Realtime 3D

# Inleiding

## Korte geschiedenis van 3D engines

### Jaren ’80:

* beperkte hardware
* geen engines ⭢ voor elke game werd alle code zelf geschreven
* later: snelle evolutie van arcade-hardware + in-house game engines

### Jaren ’90 en ‘00

* Eerste 3D games: Doom en Quake
* Verkoop licenses
* Licensing model: Quake III & Unreal
* Engine ⭤ Content

### Nu

* Toegankelijker en goedkoper ⭢ aantrekkelijk voor independent devs
* Higher level programmeertalen
* Cross-platform
* o.a. Unity, Unreal Engine, CryEngine, Source Engine, RageEngine, Blender, JMonkey3D

## Wat is een 3D engine

* developer focus = highlevel
* een framework (verzameling van componenten/bouwblokken)
* **scripting** (game loop; logica van de game/applicatie)
* talen: C++, C#, Python, Java, Javascript, Lua, …
* IDE’s: MonoDevelop, Visual Studio, Eclipse, Notepad
* **level-editor** (het creeren en aanpassen van levels)
* **input** (interactiviteit, o.a. keyboard, muis, joystick, gamepad, touch, leapmotion, kinect)
* **graphics**
* o.a. assets importeren (o.a. 3D models, textures), shaders, materials, lighting & shadows, particles, post processing effects, animation
* **physics**
* o.a. zwaartekracht en andere krachten, collision detection, fluid dynamics, ragdolls
* physics engines: Havok, PhysX, ODE, Box2D
* **audio** (3D positional sound (volume, reverb, distortion), sound effects & sound input)
* **network**
* high-level network programming, geen zorgen maken over zaken als TCP/UDP
* cloud-based oplossingen zoals photon, Google Play Game Services, etc.
* **AI & pathfinding** (nabootsen van intelligentie)
* **GUI** (Graphical User Interface)
* o.a. HUD, knoppen, menu’s
* vaak niet out of the box ⭢ plugin-alternatieven beschikbaar
* **build** (creeren van executables)
* exporteren naar verschillende platformen
* optimalisaties per platform

## Unity

een cross-platform game engine die ontwikkeld werd door Unity Technologies.

* lage leercurve
* veel features
* snelle resultaten
* programmeertaal is C# (lijkt op Java)
* meegeleverde IDE is Monodevelop
* is **component based**:
* een project bestaat uit GameObjects
* component: voegt functionaliteit toe aan GameObjects
* GameObject: een container voor componenten
* elk GameObject heeft ten minste een Transform-component (positie/rotatie/schaal)
* voordeel: flexibel!
* zelf maken: nieuwe C#-klasse maken en laten overerven van MonoBehaviour
* Heeft standaard 2 methodes:
* Start (wordt eenmalig uitgevoerd bij initialisatie)
* Update (wordt iedere frame uitgevoerd)

### Componenten

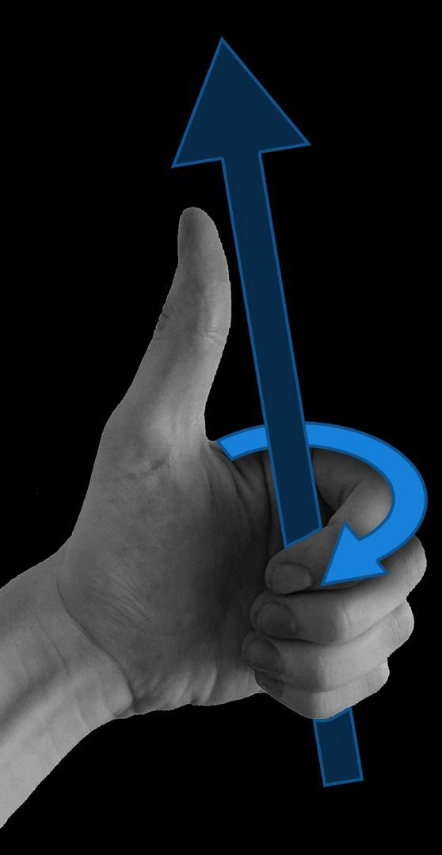
* **Transform**: positie, rotatie en schaal van het 3D-object
* **Mesh Filter**: bepaalt de vorm van het 3D-object
* kan verwijzen naar een 3D-model uit onze assets of een standaard vorm zoals bv. “Cube”.
* **Box Collider**: definieert een zogenaamde *bounding box* om te bepalen wanneer een 3D-object kruist met een andere object, o.a. nuttig voor physics.
* **Mesh Renderer**: zorgt ervoor dat ons 3D-object ook effectief op het scherm gerenderd wordt
* **Material**: bepaalt de kleur en enkele basiseigenschappen m.b.t. uiterlijk en belichting

