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

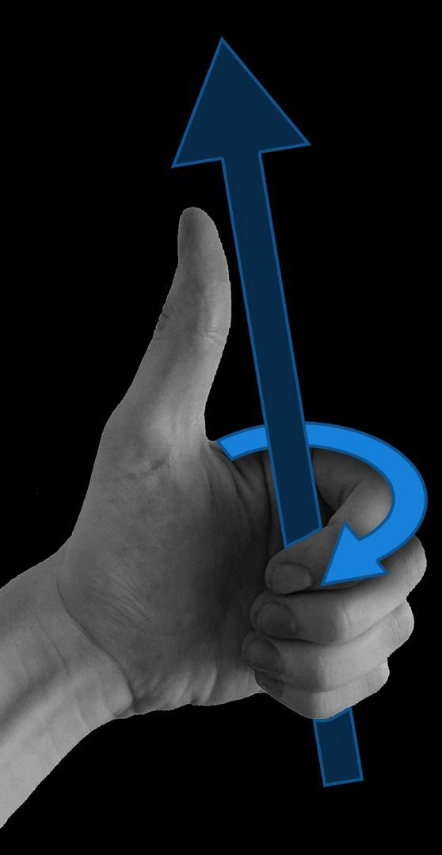
* **Transform**: positie, rotatie en schaal van het 3D-object
* **Mesh Filter**: bepaalt de vorm van het 3D-object
* kan verwijzen naar een 3D-model uit onze assets of een standaard vorm zoals bv. “Cube”.
* **Box Collider**: definieert een zogenaamde *bounding box* om te bepalen wanneer een 3D-object 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

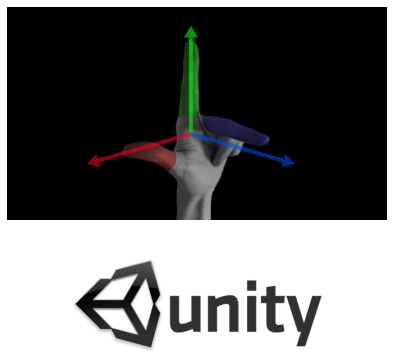
## Coördinatensysteem

3 richtingen:

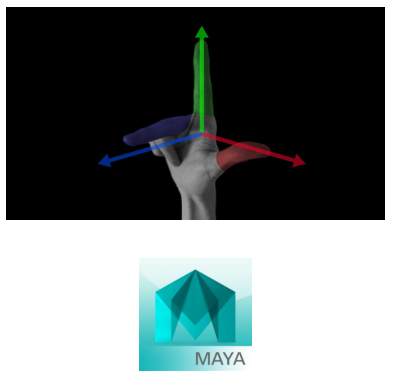
* Links – Rechts (**X = duim**)
* Boven – Onder (**Y = wijsvinger**)
* Voor – Achter (**Z = middenvinger**)



# 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-coordinaten)

**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

Posites 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 procedurale generatie**: volledig willekeurige generatie van data

**seeded procedurale 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

## 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

**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

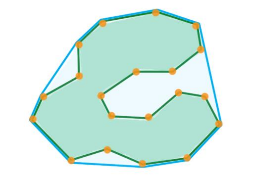
* niet gedetailleerd
* om te vermijden dat er uitgebreide berekeningen gebeuren als de twee objecten niet eens in de buurt van mekaar liggen

1. narrow phase: de effectieve, precieze collisie berekenen

**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

### Krachten

**friction force**: wrijving tussen 2 oppervlakken

* zorgt voor vertraging ⭢ brengt bewegende objecten tot stilstand

**static friction force**: kracht nodig om een object in beweging te krijgen

**dynamic friction force**: kracht nodig om een object in beweging te houden

* kleiner dan de static friction force