

UNITY PER VR

**INTRODUZIONE TEORICO-PRATICA ALLO SVILUPPO SOFTWARE
PER REALTÀ VIRTUALE USANDO UN GAME ENGINE MODERNO E
SCRIPTING OBJECT-ORIENTED: UNITY E C#**

UNIVERSITA' DI PISA - 10,17 MARZO 2023 - GIUSEPPE TURINI

RINGRAZIAMENTI

Questo seminario e' stato reso possibile grazie a:

- Il Prof. Vincenzo Ferrari per l'invito ed il supporto del seminario.
- Il Dipartimento di Ingegneria dell'Informazione (DII) dell'Università di Pisa per ospitare, aver pubblicizzato, e per l'organizzazione di questo seminario.
- Tutti i colleghi del centro EndoCAS per la continua collaborazione nella ricerca, e per avermi ospitato tutti questi anni durante il mio non-teaching term.
- La Kettering University per il continuo supporto nella mia attivita' accademica.



BIO E CONTATTI

Nome: Giuseppe Turini, Assoc. Prof. of Comp. Sc. @ Kettering University

Website: sites.google.com/view/turinig www.kettering.edu

Contatti: turinig@gmail.com gturini@kettering.edu

Istruzione: M.Sc. in Informatica @ Universita' di Pisa
Ph.D. in Health Technologies @ Universita' di Pisa

Ex-Labs: EndoCAS Research Center @ Universita' di Pisa
Visual Computing Lab @ CNR di Pisa

Didattica: Programmazione 1/2/3, Computer Graphics, VR, Game Development

Ricerca: Computer-Assisted Surgery, VR-AR, HCI-HMI

NOTE BIOGRAFICHE (2): KETTERING UNIVERSITY



AVVERTENZE

Questo seminario e' stato progettato considerando un pubblico con background ingegneristico (conoscenza base di HW/SW e programmazione).

Il materiale teorico e gli esempi applicati sono basati su Unity (2021.3.18), Microsoft Visual Studio 2019, e piattaforma Windows (11 Pro, x64-based).

Gran parte della terminologia tecnica e' mensionata in Inglese (soprattutto per evitare incorrettezze con la localizzazione Unity in Italiano)

Tutto il materiale usato (slides, scripts, e package Unity) e' disponibile online sul mio GitHub:
github.com/turinig/uxvr

CONTENUTI

Venerdì' 10 Marzo 2023 14:30-18:30

Parte 1A (~2 ore): **Introduzione a Unity**

Basics, editor e builds, gestione scena/progetto, collisioni e fisica, etc.

Parte 1B (~2 ore): **Scripting in Unity**

Script components, classi principali, event functions, etc.

Venerdì' 17 Marzo 2023 14:30-18:30

Parte 2A (~1 ore): **Sessione di Scripting Live**

Sviluppo di un mini-game da 0 (build, livelli multipli, inputs, UI, etc.).

Parte 2B (~2 ore): **Introduzione alla VR**

Fondamentali, breve storia, ecosistema HW/SW, applicazioni VR, etc.

Parte 2C (~1 ore): **Sviluppo VR App in Unity**

VR design, VR in Unity, UI, inputs, interazioni, comfort, safety, etc.

CONTENUTI (2): ARGOMENTI NON INCLUSI

Questo seminario non include i seguenti argomenti
(che non verranno trattati, o saranno soltanto accennati):

- Sviluppo apps/games 2D: docs.unity3d.com/manual/unity2d
- Graphics(texturing, shading, profiling): docs.unity3d.com/manual/graphic
- Environments(terrains, trees): docs.unity3d.com/manual/creatingenvironments
- Multiplayer/networking: docs.unity3d.com/manual/unet
- Audio(mixing, profiling): docs.unity3d.com/manual/audio
- Animation(keyframing, controllers): docs.unity3d.com/manual/animationsection
- Navigation/pathfinding(AI, navmeshes): docs.unity3d.com/manual/navigation

PARTE 1A: INTRO A UNITY

"Unity is so much more than the world's best real-time development platform - it's also a robust ecosystem designed to enable your success. Join our dynamic community of creators so you can tap into what you need to achieve your vision."

UNITY TECHNOLOGIES

Campi di Applicazione: videogiochi, architettura, automotive, film, etc.

"Create once, deploy across 25+ leading platforms and technologies to reach the largest possible audience."

UNITY TECHNOLOGIES

Vedi: unity.com

UNITY: GAME ENGINE MODERNO

Game Engine: Un game engine e' una applicazione software progettata principalmente per lo sviluppo di video giochi, e generalmente include anche librerie di codice e altri programmi di supporto (e.g., mesh editing/texturing, etc.).

Le funzionalita' di base solitamente includono:

- Un modulo di **visualizzazione** (renderer).
- Un modulo di **collision detection** (CD).
- Un modulo per la **fisica interattiva** (physics engine).
- Un modulo per la compilazione su piattaforme multiple (builder).
- Etc.

Game Engine Moderno: Un game engine moderno e' una estensione di un game engine tradizionale che integra funzionalita' extra, solitamente per lo sviluppo di applicazioni VR, AR, interactive simulators, etc.

UNITY: TERMINOLOGIA

Unity Project: Progetto che rappresenta una applicazione sviluppata in Unity.

Unity Project Folder: Cartella contenente tutti i dati di un progetto Unity.

Scene: Modulo/parte/livello di un progetto Unity.

Asset: Elemento (immagine, font, etc.) importato nel progetto e pronto all'uso.

Assets Folder: Sottocartella del Unity Project Folder contenente tutti gli assets.

GameObject: Elemento base per creare una scena, configurabile tramite componenti.

Component: Elemento di un gameobject per configurarne il comportamento.

Script: File di codice integrato in un progetto Unity.

Script-Component: Uno script configurato per essere un custom-component.

Build: Compilazione del progetto Unity nella sua versione eseguibile.

UNITY: MODULI PRINCIPALI

Unity consiste di molteplici “moduli”, ognuno progettato per svolgere una funzione specifica. Questa di seguito e’ una breve lista dei moduli principali:

- **Unity Hub:** Software di interfaccia con progetti/installazioni/tutorials Unity, permette di gestire versioni multiple di Unity.
- **Unity Editor:** Software di interfaccia con il game engine, permette di configurare assets (modelli 3D, scripts, immagini, fonts, etc.), integra funzionalita’ di editing 3D, ed estende l’IDE collegato per l’esecuzione/debugging del codice.
- **Unity Framework:** Libreria delle classi Unity (C#), permette di creare script-components e facilita lo sviluppo di codice. Disponibile sia online che locale.
- **Unity Manual:** Manuale utente/sviluppatore. Disponibile sia online che locale.
- **Unity Asset Store:** Negozio/repository online per assets/moduli/plugins per facilitare lo sviluppo in Unity.

UNITY: USARE LO EDITOR

Lo Unity Editor e' il software principale per lo sviluppo di applicazioni in Unity. Consiste in una interfaccia dockable e multi-windows, con layout configurabile. Il suo compito principale e' abilitare la configurazione di assets, e il loro controllo via script.

Quella che segue e' una breve introduzione all'editor e alle sue windows principali. Altre windows importanti sono elencate negli **Appendici**. Per la lista completa delle windows disponibili: **menu > Window**

Vedi: docs.unity3d.com/manual/usingtheeditor

UNITY: USARE LO EDITOR (2)



UNITY: USARE LO EDITOR (3)

Questa e' la lista delle windows e pannelli principali dello Unity Editor:

- **Toolbar (A):** Include controlli "play mode", "visibility layers", e editor layout.
- **Hierarchy (B):** Visualizzazione gerarchica-testuale della scena corrente.
- **Game View (C):** Visualizzazione a schermo dell'app come acquisita dalla camera virtuale. Utilizzare i controlli "play mode" per iniziare la simulazione.
- **Scene View (D-E):** Visualizzazione 3D della scena corrente. Include elementi visivi di assistenza(gizmos). Scene e Hierarchy mostrano lo stesso contenuto.
- **Inspector (F):** Pannello context-sensitive per visualizzare/configurare le proprieta dell'elemento correntemente selezionato(asset, gameobject. etc.).
- **Project (G):** Visualizzazione di tutti gli assets del progetto. Specchio del contenuto della sottocartella "Assets" nella cartella di progetto.
- **Statusbar (H):** Notifiche e accesso rapido a moduli relativi.

FONDAMENTALI: SCENES

Scene: Una scena e' simile a un livello del gioco, o a una parte/modulo della app.

Per esempio: puoi creare una scena per un menu, o per i crediti finali, etc.

Lo Unity Editor permette di creare/caricare/salvare scene, e il loro caricamento singolo o multiplo puo' anche essere controllato a runtime via scripting.

Multi-scene: Nell'editor si possono caricare scene multiple a fini di editing, e via script si possono caricare scene multiple a runtime (in diverse modalita').

Vedi: docs.unity3d.com/manual/quickstart3dcreate

Vedi: docs.unity3d.com/manual/creatingscenes

Vedi: docs.unity3d.com/manual/multisceneediting

FONDAMENTALI: GAMEOBJECTS

GameObject: Elemento base per creare una scena, configurabile tramite component.

Un GameObject puo' rappresentare elementi 3D (props), caratteri (NPC), etc. Sostanzialmente, ogni elemento della app/gioco e' un GameObject.

Un GameObject esiste in 3D solo se incluso in una scena.

Esempio: Creando una nuova scena (**menu > File > New Scene > Basic (Built-In)**), la Hierarchy window mostra il nome della scena creata/caricata (**Untitled**) che include 2 GameObjects, **Main Camera** e **Directional Light**.

Vedi: docs.unity3d.com/manual/quickstart3dcreate

Vedi: docs.unity3d.com/manual/class-gameobject

Vedi: docs.unity3d.com/manual/creatingscenes

FONDAMENTALI: COMPONENTS

Component: Il comportamento (behavior) di un GameObject e' determinato da "blocchi di funzionalita" chiamati componenti (Components).

Un Component di un GameObject puo' essere aggiunto/rimosso/riconfigurato sia nell'editor che a runtime via scripting.

Ogni component e' rappresentato da una classe dello Unity framework.

Lo sviluppatore puo' creare "custom components" implementando degli scripts specifici (script-components, derivando la classe **MonoBehaviour**).

Nota: L'unico Component non rimuovibile (e sempre presente) e' il **Transform**, che rappresenta la configurazione 3D del GameObject (e alcuni dati gerarchici/parenting).

Vedi: docs.unity3d.com/manual/quickstart3dcreate

Vedi: docs.unity3d.com/manual/components

FONDAMENTALI: COMPONENTS (2)

Questa e' una breve lista dei componenti piu' usati:

- **Transform**: Configurazione 3D di un GameObject, e le sue informazioni di parenting. Sempre presente, e non rimuovibile.
- **Camera**: Camera virtuale che cattura una vista dell'ambiente 3D per visualizzarlo a schermo. Una sempre necessaria, e marcata **MainCamera**.
- **Audio Listener**: Microfono virtuale che cattura i suoni dell'ambiente 3D per riprodurli in output. Massimo 1 per scena.
- **Light**: Fonte di luce per illuminare l'ambiente 3D.
- **Mesh Filter**: Definisce la geometria (modello 3D) di un GameObject.
- **Mesh Renderer**: Definisce la visualizzazione di un componente Mesh Filter.
- **Collider**: La geometria di un GameObject solo a fini di collision detection.
- **Rigidbody**: Dati fisici di un GameObject per la simulazione fisica interattiva.

FONDAMENTALI: SCRIPTS

Script: Lo sviluppatore puo' programmare molti aspetti di una app Unity implementando scripts in C# (programmazione object-oriented).

Script-Component: Lo sviluppatore puo' creare componenti particolare implementando script speciali chiamati script-component (che devono derivare dalla classe dello Unity framework chiamata **MonoBehaviour**).

Lo scripting in C# per Unity verra' discusso in dettaglio nella **Parte 1B**.

Vedi: docs.unity3d.com/manual/quickstart3dcreate

FONDAMENTALI: ASSETS

Asset: Un qualunque **elemento importato in un progetto Unity**, e pronto per l'uso.