# Mesh

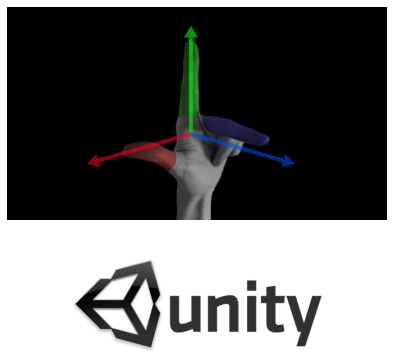
## Coördinatensysteem

3 richtingen:

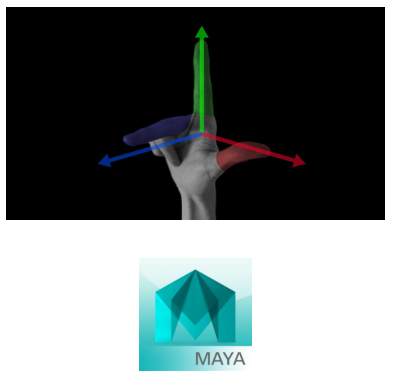
* Links – Rechts (**X = duim**)
* Boven – Onder (**Y = wijsvinger**)
* Voor – Achter (**Z = middenvinger**)



# Rotatie



# Linkshandig



# Rechtshandig

## Anatomie van een mesh

**mesh** (*een maas*):

* in het echt: een maas (een barrière gemaakt van verbonden strengen van metaal, vezels of andere flexibele/kneedbare materialen)
* in 3D computer graphics: een verzameling van vertices, edges en faces die de vorm van een veelhoekig object definiëren

**vertex**: een punt met een bepaalde positie in de 3D-ruimte (X, Y, Z)

* heeft geen dikte, breedte of hoogte
* is niet hetzelfde als vector ⭢ een vertex kan extra eigenschappen hebben (kleur, normaalvector, raaklijnvector, UV-coördinaten)

**wireframe**: de weergave van de verbindingen tussen alle vertices

**edge**: een verbinding tussen 2 vertices

* vormt de randen van een vlak (*face*)

**triangle**: een gesloten aaneenschakeling van 3 edges in hetzelfde vlak

* is coplanair (*alle edges liggen op één vlak*)
* de meest eenvoudig mogelijke 3D-vorm

**quad**: een gesloten aaneenschakeling van 4 edges

* niet noodzakelijk coplanair (*alle edges liggen op één vlak*)

**face/polygon/ngon**: een gesloten aaneenschakeling van minimum 3 edges

**triangulatie**: het omvormen van alle polygonen naar triangles

**back-face culling**: een techniek om niet-zichtbare polygonen niet te tekenen

* afhankelijk van de richting van de triangulatie en de positie van de camera

## Vectoren

**vector**: een geometrische waarde met een lengte en een richting

* wordt gevisualiseerd als een pijl van de oorsprong (0,0,0) naar een bepaald punt (x,y,z)
* de positie is niet van belang
* lengte berekenen d.m.v. de stelling van Pythagoras ⭢ √(x²+y²)
* richting berekenen d.m.v. goniometrie (tangens) ⭢ tan(α) = y / x

**eenheidsvector**: genormeerde vector waarvan de norm 1 is

* bekomen door de posities van 2 obj. van elkaar af te trekken en te delen door de lengte v.d. vector

## Normaalvectoren

**normaalvector**: een vector die loodrecht staat op een vlak

* belangrijk om lichtinval op vlakken te kunnen berekenen
* bepaalt de hoek van de lichtbron en van de kijker t.o.v. het vlak

