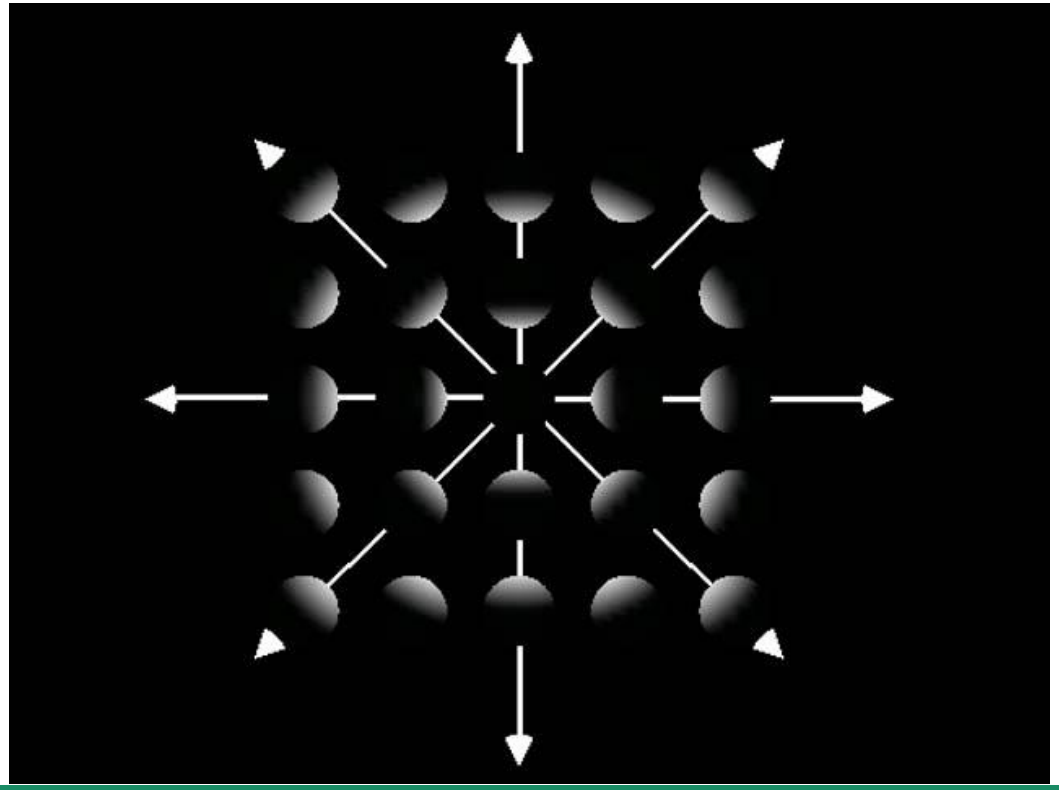
Lichtbron op oneindig bvb de zon, de lichtstralen lopen evenwijdig