Gli assets includono elementi visuali, audio, testo, e altro ancora.

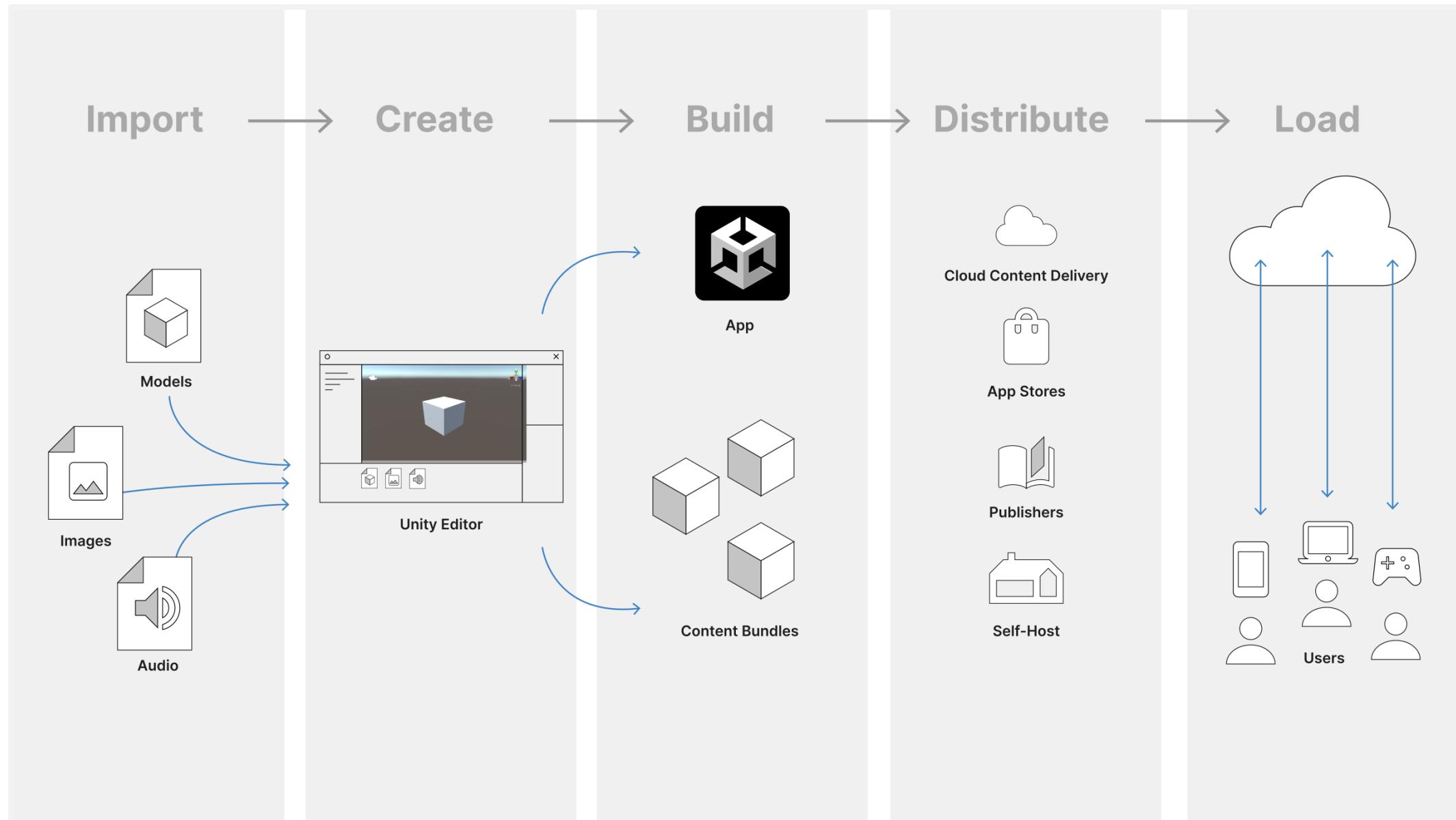
Un asset puo' essere importato dall"**esterno**" (e.g., modello 3D creato in Blender), oppure puo' essere creato all"**interno**" di Unity (e.g., animation controller).

Asset Workflow: Di solito uno sviluppatore segue la seguente procedura lavorando con assets in Unity (vedi anche diagramma seguente):

- **Importa** l'asset nello Unity editor.
- **Crea** contenuto per la tua app usando l'asset importato.
- **Build** (genera/compila) l'eseguibile della tua app in Unity.

Vedi: docs.unity3d.com/manual/assetworkflow

FONDAMENTALI: ASSETS (2)



FONDAMENTALI: PREFABS

Prefab: Un prefab(prefab-asset) consiste in un GameObject convertito in un asset a scopi di clonazione/condivisione/riuso.

Il **prefab-asset** rappresenta un template che lo sviluppatore puo' usare per creare **prefab-cloni (prefab-instances)** in una scena, o per controllare copie del prefab-asset in scene multiple. Ogni cambiamento effettuato al prefab-asset e' automaticamente applicato a tutti i suoi prefab-cloni in ogni scena.

Questo sistema permette di mantenere tutti i prefab-cloni in sync con il prefab-asset, ma allo stesso tempo permette di configurare ogni prefab-clone individualmente (se necessario).

Nota: Un prefab-asset di un modello 3D non va confuso con l'asset originale del modello 3D.

Vedi: docs.unity3d.com/manual/prefabs

COMPONENTI: TRANSFORM

Il componente **Transform** rappresenta la configurazione 3D di un **GameObject**, ovvero: posizione, rotazione, e scaling.

Inoltre memorizza anche le informazioni di parenting (vedi **Parenting**, class e scripting).

Nota: L'unico Component non rimovibile (e sempre presente) e' il **Transform**, che rappresenta la configurazione 3D del **GameObject** (e alcuni dati gerarchici/parenting).

Nota: I **GameObjects** che sono elementi della User Interface (UI) non hanno componenti **Transform**, e invece hanno componenti **RectTransform** (vedi **User Interface**).

Vedi: docs.unity3d.com/manual/class-transform

FONDAMENTALI: PARENTING

Parenting: In Unity, i GameObjects (GOs) sono organizzati in un sistema gerarchico, ovvero un GO "genitore" (parent) può includere altri GOs come figli (children).

Nota: Un GameObject senza parent usa il sistema di riferimento della scena.

Il componente Transform di un GameObject senza parent rappresenta la configurazione 3D del GO rispetto al **sistema di riferimento globale (della scena)**.

Nota: Un GameObject con parent usa il sistema di riferimento del parent.

Il componente Transform di un GameObject con parent rappresenta la configurazione 3D del GO rispetto al **sistema di riferimento locale (del GO parent)**, ed è affetto da tutte le trasformazioni geometriche dei componenti Transform degli "avi" (parent etc.).

Vedi: docs.unity3d.com/manual/class-transform

UNITY: COLLISION DETECTION

Collision Detection:

La Collision Detection (CD) e' la fase di un game engine dedicata al rilevamento collisioni (intersezioni) tra oggetti in una scena.

Di solito la CD: viene effettuata a coppie (oggetto vs oggetto), e' inclusa nel modulo che gestisce la fisica interattiva (physics engine), e la sua frequenza e' differente da quella di visualizzazione.

Collider: In Unity le collisioni sono gestite usando dei componenti speciali: i **Colliders**.

Un GameObject "capace" di collidere con altri oggetti deve avere un componente Collider, configurato in modo appropriato.

Un Collider e' una forma geometrica usata al solo fine del rilevamento collisioni.

Un Collider e' invisibile, e puo' **non** avere la stessa "forma" del suo GameObject.

Vedi: docs.unity3d.com/manual/collidersoverview

COMPONENTI: COLLIDERS

Primitive Colliders: Le forme più semplici ed efficienti da usare come Collider sono i Primitive Colliders: **Box Collider**, **Sphere Collider**, e **Capsule Collider**.

Compound Colliders: Questi Colliders sono delle aggregazioni di Primitive Colliders inclusi in un singolo GameObject.
Permettono una migliore approssimazione della forma voluta, continuando ad essere efficienti computazionalmente.

Mesh Colliders: Per specificare la forma del Collider occorre usare un Mesh Collider.
Un Mesh Collider permette la configurazione del modello 3D usato dal Collider, a scapito dell'efficienza computazionale (e altre limitazioni).

Vedi: docs.unity3d.com/manual/collidersoverview

FONDAMENTALI: COLLIDERS (2)

Static Colliders: Un GameObject che include Colliders, ma non e' attivo per la fisica interattiva (non ha componenti Rigidbody) viene detto Static Collider.

Questa configurazione viene utilizzata per elementi statici della scena.

Trigger Colliders: Nei casi in cui un Collider **serva solo per scopi di rilevamento**, e non di effettiva interazione/collisione (e.g., sensore di prossimita') occorre configurare il Collider come Trigger Collider.

Questa configurazione viene utilizzata per scopi di scripting.

Vedi: docs.unity3d.com/manual/collidersoverview

UNITY: FISICA INTERATTIVA

Fisica Interattiva: Unity include un modulo per la simulazione fisica interattiva (physics engine) che assicura che gli oggetti accelerino e rispondano alle collisioni in modo realistico etc.

Il physics engine di default in progetti 3D e': **Nvidia PhysX**.

Corpo Rigido: Nella realta', un corpo rigido e' un oggetto fisico che non si deforma all'applicazione di forze. Ovvero: la distanza tra una qualunque coppia di punti di un corpo rigido rimane costante nel tempo, indipendentemente dalle forze esterne applicate.

Rigidbody: Unity simula la fisica del corpo rigido usando il componente **Rigidbody**.

Vedi: docs.unity3d.com/manual/rigidbodiesoverview

COMPONENTI: RIGIDBODIES

Rigidbody: Il componente Rigidbody fornisce caratteristiche fisiche ad un GameObject, e (di default) abilita' il physics engine al controllo del GameObject in posizione (gestendo collisioni, applicando forze, calcolando velocita', etc.).

Rigidbody Collider: Un GameObject con un Rigidbody component (non Kinematic) che include anche un Collider (non Trigger).

Kinematic Rigidbody Collider: A volte si puo' volere che la CD (physics engine) rilevi le collisioni di un GameObject, ma non si vuole che il physics engine ne controlli il movimento. In questi casi occorre configurare il componente Rigidbody come Kinematic.

Vedi: docs.unity3d.com/manual/rigidbodiesoverview

COMPONENTI: COLLIDER-RIGIDBODY

Queste sono tutte le configurazioni possibili Collider-Rigidbody:

- **Static Collider:** Collider (non Trigger) con nessun Rigidbody.
- **Rigidbody Collider:** Collider (non Trigger) con un Rigidbody (non Kinematic).
- **Kinematic Rigidbody Collider:** Collider (non Trigger) con un Rigidbody configurato come Kinematic.
- **Static Trigger Collider:** Trigger Collider con nessun Rigidbody.
- **Rigidbody Trigger Collider:** Trigger Collider con un Rigidbody (non Kinematic).
- **Kinematic Rigidbody Trigger Collider:** Trigger Collider con un Rigidbody configurato come Kinematic.
- **Pure Rigidbody:** Rigidbody (Kinematic o non) senza Collider (Trigger o non).

Vedi: docs.unity3d.com/manual/collidersoverview

COMPONENTI: CAMERAS

Il componente Camera agisce come una videocamera reale.

Dovrai sempre avere almeno 1 camera nella scena (“taggata” **MainCamera**).

E’ possibile avere camere multiple in una scena (e.g. split-screen, picture-in-picture, etc.).

Viewing Frustum: Il volume 3D visto dalla camera è chiamato “viewing frustum”: la sua forma dipende dal tipo di proiezione voluta, e la sua dimensione dipende dalla configurazione della camera.

Viewport: L’area della finestra su cui la camera disegna è chiamata “viewport”: ha sempre forma rettangolare, ed è limitato dalla dimensione della finestra.