**eenheidsnormaalvector**: een vector met de een richting van een normaalvector en lengte 1

## Hard/soft edges

**soft edge**: meerdere vlakken delen dezelfde vertices (*shared vertices*)

* 1 normaalvector per hoek
* goed voor ronde vormen, niet voor hoekige
* de vlakken worden belicht alsof er maar 1 vlak zou zijn

**hard edge**: vertices worden niet gedeeld door meerdere vlakken (*duplicate vertices*)

* 1 vertex en 1 normaalvector per aangrenzend vlak
* Er is een duidelijke scheiding van belichting tussen de 2 vlakken

## Meshmanipulatie

Posities van de vertices in real-time wijzigen is erg CPU-intensief

* vaak beter om shaders te gebruiken

**procedurele generatie**: het genereren van data/content met behulp van een algoritme

**random procedurele generatie**: volledig willekeurige generatie van data

**seeded procedurele generatie**: de data wordt gegenereerd op basis van een seed

* seed: een random gegenereerd getal waarop men zich baseert om de data te genereren
* elke keer je dezelfde seed gebruikt, krijg je dezelfde data

# Transformaties & Physics

## Transformaties

**Transform-component**: het assenstelsel van een GameObject

* Omvat de positie, rotatie en schaal
* Elke spelwereld heeft een default assenstelsel met oorsprong (0,0,0), rotatie (0,0,0) en schaal (1,1,1)

**Hoeken van Euler**: beschrijven de rotatie als een samenstelling van drie rotaties om de coördinaatassen

* Roteren rond de x, y en/of z-as
* Rotatie van 0 – 360°
* Erg goede uitleg van Eulerhoeken en Gimball locks: <https://youtu.be/zc8b2Jo7mno>

**Gimbal lock**: een probleem dat optreedt bij hoeken van Euler, waarbij 2 van de 3 rotatieassen op hetzelfde vlak komen te liggen, en daardoor een bepaalde rotatieas *gelockt* wordt (er dus niet meer direct rond die as gedraaid kan worden)

**Quaternions**: een alternatief voor hoeken van Euler, om rotaties rond 3 assen voor te stellen

* Heeft naast **x**, **y** en **z** een vierde waarde: **w**
* Voordelen:
* geen Gimbal locks
* vlotte, directe en consistente interpolatie (tegenover Eulerhoeken)
* eenvoudig om berekeningen mee te doen

## Colliders

**collision detection**: berekenen of 2 of meer objecten elkaar raken

* wiskundige raakpunten bepalen op basis van de vorm van objecten
* doel: objecten op elkaar te laten reageren (bv. botsen)
* twee fases:

1. broad phase: bekijken of er een mogelijke collision is

* niet gedetailleerd
* om te vermijden dat er uitgebreide berekeningen gebeuren als de twee objecten niet eens in de buurt van mekaar liggen

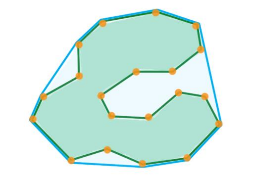
1. narrow phase: de effectieve, precieze collisie berekenen

**collider**: een geometrische vorm die gebruikt wordt om collision detection te berekenen tussen objecten

**primitive colliders**: colliders op basis van eenvoudige geometrische figuren (o.a. box, sphere, capsule)

**compound colliders**: een samenstelling van *primitive colliders* om meer complexe vormen te dekken

**mesh colliders**: de volledige wireframe van de mesh wordt gebruikt om de collision te testen

* 2 soorten:
* bolle mesh colliders (*convex*, vereisen het minste rekenkracht)
* holle mesh colliders (*concave*)
* convex hull: holle objecten sneller berekenen ⭢ omhullen in een bolle vorm (vereenvoudigde weergave, dus kan zorgen voor ongewenste effecten)
* convex decomposition: opdelen in meerdere bolle vormen ⭢ nauwkeuriger

**simplified mesh colliders**: een vereenvoudigde versie van de mesh wordt gebruikt om de collision te testen

## Physics

**physics**: simulatie van de wetten van de fysica

* vereenvoudiging van de werkelijkheid

**rigid body**: een hard, niet vervormbaar object

* er kunnen krachten op uitgeoefend worden
* kunnen met elkaar botsen
* gemaakt van een fysisch materiaal