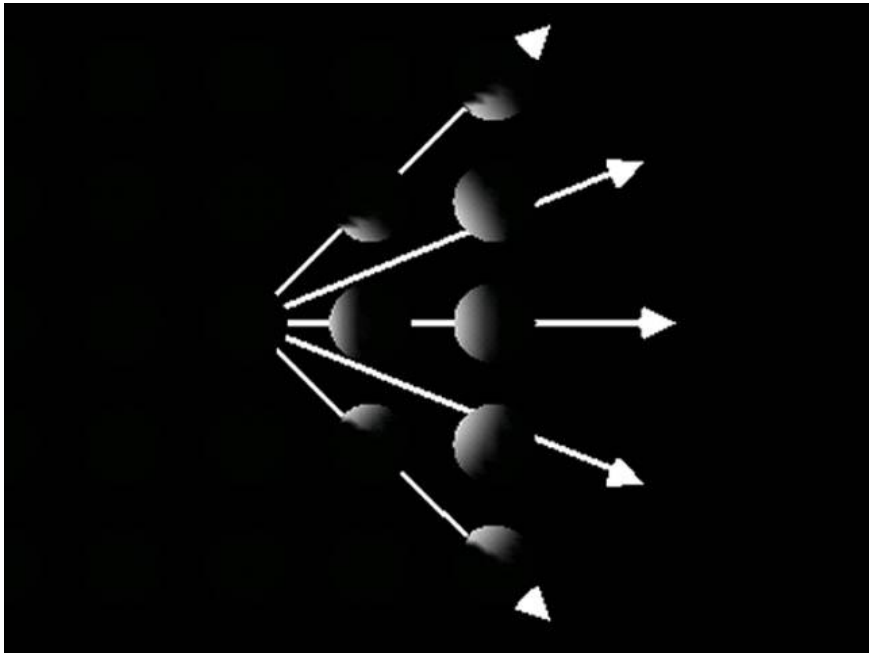
Point light

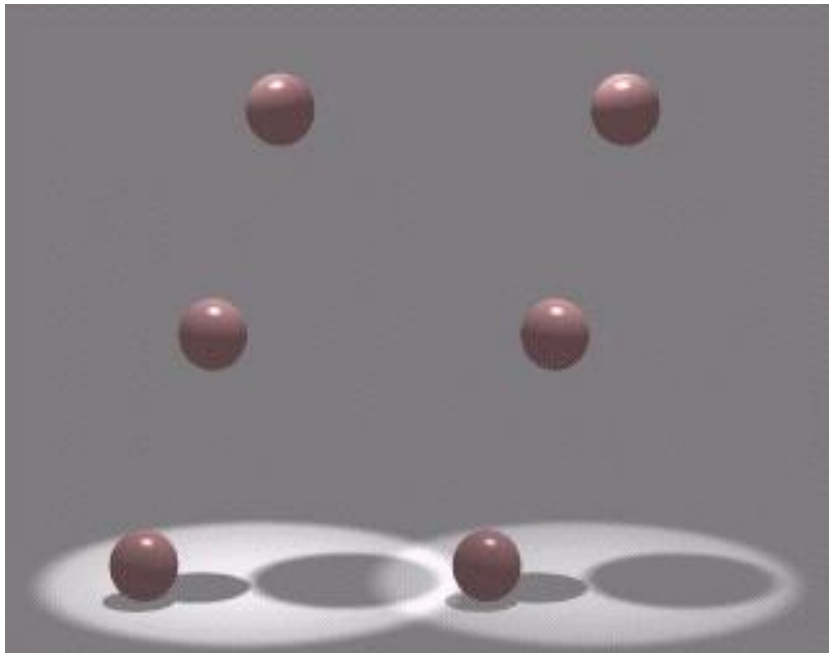
Een puntlichtbron stuurt het licht in alle richtingen, vb een gloeilamp



Spot light

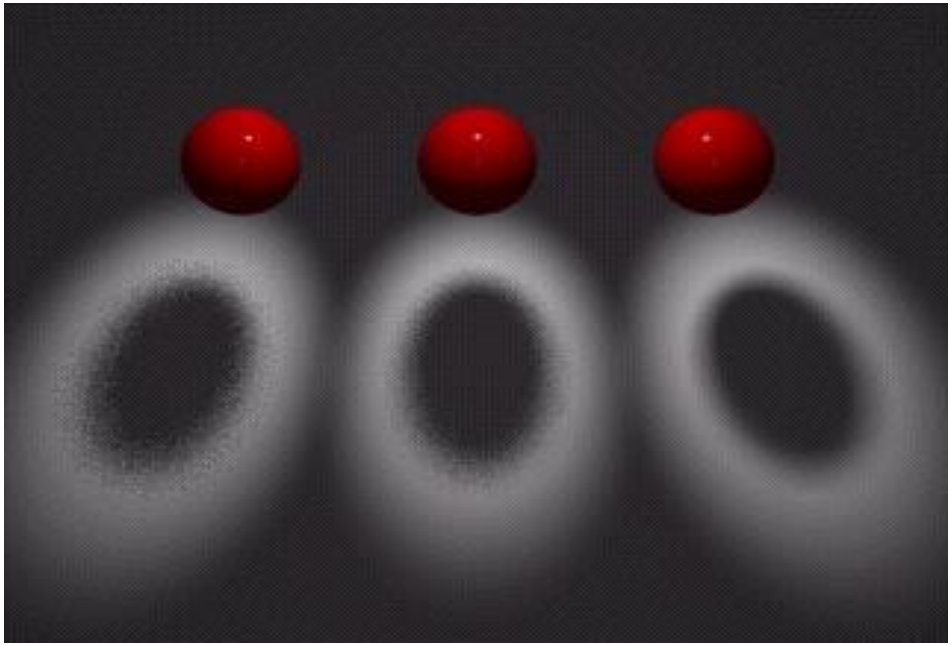
De lichtintensiteit van de centrale kegelas neemt af