Vedi: docs.unity3d.com/manual/camerasoverview

Vedi: docs.unity3d.com/manual/class-camera

COMPONENTI: CAMERAS (2)

Queste sono le proprie' piu' significative del componente Camera:

- **Clear Flags:** Modalita' pulizia schermo prima che la camera inizi a disegnare.
- **Culling Mask:** Selezione di layers considerati dalla camera durante il disegno.
- **Projection:** La proiezione utilizzata dalla camera (prospettica o ortografica).
- **Clipping Planes:** Distance dalla camera per iniziare e finire il disegno.
- **Viewport Rect:** Area dello schermo/window dove la camera disegna.
- **Depth:** Ordine di disegno della camera in caso di camere multiple.
- **Target Texture:** Settaggio per disegnare su immagine invece che a schermo.
- **Target Display:** Settaggio per selezionare lo schermo su cui disegnare.

Vedi: [docs.unity3d.com/manual/class-camera](https://docs.unity3d.com/manual/class-camera.html)

UNITY: PLAY MODE

Nell'Editor, i bottoni centrali della Toolbar (Play, Pause, e Step) controllano l'esecuzione della app all'interno dell'Editor:

- **Play:** Esegue la app dall'Editor, consentendo l'ispezione della app come in versione eseguibile. Questa esecuzione dall'Editor viene chiamata Play Mode.
- **Pause:** Mette in pausa l'esecuzione in Play Mode, consentendo l'ispezione dei vari elementi della app (di solito a fini di debugging).
- **Step:** Questo bottone e' attivo solo in Play Mode (o in pausa), e permette l'esecuzione solo del frame successivo mantenendo il Play Mode in pausa.

Nota: In Play Mode, ogni cambiamento effettuato all'interno dell'Editor e' considerato temporaneo (i.e., fatto a scopi di testing) e quindi annullato all'uscita dal Play Mode.

Vedi: docs.unity3d.com/manual/gameview

UNITY: BUILD

Il pannello **Build Settings** (menu > File > Build Settings...) include tutte le opzioni necessarie per compilare una app Unity e generare la sua versione esegibile (per la piattaforma selezionata tra quelle supportate). Queste sono le opzioni di build più importanti:

- **Scenes in Build:** Selezione/ordine delle scene incluse nella compilazione.
- **Platform:** La piattaforma obiettivo della build.
- **Build / Build and Run:** Lancia la compilazione, ed esegue la app.
- **Player Settings:** Pannello (menu > Edit > Project Settings > Player) per configurare ulteriori dettagli di una build.

Vedi: docs.unity3d.com/manual/publishingbuilds

Vedi: docs.unity3d.com/manual/buildsettings

Vedi: docs.unity3d.com/manual/class-playersettings

PARTE 1B: SCRIPTING IN UNITY

"Scripting is an essential ingredient in all applications you make in Unity."

UNITY TECHNOLOGIES

Vedi: docs.unity3d.com/manual/scriptingsection

SCRIPTING: CONFIGURAZIONE IDE

IDE: Un integrated development environment (IDE) e' una applicazione che fornisce funzionalita' per facilitare la programmazione e lo sviluppo software in generale.

Unity supporta i seguenti IDEs:

- Microsoft Visual Studio (default)
- Microsoft Visual Studio Code
- JetBrains Rider

Configura il IDE utilizzato dallo Unity Editor con:

menu > Edit > Preferences > External Tools > External Script Editor

Vedi: docs.unity3d.com/manual/scriptingtoolsides

SCRIPTING: CONFIGURAZIONE IDE (2)

Unity integra Microsoft Visual Studio attraverso il **Code Editor Package** per Visual Studio. Questo package e' pre-installato/incluso in ogni installazione di Unity.

Controlla/aggiorna questo package dallo Unity Editor con:

menu > Window > Package Manager > Visual Studio Code Editor

menu > Window > Package Manager > Visual Studio Editor

Unity usa il compilatore C# di Visual Studio per compilare gli scripts.

Quando usi il package Visual Studio Editor insieme a Visual Studio, sia lo Unity Editor che Visual Studio mostrano gli errori (warnings e messaggi) relativi agli scripts del progetto.

Vedi: docs.unity3d.com/manual/visualstudiotegration

SCRIPTING: SCRIPT-COMPONENTS E C#

Inheritance: Solo la single-inheritance e' supportata in C# (multiple-inheritance no).
Ovvero: una classe in C# puo' derivare solo da 1 altra classe.

La classe che implementa uno script-component deve derivare da **MonoBehavior** (direttamente o indirettamente).

Garbage Collection: C# usa una garbage collection del tipo tracing/generational (3 gens).
Il GC "pulisce" oggetti in base al loro tempo-vita e dimensione.

Constructor: La classe che implementa uno script-component non deve implementare **alcun costruttore** (questo task e' responsabilita' dello Unity Editor).

Naming: Lo script e la classe che lo script implementa devono avere lo stesso nome, capitalizzazione inclusa (requisito di C# scripting in Unity).

SCRIPTING: PRIMO SCRIPT-COMPONENT

In Unity, gli script C# sono solitamente creati all'interno dell'editor.

Per creare un nuovo script C# usa:

Project panel > top-left menu (+) > C# Script

Project panel > right-mouse-click su Assets folder/subfolder > Create > C# Script
menu > Assets > Create > C# Script

Il nuovo script viene **creato nella cartella/sottocartella di Assets selezionata.**

Il nome del nuovo script puo' essere cambiato immediatamente alla sua creazione.

E' buona prassi **nominare lo script in modo appropriato immediatamente** (perche' la classe interna verra' creata automaticamente con lo stesso nome).

Vedi: docs.unity3d.com/manual/creatingandusingscripts

SCRIPTING: PRIMO SCRIPT-COMPONENT (2)

Se doppio-clicchi (double-left-mouse-click) su uno script C# nell'editor, Unity apre lo script nel IDE configurato nelle preferenze.

Un nuovo script C#, non rinominato, contiene questo codice (generato automaticamente):