### Krachten

**friction force** (frictie): kracht die ontstaat wanneer 2 objecten langs elkaar schuiven

* hoeveelheid wrijving hangt af van de ruwheid van de oppervlakken
* zorgt voor vertraging ⭢ brengt bewegende objecten tot stilstand
* aanpassen in Unity door een Physic Material te creëren en toe te wijzen aan een collider component van een GameObject

**static friction force**: de wrijvingskracht tussen 2 stilstaande objecten

* deze kracht moet overwonnen worden om een object in beweging te krijgen

**dynamic friction force**: de wrijvingskracht tussen 2 objecten, waarvan er minstents 1 beweegt

* deze kracht moet overwonnen worden om een object in beweging te houden
* kleiner dan de static friction force

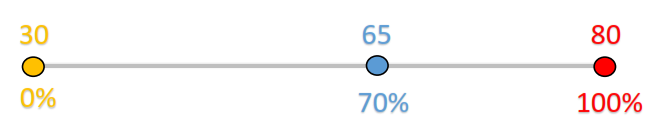
# Animation

animatie: de illusie van beweging door het na elkaar afspelen van verschillende stilstaande beelden, zogenaamde frames

## Interpolatie

**interpolatie**: de onbekende waarde bepalen tussen 2 gekende waardes, op basis van een percentage

**lerp** (*linear interpolation*): elke frame verplaatst het object een bepaald percentage van de volledige afstand



## Coroutines

**coroutine**: een IEnumerator die toelaat om code te schrijven buiten de Update-methode, die ook elke frame wordt uitgevoerd

* laat toe iteratiestructuren te gebruiken die elke frame gepauzeerd worden, zodat de volgende frame berekend kan worden (*cooperative multitasking*)

**cooperative multitasking**: het uitvoeren van code wordt afgewisseld tussen verschillende processen (routines), om de Update-methode niet te zwaar te belasten en de framerate hoog te houden

## Tweening

**tweening** (*in between*): automatisch animeren tussen twee waarden.

## Model animations (Mecanim)

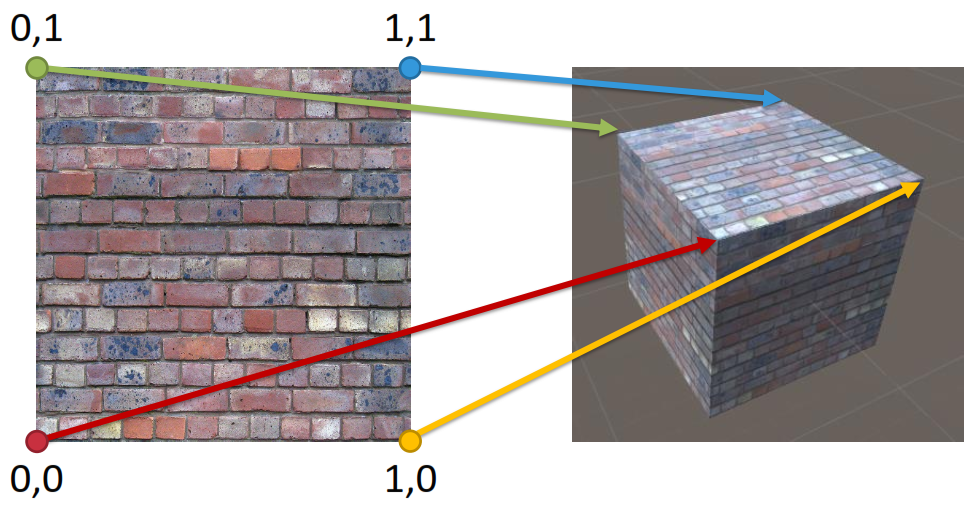
**in place motion**: de animatie verplaatst het object niet, m.a.w. het object blijft *in place*

**root motion**: de verplaatsing (*translate*) gebeurt door de animatie zelf

**Mecanim**: een animatiesysteem meegeleverd met Unity

# UV-mapping

**UV-mapping**: 2D textures mappen op een 3D-object op basis van vertices



## Textures

**POT-texture**: een texture waarvan de resolutie vierkantig en een macht van twee is

* 2x2, 4x4, 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, 2056x2056, 4096x4096