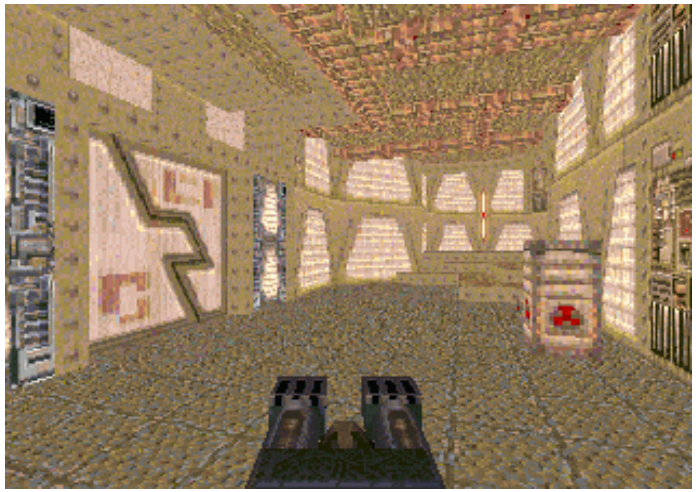
Area Light source

Een oppervlak doet dienst als lichtbron



Light Maps

Een grijswaardeafbeelding die de lichtintensiteitsverdeling voorstelt wordt vermenig-vuldigd met een basistexture



In realiteit : Radiosity

Een belicht oppervlak gedrag zich ook als lichtbron vermits het deels het licht weerkaatst