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class NewBehaviourScript : MonoBehaviour {
    // Start is called before the first frame update
    void Start() {}
    // Update is called once per frame
    void Update() {}
}
```

Vedi: docs.unity3d.com/manual/creatingandusingscripts

SCRIPTING: PRIMO SCRIPT-COMPONENT (3)

Nello script precedente:

- 3 direttive `using` di C#: le prime 2 per usare namespaces C#, l'ultima per usare il namespace principale di Unity.
- Dichiarazione della classe: stesso nome del file di script, con derivazione dalla classe Unity MonoBehaviour (che lo rende uno script-component). La derivazione completa è: MonoBehaviour - Behaviour - Component - Object.
- 2 funzioni definite all'interno della classe: entrambe sono funzioni-evento, ovvero funzioni chiamate automaticamente da Unity in risposta ad eventi rilevati dal game engine.

Vedi: docs.unity3d.com/manual/creatingandusingscripts

Vedi: learn.microsoft.com/dotnet/csharp/language-reference/keywords/using-directive

Vedi: learn.microsoft.com/dotnet/csharp/language-reference/keywords/namespace

Vedi: docs.unity3d.com/scriptreference/monobehaviour

SCRIPTING: PRIMO SCRIPT-COMPONENT (4)

Alcune note relative all'anatomia dello script appena discussa:

- Questo script/classe rappresenta uno script-component a causa della derivazione da MonoBehaviour, ma si possono programmare anche classi standard come in OOP (solo che non potranno essere usate come components).
- Negli script-component deve essere evitata la programmazione di costruttori perché l'instanziazione degli oggetti-components è compito dell'editor. Non farlo (i.e., implementare costruttori in script-components) interferisce con il normale funzionamento di Unity e può creare problemi seri nel progetto.
- Esistono molte funzioni-evento che si possono utilizzare: ognuna legata ad un preciso evento, con o senza un determinato scheduling, e con o senza inputs, ma di solito senza alcun output.

Vedi: docs.unity3d.com/manual/creatingandusingscripts

SCRIPTING: PRIMO SCRIPT-COMPONENT (5)

Uno script-component implementa un componente personalizzato.

Un oggetto/istanza della classe dello script-component **esiste solo se lo script-component viene attaccato ad un GameObject**.

Il codice di uno script-component viene eseguito solo se lo script-component viene attaccato ad un GameObject.

Il codice degli script* viene eseguito quando si lancia la app (in Play Mode o da eseguibili).

Il codice degli script viene compilato appena lo script viene salvato.

Vedi: docs.unity3d.com/manual/creatingandusingscripts

SCRIPTING: USARE LO INSPECTOR

Nell'implementare uno script-component possiamo decidere di esporre alcune variabili della classe nel pannello Inspector al fine di facilitarne l'ispezione e la modifica.

Le variabili public e non-static sono automaticamente visualizzate nel pannello Inspector.

Nota: Lo Inspector puo' cambiare i nomi delle variabili visualizzate a fini di leggibilita'.

Nota: Lo Inspector cambia la visualizzazione delle variabili a seconda del loro tipo.

Nota: Lo Unity Editor permette di modificare le variabili di uno script-component usando il pannello Inspector in Play Mode per facilitare il testing e debugging. Queste modifiche (fatte in Play Mode) sono annullate appena si esce dal Play Mode.

Vedi: docs.unity3d.com/manual/variablesandtheinspector

SCRIPTING: CICLO-VITA DI SCRIPTS

Ordine di Esecuzione delle Funzioni Evento:

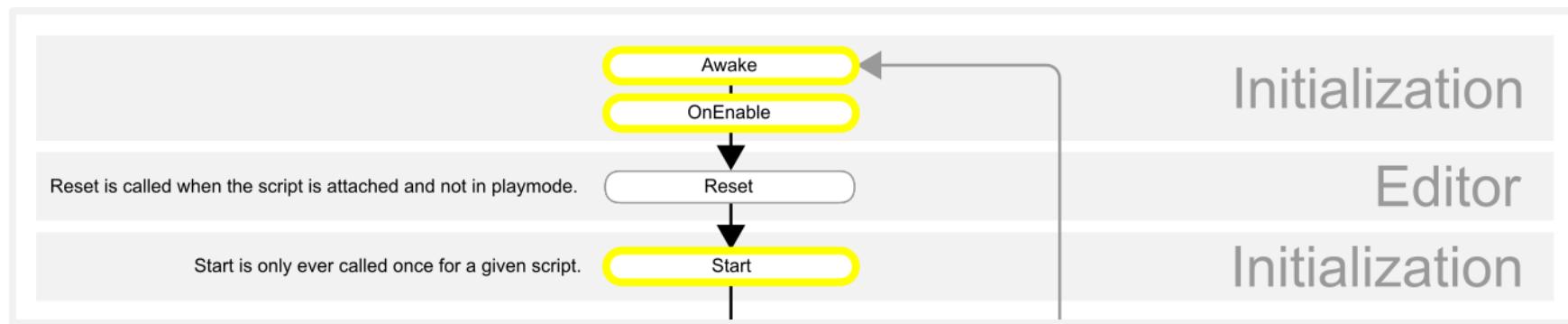
L'esecuzione di uno script-component in Unity consiste principalmente nell'esecuzione di molteplici funzioni-evento in un ordine predeterminato.

Il diagramma di flusso seguente illustra l'ordine di esecuzione delle funzioni-evento, raggruppando le funzioni-evento per scopo (quelle piu' usate sono evidenziate in giallo).

Nota: Alcune funzioni-evento sono eseguite con uno scheduling predeterminato, altre sono attivate in risposta a specifici eventi relativi a input, fisica, rendering etc.

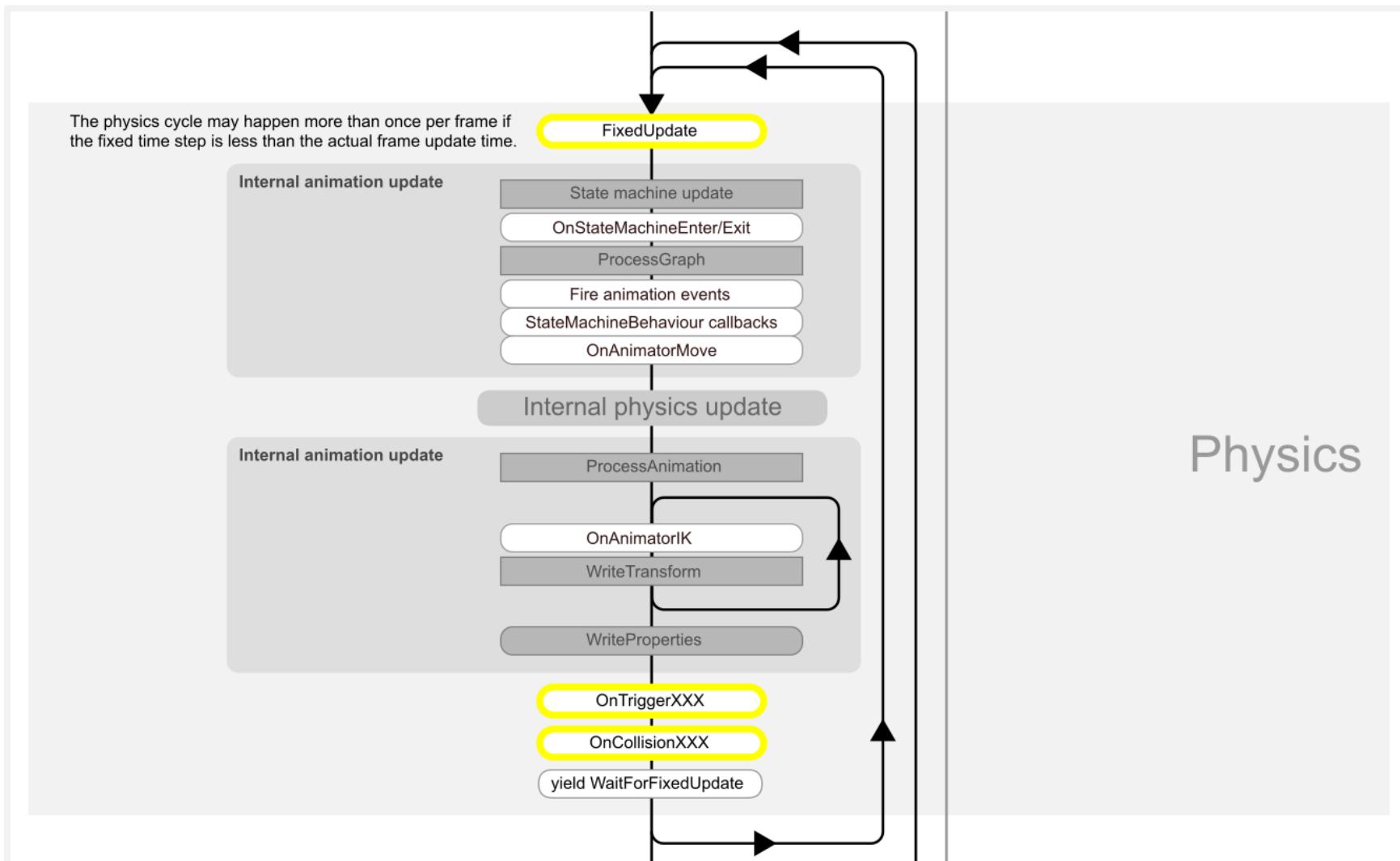
Vedi: docs.unity3d.com/manual/executionorder

SCRIPTING: CICLO-VITA DI SCRIPTS (2)



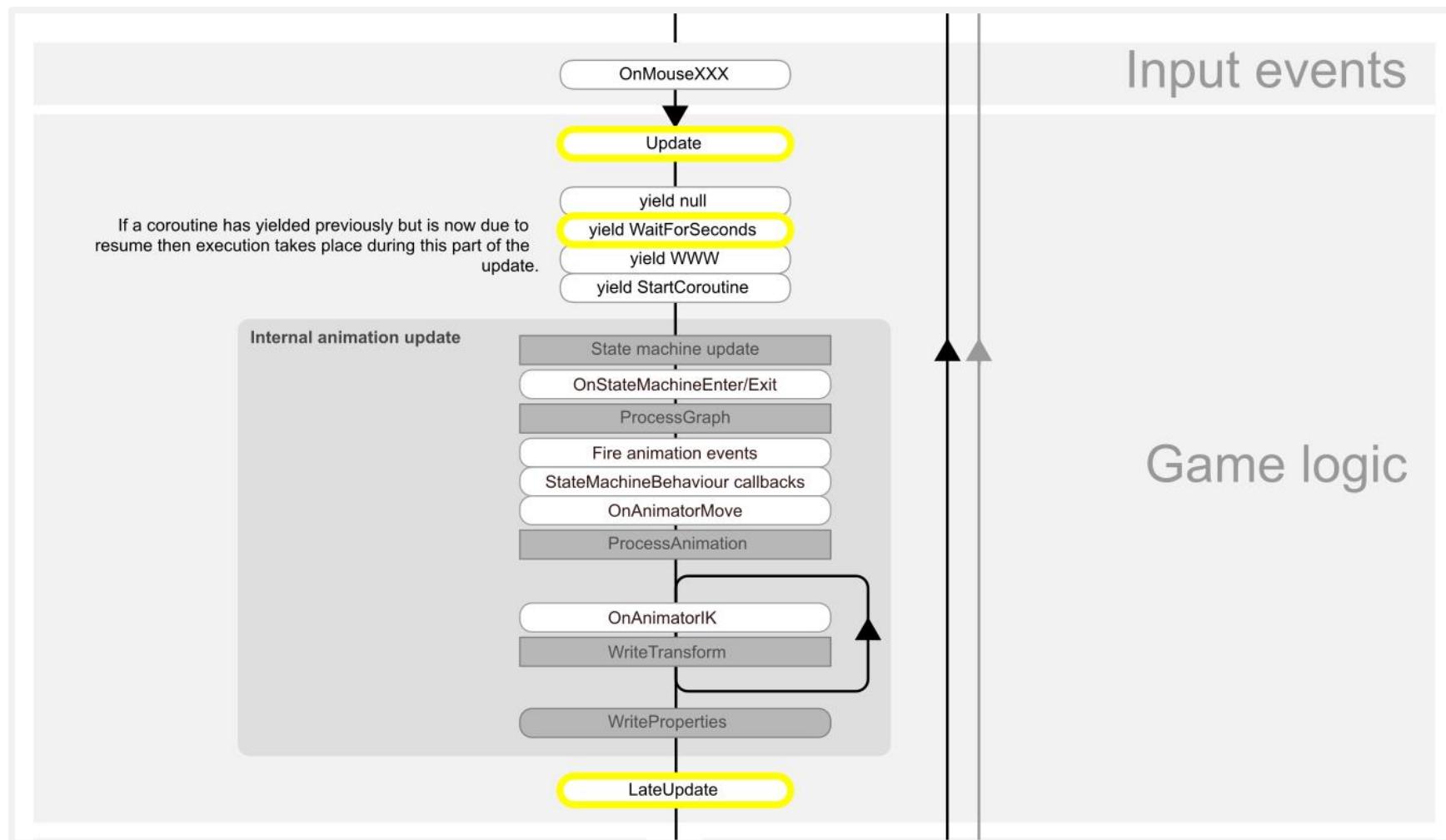
Vedi: docs.unity3d.com/manual/executionorder

SCRIPTING: CICLO-VITA DI SCRIPTS (3)



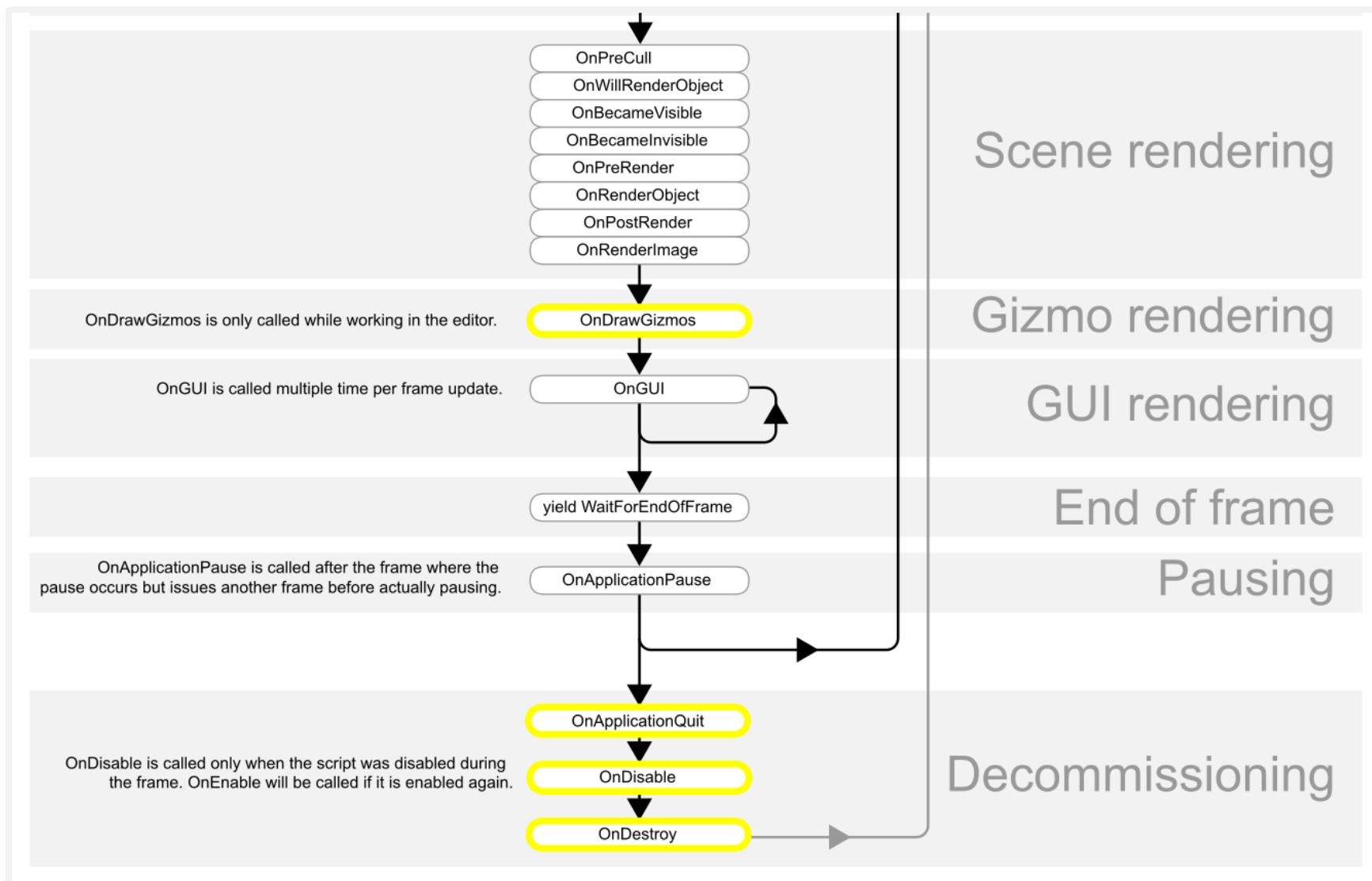
Vedi: docs.unity3d.com/manual/executionorder

SCRIPTING: CICLO-VITA DI SCRIPTS (4)



Vedi: docs.unity3d.com/manual/executionorder

SCRIPTING: CICLO-VITA DI SCRIPTS (5)



Vedi: docs.unity3d.com/manual/executionorder

SCRIPTING: PRIMO CARICAMENTO SCENA

Queste funzioni-evento vengono attivate quando una scena viene caricata (ed eseguite una sola volta per ciascun GameObject nella scena):

- **Awake:** Funzione-evento viene eseguita sempre* prima di qualunque funzione-evento Start (esecuzione differita per oggetti inattivi al caricamento).
- **OnEnable:** Funzione-evento eseguita appena un oggetto viene attivato (enabled), quando una scena viene caricata o quando un oggetto viene creato.
- **Start:** Funzione-evento chiamata solo 1 volta prima* della prima chiamata della funzione-evento Update (ma solo se lo script è attivo, enabled).
- * Eccezioni a questo ordine di esecuzione per oggetti creati a runtime.

Vedi: docs.unity3d.com/manual/executionorder