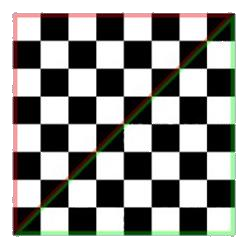
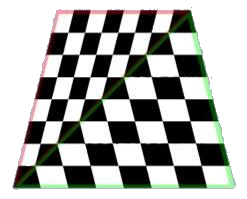
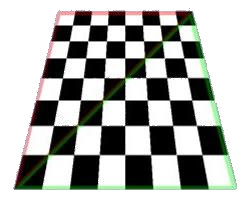
**NPOT-texture**: een texture van eender welk formaat

* minder optimalisatie ⭢ minder performant
* niet ondersteund door oudere hardware

**affine texture mapping**: coördinaten worden lineair geïnterpoleerd over een vlak, per triangle

**perspective correct texture mapping**: coördinaten worden berekend t.o.v. de kijkhoek

* trager, maar correcter

flat mapping affine mapping correct mapping

## Vertex colors

Iedere vertex heeft één kleur, maar kleuren kunnen worden geïnterpoleerd tussen vertices

## Raycasting

**raycasting**: een straal tekenen vanuit een bepaald punt, in een bepaalde richting

* met als doel een raakpunt tussen en lijn en een vlak/object te bepalen
* m.b.v. een vector, het bereik is de lengte van de vector

## Camera

**viewpoint**: het standpunt van de camera

**viewing frustrum**: het deel van de wereld die op beeld zichtbaar is

* alles tussen de near en far clipping pane
* zelfde als de *field of view* van de gewone camera

**near clipping plane**: alles wat dichter is bij de viewpoint dan de near clip plane, wordt niet getoond

**far clipping plane**: alles wat verder is dan de far clip plane, wordt niet getoond

**raytracing**: een 3D-scene wordt *gefotografeerd*, met als doel een 2D afbeelding te verkrijgen.

* het 2D-beeld wordt pixel per pixel opgebouwd
* minder geschikt voor realtime 3D

# Model selection & performance

## Eigenschappen van 3D-modellen die de performance beïnvloeden

* aantal vertices
* hard/soft edges
* aantal triangles
* mobiel: best tussen 300 – 1.500
* low-end: best tussen 500 – 20.000
* high-end best tussen 20.000 – 100.000
* next-gen: best tussen 100.000 – 500.000
* aantal materials
* grootte/kwaliteit van textures
* LOD (level of detail)
* Shaders

## CPU ⭤ GPU

**CPU**: doet complexe, eenvoudige berekeningen relatief traag

* berekent o.a. game logic, lichtinval, voorbereiding van shaders, communicatie naar GPU
* aantal objecten is belangrijk, aantal triangles minder

**GPU**: doet meerdere eenvoudige berekeningen tegelijk en snel

* ieder object wordt naar de GPU gestuurd
* meerdere malen per material en per real-time licht

## Performantie

**draw calls**: een vraag van de engine naar de graphics API (bv. OpenGL, Direct3D) om een object te renderen

* aantal draw calls = aantal objecten die gerenderd worden (zo laag mogelijk houden)

**draw call batching**: het aantal draw calls verminderen door

* niet-bewegende objecten samen te voegen tot één grote mesh (static batching)
* bewegende meshes die klein genoeg zijn transformeren op de CPU, en gelijkaardige resultaten combineren tot één grote mesh (dynamic batching)
* objecten moeten hetzelfde material hebben

**LOD** (*level of detail*): efficiënter renderen door het verlagen van de kwaliteit van objecten door vertextransformaties, als het object ver van de camera is, van lagere prioriteit is, etc.

* verschillende versies van een model worden voorzien, met verschillende *levels of detail*

**object pooling**: een techniek (of design pattern) waarbij een aantal objecten geïnitialiseerd worden buiten het zicht van de camera om deze daarna te kunnen gebruiken wanneer ze nodig zijn

* een nieuw object moet zo niet telkens gecreëerd en vernietigd worden
* heel wat rekenkracht wordt uitgespaard

**culling**: een techniek die ervoor zorgt dat objecten die buiten het gezichtsveld van de camera liggen niet gerenderd worden

* zorgt voor een lagere GPU-belasting
* bv.