-> reflectie van reflectie van reflectie …

van links naar rechts:
geen weerkaatsingen, 1 weerkaatsing, 3 weerkaatsingen

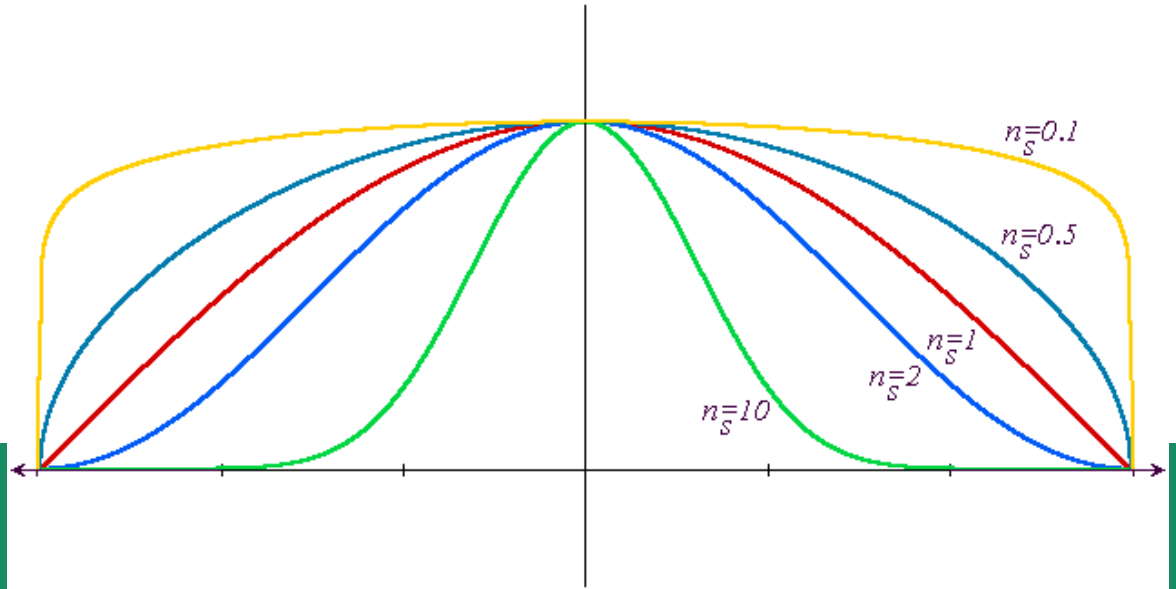
Reflectie

- Bepaalt samen met de kleur de feel van het object: hoe glanzend/mat/zacht/hard een object er uit ziet
- Sommige reflectie-algoritmes gebaseerd op optica, andere niet