SCRIPTING: ORDINE DELLE UPDATES

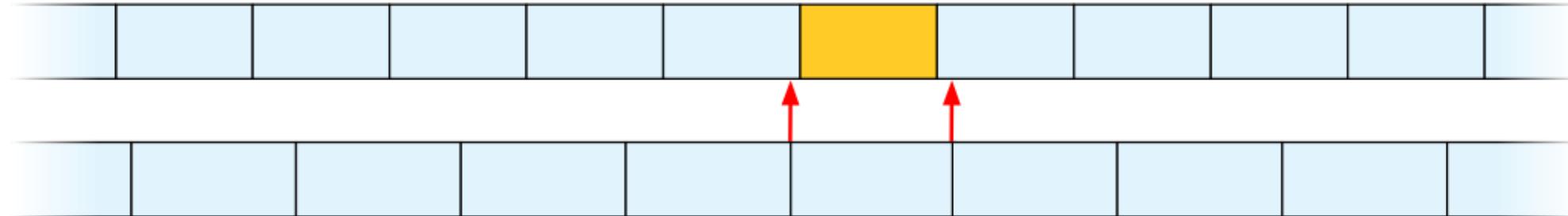
Queste funzioni-evento sono tutte eseguite con scheduling “predeterminato”:

- **FixedUpdate:** Funzione-evento eseguita alla frequenza fissa configurata per il physics engine e la collision detection (default 50 Hz), che dovrebbe essere utilizzata per codice che riguarda la fisica interattiva.
- **Update:** Funzione-evento eseguita 1 volta per ogni visualizzazione a video (frame), quindi con la stessa frequenza (variabile) del frame-rate, che di solito e’ utilizzata per gran parte del codice di gestione della logica e degli input.
- **LateUpdate:** Funzione-evento eseguita 1 volta per ogni frame, ma dopo l’esecuzione di ogni funzione-evento Update.
- * L’ordine di esecuzione tra medesime funzioni-evento di script differenti e’ “non-deterministico” (lo sviluppatore non dovrebbe considerarle ordinate).

Vedi: docs.unity3d.com/manual/executionorder

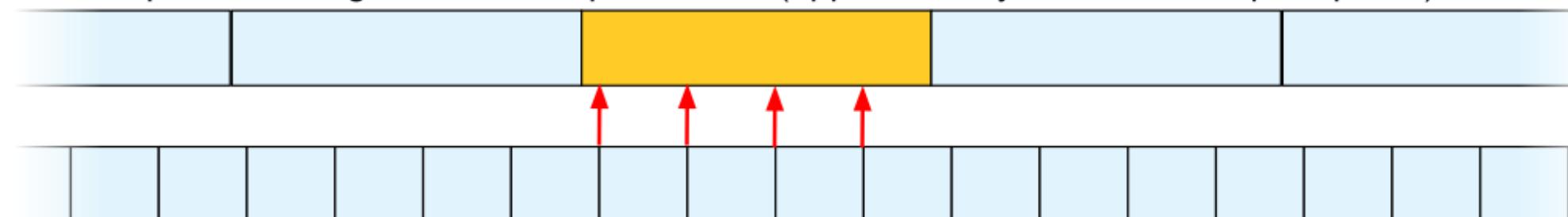
FONDAMENTALI: UPDATE RATES

Update running at 60 frames per second (approximately 0.01666 seconds per update)



FixedUpdate running at 50 updates per second (0.02 seconds per update)

Update running at 25 frames per second (approximately 0.04 seconds per update)



FixedUpdate running at 100 updates per second (0.01 seconds per update)

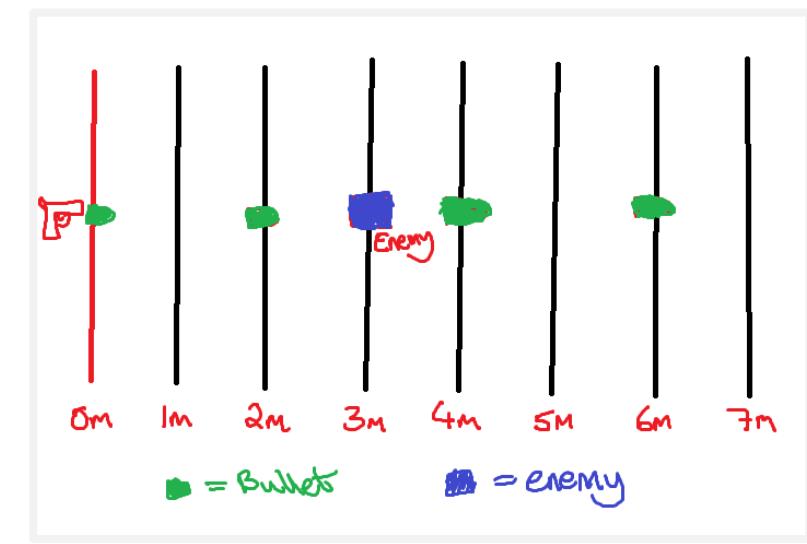
Vedi: docs.unity3d.com/manual/timeframemanagement

FONDAMENTALI: UPDATE RATES (2)

La frequenza di rilevamento delle collisioni e' la stessa degli aggiornamenti della fisica.
Per configurare la frequenza di collision-detection/physics-updates (default 50 Hz):

menu > Edit > Project Settings > Time > Fixed Timestep

La natura discreta degli aggiornamenti della fisica (discrete physics simulation) e della collision detection puo' risultare in situazioni limite (vedi figura) in cui alcune collisioni non vengono rilevate (fast-moving objects vs very-small colliders).



Vedi: docs.unity3d.com/manual/collidersoverview

Vedi: docs.unity3d.com/manual/timeframemanagement

Vedi: answers.unity.com/questions/1093106/bulletprojectile-collision-question

SCRIPTING: DECOMMISSIONE

Queste funzioni-evento sono relative alla terminazione/disabilitazione di elementi della app:

- **OnDestroy**: Funzione-evento chiamata al termine dell'ultimo frame di esistenza di un oggetto (causa distruzione oggetto o chiusura scena).
- **OnApplicationQuit**: Funzione-evento chiamata su tutti gli oggetti appena prima della chiusura della applicazione (in Play Mode nell'editor viene chiamata appena prima della terminazione del Play Mode).
- **OnDisable**: Funzione-evento chiamata quando lo script-component viene disattivato o diventa inattivo.

Vedi: docs.unity3d.com/manual/executionorder

SCRIPTING: ALTRE FUNZIONI-EVENTO

Questa e' una breve lista di altre funzioni-evento usate comunemente:

- **OnTriggerXXX:** Funzione-evento chiamata in risposta ad una collisione di un Trigger Collider, e' disponibile in 3 versioni per l'inizio (OnTriggerEnter), durante (OnTriggerStay), e il termine (OnTriggerExit) della collisione.
- **OnCollisionXXX:** Funzione-evento chiamata in risposta ad una collisione di un Rigidbody Collider, e' disponibile in 3 versioni per l'inizio (OnCollisionEnter), durante (OnCollisionStay), e il termine (OnCollisionExit) della collisione.
- **OnDrawGizmos/OnDrawGizmosSelected:** Funzione-evento usata per disegnare gizmos nella Scene View dell'editor e facilitare cosi' lo sviluppo

Vedi: docs.unity3d.com/manual/executionorder

Vedi: docs.unity3d.com/scriptreference/monobehaviour.ontriggerenter

Vedi: docs.unity3d.com/scriptreference/monobehaviour.oncollisionenter

Vedi: docs.unity3d.com/scriptreference/monobehaviour.ondrawgizmos

SCRIPTING: COLLISION ACTION MATRIX

| Collision detection occurs and messages are sent upon collision | | | | | | |
|---|-----------------|--------------------|------------------------------|-------------------------|----------------------------|--------------------------------------|
| | Static Collider | Rigidbody Collider | Kinematic Rigidbody Collider | Static Trigger Collider | Rigidbody Trigger Collider | Kinematic Rigidbody Trigger Collider |
| Static Collider | | Y | | | | |
| Rigidbody Collider | Y | Y | Y | | | |
| Kinematic Rigidbody Collider | | Y | | | | |
| Static Trigger Collider | | | | | | |
| Rigidbody Trigger Collider | | | | | | |
| Kinematic Rigidbody Trigger Collider | | | | | | |

| Trigger messages are sent upon collision | | | | | | |
|--|-----------------|--------------------|------------------------------|-------------------------|----------------------------|--------------------------------------|
| | Static Collider | Rigidbody Collider | Kinematic Rigidbody Collider | Static Trigger Collider | Rigidbody Trigger Collider | Kinematic Rigidbody Trigger Collider |
| Static Collider | | | | | Y | Y |
| Rigidbody Collider | | | | Y | Y | Y |
| Kinematic Rigidbody Collider | | | | Y | Y | Y |
| Static Trigger Collider | | Y | Y | | Y | Y |
| Rigidbody Trigger Collider | Y | Y | Y | Y | Y | Y |
| Kinematic Rigidbody Trigger Collider | Y | Y | Y | Y | Y | Y |

Vedi: docs.unity3d.com/manual/collidersoverview

SCRIPTING: INTERAZIONI OBJECT-COMPONENT