**frustrum culling**: het weglaten van objecten die zich niet in het frustrum van de camera bevinden

* wordt automatisch gedaan door Unity

**occlusion culling**: het weglkaten van objecten die niet zichtbaar zijn doordat één of meerdere andere objecten in de weg staan

* moet manueel geactiveerd worden in Unity door een occlusion map te *baken*

**mipmapping**: kleinere resolutie textures gebruiken voor kleinere triangles

* minder data om te versturen naar de GPU

# Normal mapping & lighting

## Lighting

**licht**:

* in het echt: een combinatie van deeltjes (*photons*) en golven (*waves*)
* in 3D computer graphics: rays; vectoren met een richting en sterkte

**ambient lighting**: het licht komt gelijkmatig uit alle richtingen (omgevingslicht)

* er is geen directe lichtbron

**diffuse lighting**: het object weerkaatst het licht van één of meerdere lichtbronnen in verschillende richtingen

* bij ruwe oppervlakken

**specular lighting**: het object weerkaatst het licht van één of meerdere lichtbronnen in één richting

* bij gladde oppervlakken
* geeft een glazend effect

## Soorten lichtbronnen

**directional light**: lichtstralen gaan maar in één richting, over de hele scène

* de positie van de lichtbron is niet belangrijk, enkel de rotatie
* bv. de zon

**point light**: lichtbron zendt uit in alle richtingen

* de intensiteit verlaagt naargelang de afstand van de bron
* alles buiten de range ontvangt geen licht

**spotlight**: de lichtbron zendt uit binnen een bepaalde hoek

* bv. een zaklamp, autolichten

**area light**: de lichtbron zendt uit in alle richtingen van 1 kant van een voorgedefinieerde rechthoek

* realistischer dan een spotlight, maar veel minder performant
* niet at runtime, enkel prerender (baking)

## Soorten technieken

**cookie**: een schaduw in de vorm van een texture ⭢ veel performanter dan schaduwen te berekenen

**global illumination**: meest realistisch mogelijke simulatie van belichting

* beste techniek = raytracing
* niet performant genoeg voor realtime toepassingen ⭢ allerlei trucs toepassen om het resultaat te benaderen

**ambient occlusion**: vanuit ieder punt op het model een groot aantal rays casten in alle richtingen

* Als de ray een ander object of deel van hetzelfde object raakt ⭢ minder lichtinval van de omgeving

**Screen Space Ambient Occlusion** **(SSAO)**: iedere frame de ambient occlusion berekenen op basis van gerenderde objecten

* ook bewegende objecten!
* minder accuraat dan AO-map

**light probes**: bewegende objecten ook ambient light laten ontvangen

* minder accuraat dan AO-map

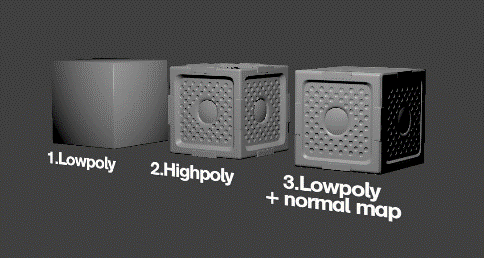
**reflection probes**: spiegeling tonen op glanzende objecten

## Normal mapping

**light maps**: vooraf berekenen van statische belichting

* Geen realtime berekeningen nodig ⭢ veel performanter
* Gebeurt automatisch in Unity voor statische objecten

**bump maps**: reliëf simuleren a.d.h.v. textures i.p.v. geometrie

* bevat hoogteinformatie: hoe witter, hoe hoger, hoe zwarter, hoe lager
* het contrast bepaalt de sterkte

**normal maps**: bevat hoogteinformatie en richting/kanteling

* zorgt voor realistischere belichting
* verkregen door een high poly model te *baken* op een low poly model

**retopologizing**: het omzetten van een model met veel polygonen (vaak gemaakt door sculpting) naar een model met minder polygonen, geschikt voor gebruik in realtime toepassingen