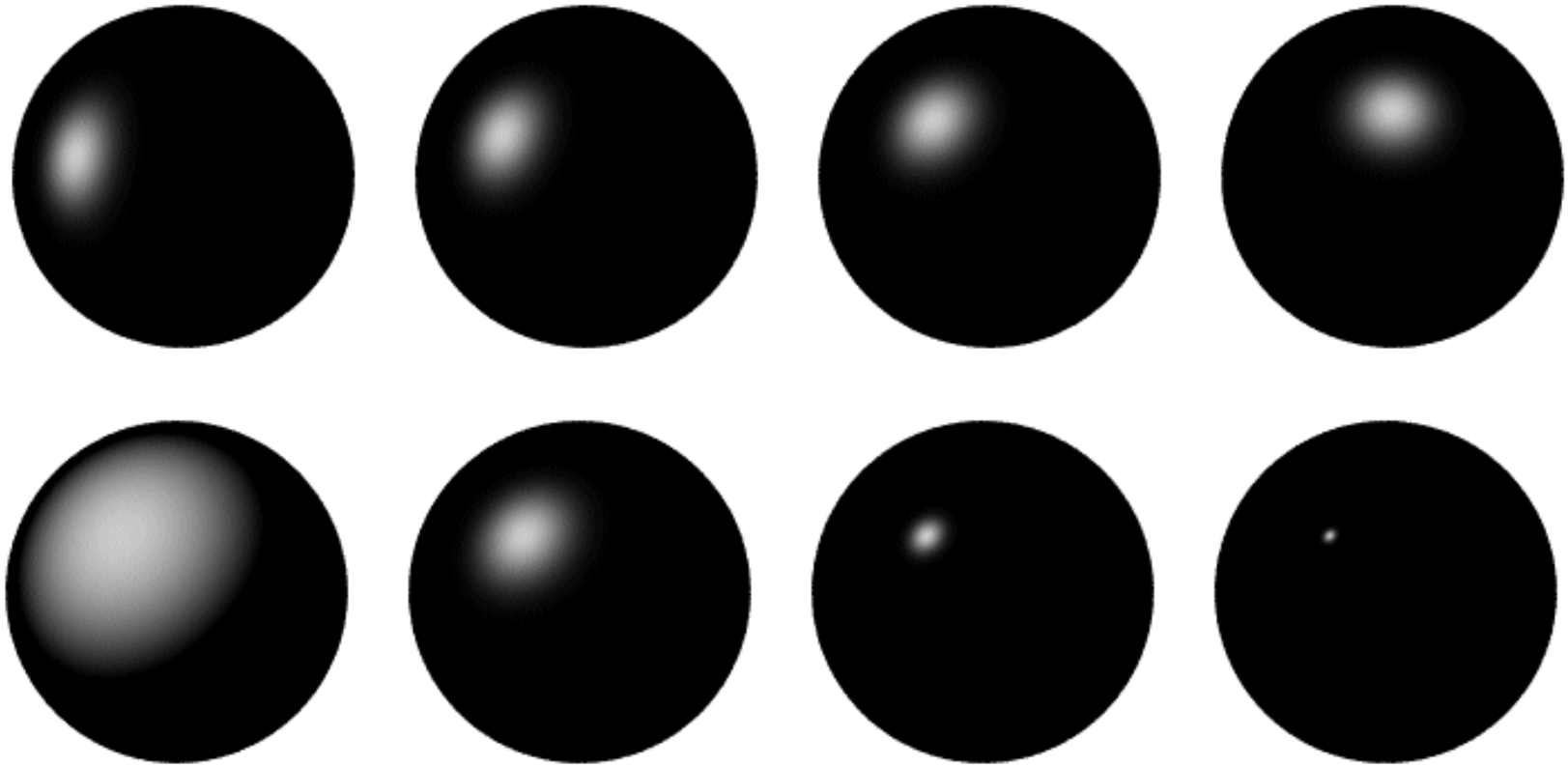
Voorbeeld: Phong reflection

- Reflectiemodel zonder wetenschappelijke basis, maar wel zeer werkbaar

$$I_{specular} = I_{light}(\cos\phi)^{n_{shiney}}$$

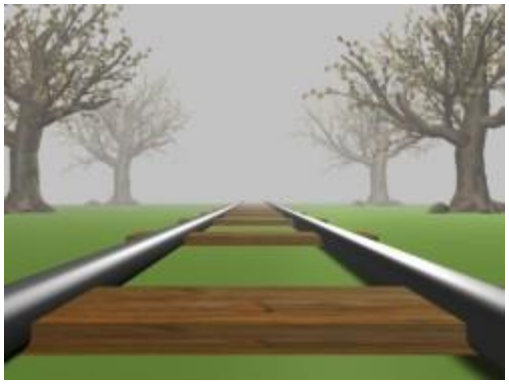


Voorbeeld: Phong reflection

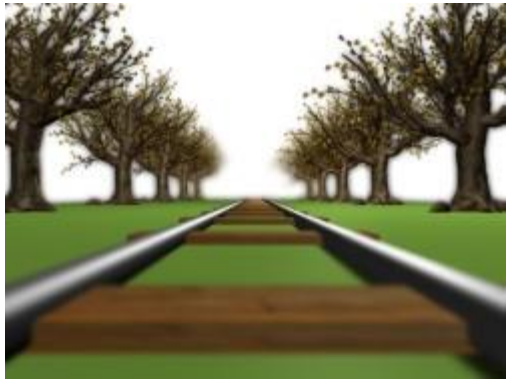


Atmosferische effecten

Fog



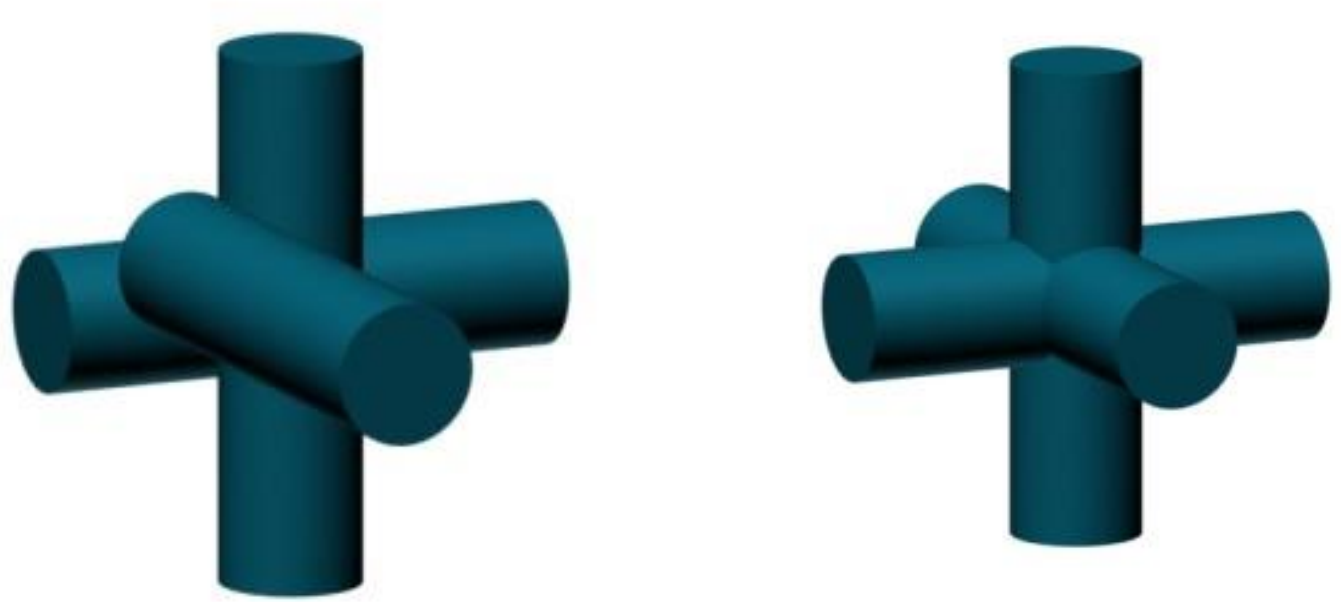
Depth-of-field



Alpha Blending

Z-buffering

De Z-buffer of dieptebuffer is een stuk geheugen dat voor elke pixel van het te renderen beeld opslaat hoe ver die pixel van de camera af ligt. Als er dan op diezelfde pixel nog een polygoon getekend gaat worden kijkt de renderer of de afstand van die nieuwe pixel wel dichterbij is dan de huidige pixel. Als dat zo is wordt de nieuwe pixel getekend en wordt ook de waarde in de dieptebuffer aangepast. Als de nieuwe pixel verder weg ligt wordt de oude pixel behouden.



Wiskundige transformaties