Questa e' una breve lista di esempi per di interazione oggetti/componenti:

- Usare **reference public** in uno script-component configurabile manualmente.
- Fare una ricerca per tag usando la funzione **FindWithTag**.
- Fare una ricerca per nome usando la funzione **Find**.
- Fare una ricerca per tipo usando la funzione **FindObjectOfType**.
- Sfruttare le collisioni usando le funzioni **OnTriggerXXX** o **OnCollisionXXX**.
- Sfruttare le collisioni/raycasting usando la funzione **Raycast**.
- Utilizzare una **reference static** configurata manualmente o automaticamente.

SCRIPTING: CLASSI UNITY UTILI

Questa e' una breve lista di alcune classi Unity comunemente usate (continua):

- Application: (static class) docs.unity3d.com/scriptreference/application
- AudioSource: docs.unity3d.com/scriptreference/audiosource
- Bounds: docs.unity3d.com/scriptreference/bounds
- Camera: docs.unity3d.com/scriptreference/camera
- Collider: docs.unity3d.com/scriptreference/collider
- Collision: docs.unity3d.com/scriptreference/collision
- Color: docs.unity3d.com/scriptreference/color
- Debug: (static class) docs.unity3d.com/scriptreference/debug
- Display: (static class) docs.unity3d.com/scriptreference/display

SCRIPTING: CLASSI UNITY UTILI (2)

Questa e' una breve lista di alcune classi Unity comunemente usate (continua):

- **GameObject:** [docs.unity3d.com/scriptreference/gameobject](https://docs.unity3d.com/ScriptReference/GameObject.html)
- **Gizmos:** (static class) [docs.unity3d.com/scriptreference/gizmos](https://docs.unity3d.com/ScriptReference/Gizmos.html)
- **GUI:** (static class) [docs.unity3d.com/scriptreference/gui](https://docs.unity3d.com/ScriptReference/Gui.html)
- **Input:** (static class) [docs.unity3d.com/scriptreference/input](https://docs.unity3d.com/ScriptReference/Input.html)
- **Light:** [docs.unity3d.com/scriptreference/light](https://docs.unity3d.com/ScriptReference/Light.html)
- **Material:** [docs.unity3d.com/scriptreference/material](https://docs.unity3d.com/ScriptReference/Material.html)
- **Mathf:** (static class) [docs.unity3d.com/scriptreference/mathf](https://docs.unity3d.com/ScriptReference/Mathf.html)
- **Matrix4x4:** [docs.unity3d.com/scriptreference/matrix4x4](https://docs.unity3d.com/ScriptReference/Matrix4x4.html)
- **Mesh:** [docs.unity3d.com/scriptreference/mesh](https://docs.unity3d.com/ScriptReference/Mesh.html)

SCRIPTING: CLASSI UNITY UTILI (3)

Questa e' una breve lista di alcune classi Unity comunemente usate (continua):

- MeshFilter: [docs.unity3d.com/scriptreference/meshfilter](https://docs.unity3d.com/ScriptReference/MeshFilter.html)
- MeshRenderer: [docs.unity3d.com/scriptreference/meshrenderer](https://docs.unity3d.com/ScriptReference/MeshRenderer.html)
- MonoBehaviour: [docs.unity3d.com/scriptReference/monobehaviour](https://docs.unity3d.com/ScriptReference/MonoBehaviour.html)
- ParticleSystem: [docs.unity3d.com/scriptreference/particlesystem](https://docs.unity3d.com/ScriptReference/ParticleSystem.html)
- Physics: (static class) [docs.unity3d.com/scriptreference/physics](https://docs.unity3d.com/ScriptReference/Physics.html)
- PlayerPrefs: (static class) [docs.unity3d.com/scriptreference/playerprefs](https://docs.unity3d.com/ScriptReference/PlayerPrefs.html)
- Pose: [docs.unity3d.com/scriptreference/pose](https://docs.unity3d.com/ScriptReference/Pose.html)
- Quaternion: [docs.unity3d.com/scriptreference/quaternion](https://docs.unity3d.com/ScriptReference/Quaternion.html)
- Random: (static class) [docs.unity3d.com/scriptreference/random](https://docs.unity3d.com/ScriptReference/Random.html)

SCRIPTING: CLASSI UNITY UTILI (4)

Questa e' una breve lista di alcune classi Unity comunemente usate (continua):

- **Ray:** [docs.unity3d.com/scriptreference/ray](https://docs.unity3d.com/ScriptReference/Ray.html)
- **RaycastHit:** [docs.unity3d.com/scriptreference/raycasthit](https://docs.unity3d.com/ScriptReference/RaycastHit.html)
- **RectTransform:** [docs.unity3d.com/scriptreference/recttransform](https://docs.unity3d.com/ScriptReference/RectTransform.html)
- **Rigidbody:** [docs.unity3d.com/scriptreference/rigidbody](https://docs.unity3d.com/ScriptReference/Rigidbody.html)
- **Screen:** (static class) [docs.unity3d.com/scriptreference/screen](https://docs.unity3d.com/ScriptReference/Screen.html)
- **Texture2D:** [docs.unity3d.com/scriptreference/texture2d](https://docs.unity3d.com/ScriptReference/Texture2D.html)
- **Time:** (static class) [docs.unity3d.com/scriptreference/time](https://docs.unity3d.com/ScriptReference/Time.html)
- **Transform:** [docs.unity3d.com/scriptreference/transform](https://docs.unity3d.com/ScriptReference/Transform.html)
- **Vector3:** [docs.unity3d.com/scriptreference/vector3](https://docs.unity3d.com/ScriptReference/Vector3.html)

DEBUGGING: LA CONSOLE WINDOW

La finestra **Console** serve per visualizzare errori/warnings/messaggi generati dall'editor in fase di compilazione o esecuzione.

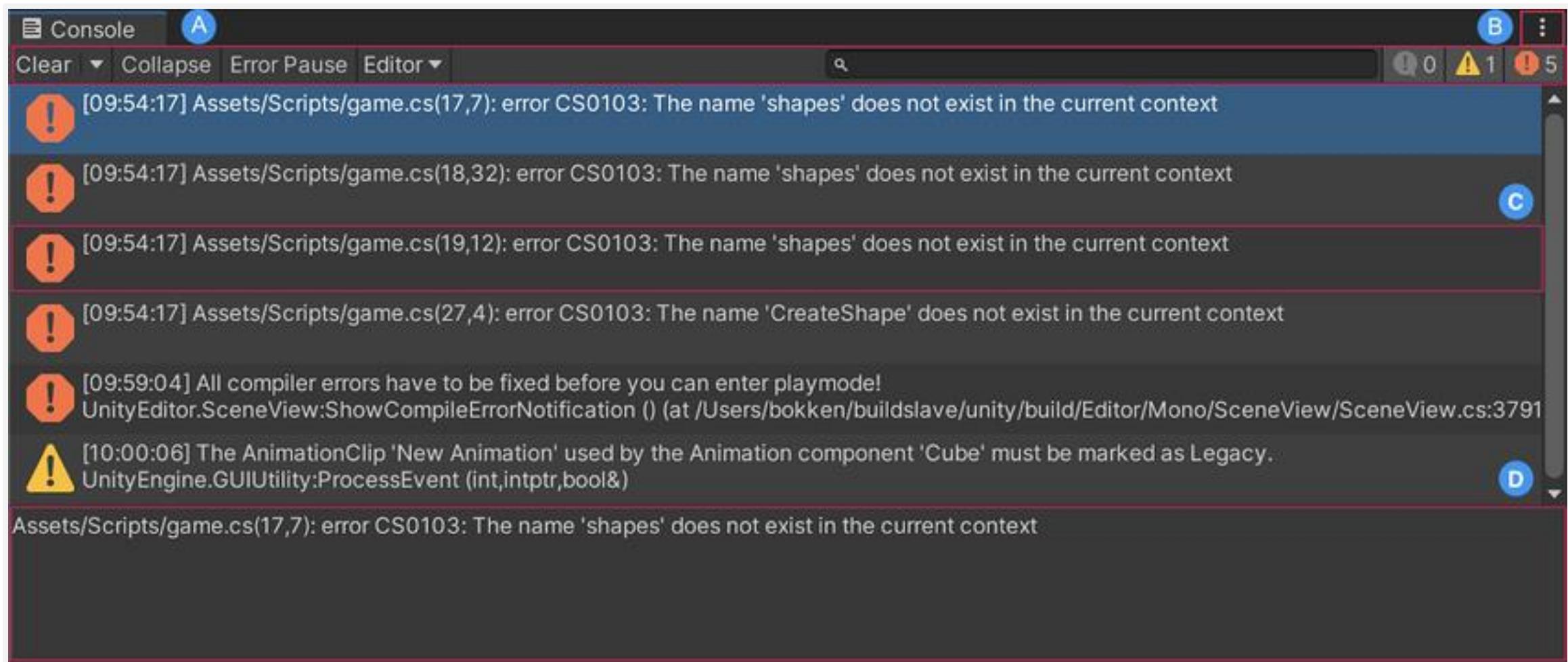
Queste informazioni sono fondamentali per il debugging del progetto.

Per facilitare il debugging usa la finestra **Console** insieme alla classe **Debug** che consente di programmare la scrittura di messaggi da codice (“print debugging”, vedi esempio).

```
public class NewBehaviourScript : MonoBehaviour {  
  
    void Start() {  
        Debug.Log("Il nome del GameObject e': " + this.gameObject);  
    }  
}
```

Vedi: docs.unity3d.com/manual/console

DEBUGGING: LA CONSOLE WINDOW (2)



Vedi: docs.unity3d.com/manual/console

DEBUGGING: DEBUG CLASS

La classe **Debug** (static class) fornisce funzionalita' base per il "print debugging" in modo da poter scrivere messaggi e valori di variabili nella finestra Console.

Queste sono alcune delle sue funzioni piu' usate:

- **Assert:** Direttiva assert come standard in altri linguaggi di programmazione.
- **Break:** Mette in pausa l'editor se in Play Mode.
- **DrawLine/DrawRay:** Disegna una linea/raggio nella Scene View.
- **Log:** Scrive un messaggio nella finestra Console.
- **.LogError:** Scrive un messaggio di errore nella finestra Console.
- **.LogWarning:** Scrive un messaggio di warning nella finestra Console.

Vedi: docs.unity3d.com/manual/managedcodedebugging

Vedi: docs.unity3d.com/scriptreference/debug

DEBUGGING: BREAKPOINTS

Per poter utilizzare breakpoints in Unity occorre:

- **Crea/Rimuovi Breakpoints in Visual Studio:** in Visual Studio...
double-mouse-left-click su margine sinistro della riga di codice
- **Attaccare Unity Editor a Visual Studio:** in Visual Studio...
menu > Debug > Attach Unity Debugger > seleziona un Unity Editor > OK
- **Attiva Play Mode nel Unity Editor:** nello Unity Editor...
premi il bottone Play del Unity Editor

Quindi il codice viene eseguito e l'esecuzione viene sospesa al primo breakpoint incontrato. In Visual Studio, premere Continue per continuare.

Vedi: docs.unity3d.com/manual/managedcodedebugging

Vedi: docs.unity3d.com/scriptreference/debug

PARTE 2A: LIVE SCRIPTING

Sessione live di sviluppo software in Unity di 1 ora circa.

L'obiettivo di questa sessione e' illustrare le fasi principali e la velocita' di sviluppo di un progetto Unity di taglia piccola, con team minimo (1 sviluppatore) e assets tutti disponibili.

In particolare, svilupperemo live un mini-game includendo:

- La **creazione** del progetto Unity da 0.
- Import di tutti gli **assets** (modelli 3D, audio clips, textures, fonts, etc.).
- Scripting della **"game logic"** dell'applicazione.
- Gestione di **scenes multiple**.
- Gestione di **dati persistenti**.
- Introduzione a **inputs** e **user interfaces** in Unity.
- Scripting di un **third-person controller** (TPS) semplice.
- Scripting di **raycasting** e **collision response**.

SCRIPTING: GAME LOGIC

Di solito il design della “game logic” di una applicazione in Unity integra questi aspetti:

- Il **design-pattern singleton** per il manager della logica.
- La **persistenza inter-scena** del manager della logica.
- Una forma di **finite-state machine** (FSM) nel manager della logica.

Vedi: en.wikipedia.org/wiki.singleton_pattern

Vedi: learn.unity.com/tutorial/implement-data-persistence-between-scenes

Vedi: en.wikipedia.org/wiki/finite-state_machine

SCRIPTING: GESTIONE SCENE

Come abbiamo già visto, le scene sono parti fondamentali di ogni progetto in Unity.

La gestione delle scene a runtime viene effettuata attraverso l'utilizzo negli script della classe **SceneManager** che offre molte funzionalità per caricare/distruggere scene.

Nota: Occorre ricordare che, per un corretto funzionamento della gestione di scene multiple, è essenziale configurare in modo appropriato le scene nei **Build Settings**.

Vedi: [docs.unity3d.com/scriptreference/scenemanagement.scenemanager](https://docs.unity3d.com/ScriptReference/Scenemanagement.SceneManager.html)

SCRIPTING: DATI PERSISTENTI

La gestione di dati persistenti in Unity include molteplici soluzioni a seconda della natura dei dati e delle ragioni per le quali è necessaria la loro persistenza:

- Dati (interni) persistenti inter-scena (`Object.DontDestroyOnLoad`).
- Dati (esterni) persistenti in file/registro di sistema (`PlayerPrefs`).
- Dati (esterni) persistenti in standard file (`JSON`, `JsonUtility`, `TXT`, etc.).
- Dati (esterni) persistenti online.

Vedi: blog.unity.com/persistent-data-how-to-save-your-game-states-and-settings

Vedi: learn.unity.com/tutorial/implement-data-persistence-between-scenes

Vedi: docs.unity3d.com/scriptreference/object.dontdestroyonload

Vedi: docs.unity3d.com/scriptreference/playerprefs

Vedi: docs.unity3d.com/scriptreference/jsonutility

FONDAMENTALI: INPUTS

La gestione degli inputs permette all'utente di controllare il software attraverso un dispositivo hardware (tastiera, mouse, etc.).

Unity supporta inputs da una varietà di dispositivi:

- Tastiera, mouse, joysticks, e gamepads/controllers.
- Touch screens, e IMUs di mobile devices.
- VR-AR controllers.

Unity supporta la gestione degli inputs attraverso 2 sistemi separati/divisi:

- **Input Manager:** modulo core/default, per la gestione base degli inputs.
- **Input System:** package aggiuntivo, per la gestione avanzata degli inputs

Vedi: docs.unity3d.com/manual/input

FONDAMENTALI: INPUTS (2)

Input Manager: La finestra Input Manager (menu > Project Settings > Input Manager) permette la definizione di assi (axes) e di azioni a loro associate.

Lo Input Manager utilizza i seguenti tipi di controlli:

- **Key:** si riferisce ad un qualunque tasto di una tastiera fisica.
- **Button:** si riferisce ad un qualunque bottone di un controller fisico.
- Un **virtual axis**: e' un asse virtuale mappato ad un controllo (bottone o tasto) e quando l'utente attiva quel controllo, l'asse riceve un valore in [-1, 1] (e lo sviluppatore puo' usare questo valore per la gestione inputs negli script).

Vedi: docs.unity3d.com/manual/input

Vedi: docs.unity3d.com/manual/class-inputmanager

FONDAMENTALI: INPUTS (3)

Input System: E' il package aggiuntivo per la gestione moderna/avanzata degli inputs, che offre un sistema flessibile che supporta ogni tipo di dispositivo di input.

Va installato dal **Package Manager** selezionando **Input System**.

Vedi: docs.unity3d.com/manual/input

Vedi: docs.unity3d.com/packages/com.unity.inputsystem

FONDAMENTALI: USER INTERFACES

Unity fornisce 3 sistemi di sviluppo di user interfaces (UIs):

- **UI Toolkit:** sistema UI più recente (raccomandato per progetti futuri).
- **The Unity UI package (uGUI):** sistema UI basato su GameObjects (standard).
- **Immediate Mode Graphical User Interface (IMGUI):** sistema UI basato su script (funzione-evento OnGUI) raccomandato solo per testing e rapid prototyping.

Vedi: docs.unity3d.com/manual/uitoolkits

Vedi: docs.unity3d.com/manual/uielements

Vedi: docs.unity3d.com/packages/com.unity.ugui

Vedi: docs.unity3d.com/manual/guiscriptingguide

PARTE 2B: INTRO A VR

Virtual Reality (VR): Tecnologia HW e SW in grado di fornire esperienze realistiche in modo artificiale, sfruttando ogni tecnologia multimediale disponibile.

La tecnologia VR e' in grado di fornire una esperienza immersiva totale che esclude il mondo fisico, e in cui l'utente e' "trasportato" in un ambiente realistico ma non reale che viene percepito come credibile dai sensi principali dell'utente.