Lineaire transformaties:

- Rotatie
- Schaling

Affiene transformaties:

- Translatie

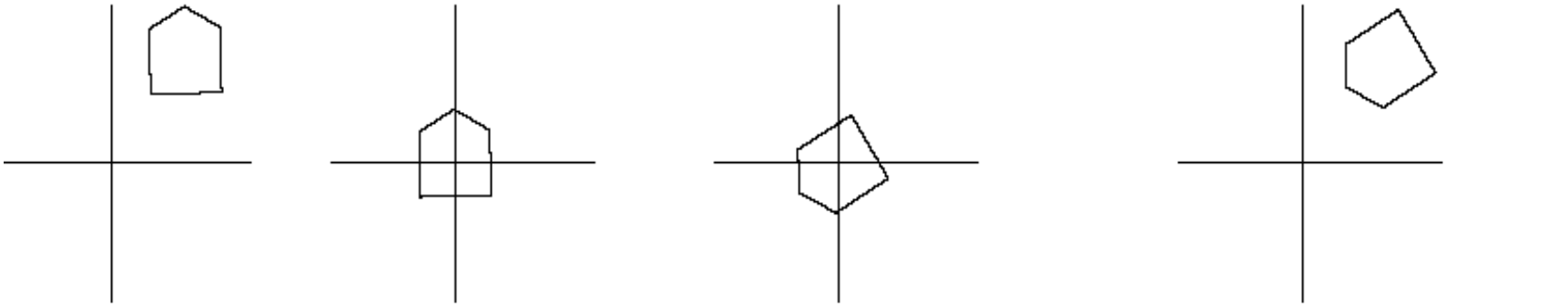
Projectie

Een transformatie wordt wiskundig uitgedrukt adhv een matrix, de transformatiematrix

Voorbeeld samenstelling

Let op de volgorde van vermenigvuldiging van matrices: de eerste bewerking komt laatst in de vermenigvuldiging!

$House\ (H)$ $T(dx, dy)H$ $R(\theta)T(dx, dy)H$ $T(-dx,-dy)R(\theta)T(dx, dy)H$



De 3D transformatiematrices

Merk op dat er drie rotatiematrices zijn!

(in 2D : roteren om een punt)

translatie

$$\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

schalen

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rotatie rond de X-as

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rotatie rond de Y-as

$$\begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rotatie rond de Z-as

$$\begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

WPF

Zie WPF introslides