Reality-Virtuality Continuum: Scala continua tra un ambiente/esperienza completamente virtuale (VR), e un ambiente/esperienza completamente reale (realta').

Include tutte le possibili variazioni e composizioni di contenuto reale e virtuale, ad esempio: realta', AR, "augmented virtuality", VR, etc.

TERMINOLOGIA: VR

Il termine "realta' virtuale" fu coniato da Jaron Lanier (VPL Research) nel 1987.

In figura gli occhiali VR "EyePhones", e i guanti VR "DataGloves" della VPL Research nel 1989.



TERMINOLOGIA: AR, MR, E XR

Augmented Reality (AR):

Tecnologia capace di aumentare una visualizzazione dal vivo del mondo reale aggiungendo (aumentandola con) elementi digitali.

Spesso gli elementi digitali aggiunti sono configurati considerando l'ambiente reale.

Mixed Reality (MR):

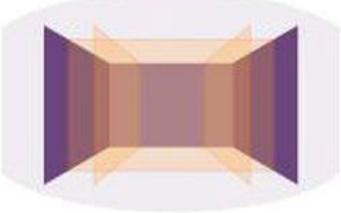
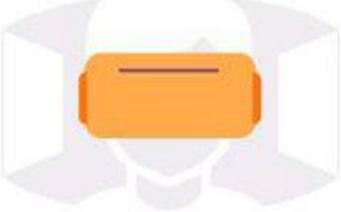
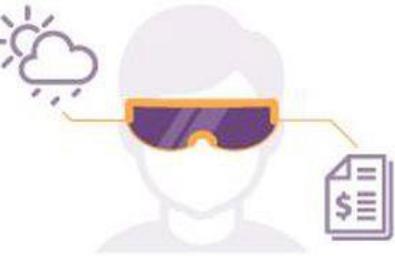
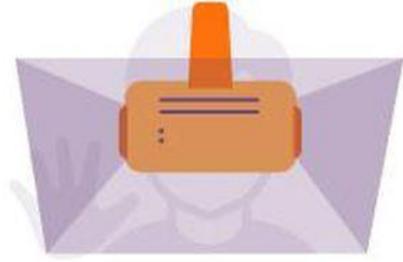
Tecnologia che combina elementi sia della VR che della AR, ma focalizzata sulle interazioni reale-virtuale.

L'obiettivo è abilitare parte del contenuto reale ad interagire con parte del contenuto digitale, e/o viceversa.

Extended Reality (XR):

Termine ombrello per VR/AR/MR e tutte le altre tecnologie che hanno come obiettivo l'estensione dei sensi dell'utente mischiando il virtuale con il reale in una esperienza unificata.

TERMINOLOGIA: AR, MR, E XR (2)

| VIRTUAL REALITY (VR) | AUGMENTED REALITY (AR) | MIXED REALITY (MR) |
|---|---|--|
| Fully artificial environment  | Virtual objects overlaid on real-world environment  | Virtual environment combined with real world  |
| Full immersion in virtual environment  | The real world enhanced with digital objects  | Interact with both the real world and the virtual environment  |

TERMINOLOGIA: IMMERSION E PRESENCE

Immersion: Il livello oggettivo di fedelta' sensoriale fornito da un sistema VR.
E' ottenuta attorniando l'utente con accurati stimoli artificiali (auditivi, visivi, tattili, olfattivi, o qualunque altro stimolo) che rendano l'esperienza virtuale realistica e credibile.

Presence: La sensazione soggettiva di essere presente in un ambiente virtuale.
E' ottenuta sfruttando sia dispositivi tecnologici che concetti di psicologia, "human factors", ergonomia, etc.

TERMINOLOGIA: STEREOPSIS

Stereopsis: La percezione della profondità generata dalla ricezione nel cervello (visual cortex) di stimoli visivi (coppie di immagini stereo) da entrambi gli occhi (visione binoculare).

IPD: La inter-pupillary distance (IPD) è la distanza tra i centri delle pupille dell'utente.

Anaglyphs: L'integrazione di ciascuna immagine di una coppia stereo in una singola figura, utilizzando filtri colore differenti (solitamente cromaticamente opposti; e.g., rosso/ciano, etc.) per l'immagine sinistra e destra della coppia stereo.

Stereo Rendering: Il processo di computer graphics per visualizzare contenuto 3D generando una coppia di immagini stereo in modo da ottenere l'illusione della percezione della profondità da parte dell'utente.

TERMINOLOGIA: HMDS E CAVES

HMD: Un head-mounted display (HMD) e' un dispositivo di visualizzazione progettato per essere indossato sulla testa. Include un visore e ottiche binoculari, e alcune inertia measurement units (IMUs) per il motion tracking.

OST-HMD: Un optical see-through HMD (OST-HMD) e' un HMD con un visore semitrasparente che permette all'utente di vedere sia il contenuto digitale disegnato sul visore che il contenuto reale parte della realta'.

VST-HMD: Un video see-through HMD (VST-HMD) e' un HMD che integra stereo camere allineate con le ottiche interne del HMD in modo da permettere la visualizzazione stereo del video streaming delle camere all'utente.

CAVE: Un cave automatic virtual environment (CAVE) consiste in una stanza in cui ogni parete e' uno schermo, e che nell'insieme costituisce un sistema per VR.

TERMINOLOGIA: COMFORT E SICKNESS

Physiological Comfort: Situazione in cui l'utente non sente conflitto nella stimolazione sensoriale (auditiva, visiva, etc.).

Una carenza in physiological comfort puo' risultare in affaticamento, nausea, etc.

Environmental Comfort: Comfort dovuto all'ambiente (reale o virtuale).

Una carenza in environmental comfort puo' risultare in claustrofobia, vertigini, etc.

VR Sickness: Malessere (affaticamento, malditesta, nausea, etc.) causato da conflitti sensoriali (auditivi, visivi, etc.) e dovuto al sistema e/o app VR usati dall'utente.

Simile come effetti e cause al mal d'auto, mal di mare, etc.

TERMINOLOGIA: AVATARS E LOCOMOTION

VR Avatar: La rappresentazione in VR dell'utente.

Codec Avatars: Inventati al Facebook Reality Labs (FRL), i Pixel Codec Avatars (PiCA) sono modelli 3D di facce, basati su AI e ottimizzati per la ricostruzione, computazione, e animazione.

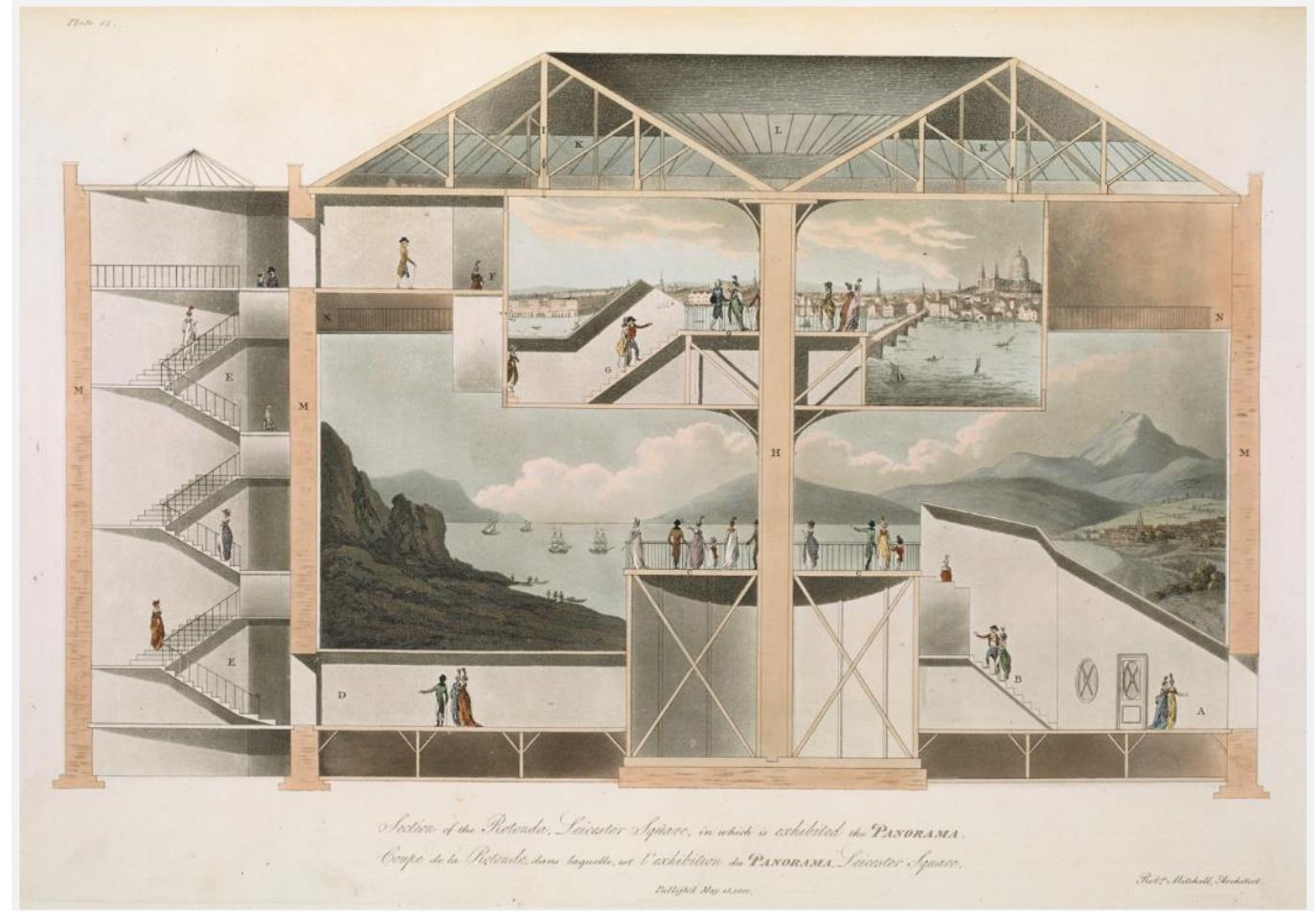
VR Locomotion: La capacità dell'utente di muoversi in VR. Si basa su tecnologia capace di tracciare i movimenti dell'utente, e su software capace di convertire questi dati in movimenti dell'avatar VR.

Motion Tracking: (aka positional tracking) Consiste nel rilevamento accurato della posizione e orientamento di alcuni elementi di un sistema VR (HMD, controllers, etc.) e/o dell'utente (sguardo, gambe, mani, dita, etc.).

STORIA: VR NEL 1800

Gran parte della tecnologia VR disponibile oggigiorno e' stata creata basandosi su idee originali che risalgono al 1800.

1801: Robert Barker progetta e costruisce la Leicester Square Rotunda a Londra, che include 2 dipinti a 360-gradi del panorama di Londra.



STORIA: VR NEL 1800 (2)

1812: Primo dipinto panoramico (360-gradi) progettato per occupare l'intero campo di vista (field of view, FOV) dello spettatore, "Battle of Borodino" by Franz Roubaud (1812).

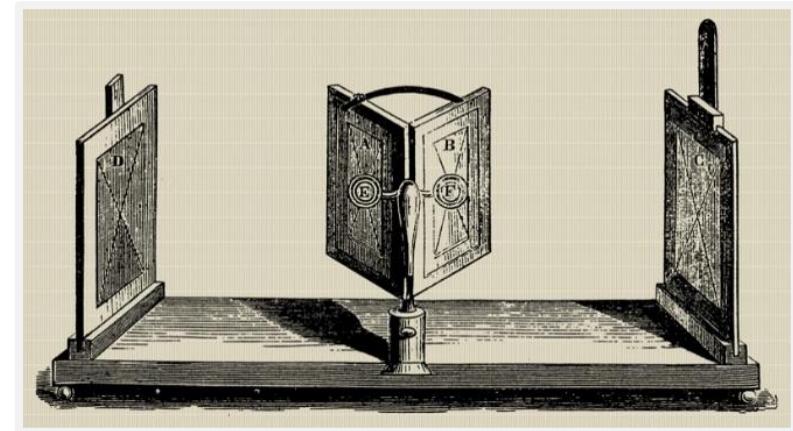


1855: Dipinto panoramico (360-gradi) progettato per occupare l'intero campo di vista (field of view, FOV) dello spettatore, "Siege of Sevastopol" by Franz Roubaud (1855).

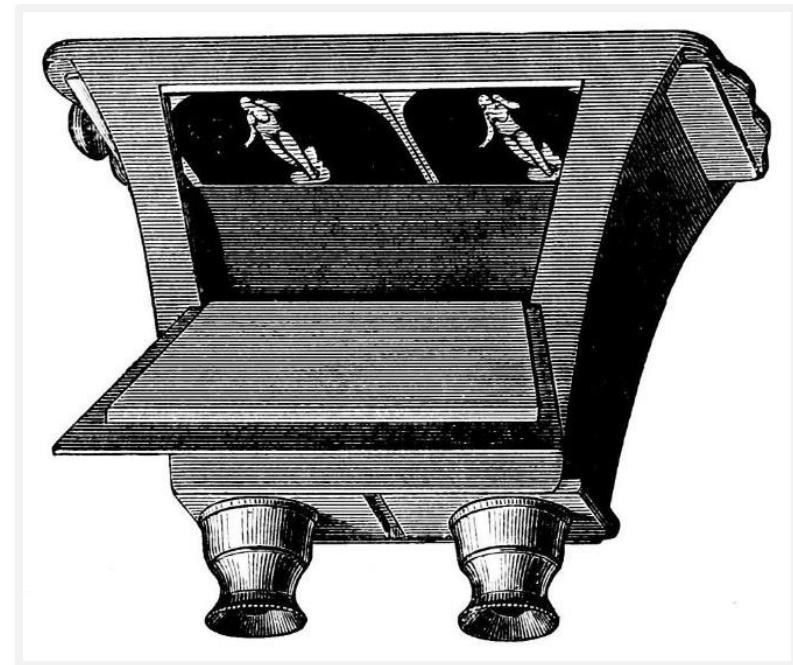


STORIA: VR NEL 1800 (3)

1838: Charles Wheatstone inventa il primo stereoscopio riflettente nel 1838.

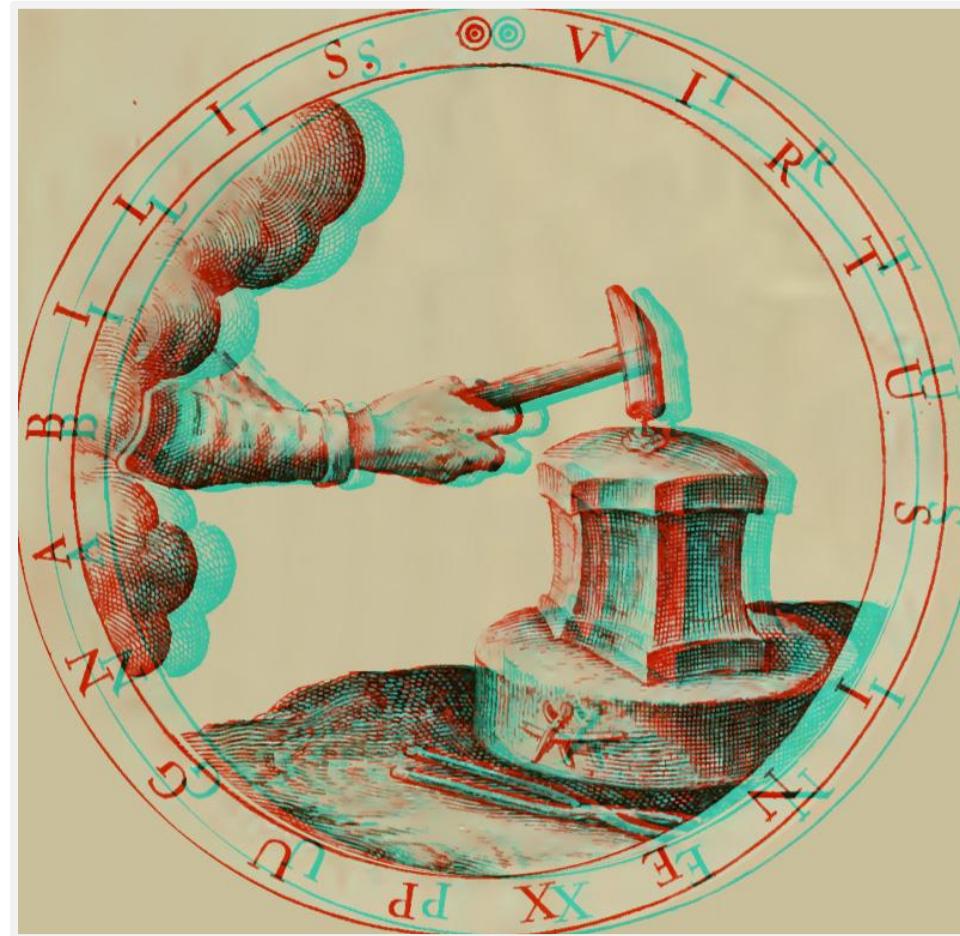


1849: David Brewster inventa il primo stereoscopio lenticolare nel 1849.



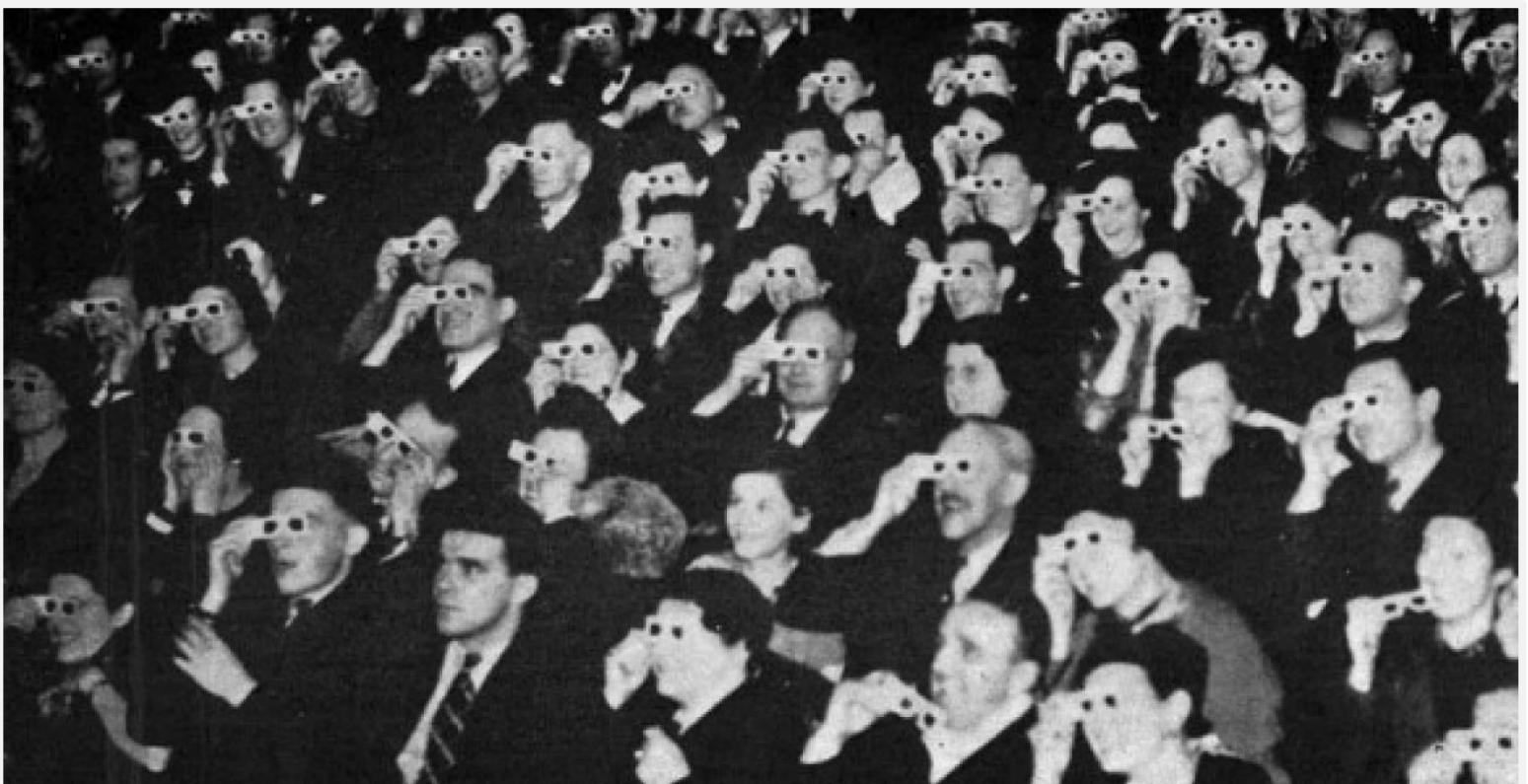
STORIA: VR NEL 1800 (4)

1853: Prima descrizione degli anaglifi da parte di W. Rollmann, utilizzando un disegno giallo/blu con occhiali-filtri rosso/blu (in figura un esempio di un anaglifo rosso/ciano).



STORIA: VR NEL 1900-1950

1922: Il primo film 3D commerciale "The Power of Love" viene distribuito fornendo occhiali per anaglifi.



STORIA: VR NEL 1900-1950 (2)

1929: Edward Link crea il primo simulatore di volo, il "Link Trainer", inventato nel 1929 e brevettato nel 1931.



STORIA: VR NEL 1900-1950 (3)

1935: Stanley G. Weinbaum scrive un racconto di science-fiction, "Pygmalion's Spectacles", inventando l'idea di occhiali VR che permettono all'utente di percepire un mondo virtuale attraverso stimoli visivi, auditivi, tattili, etc.



STORIA: VR NEL 1900-1950 (4)

1939: Lo stereoscopio View-Master viene brevettato nel 1939 da William Gruber, per poi essere fabbricato da Sawyers.



STORIA: VR NEL 1950-1975

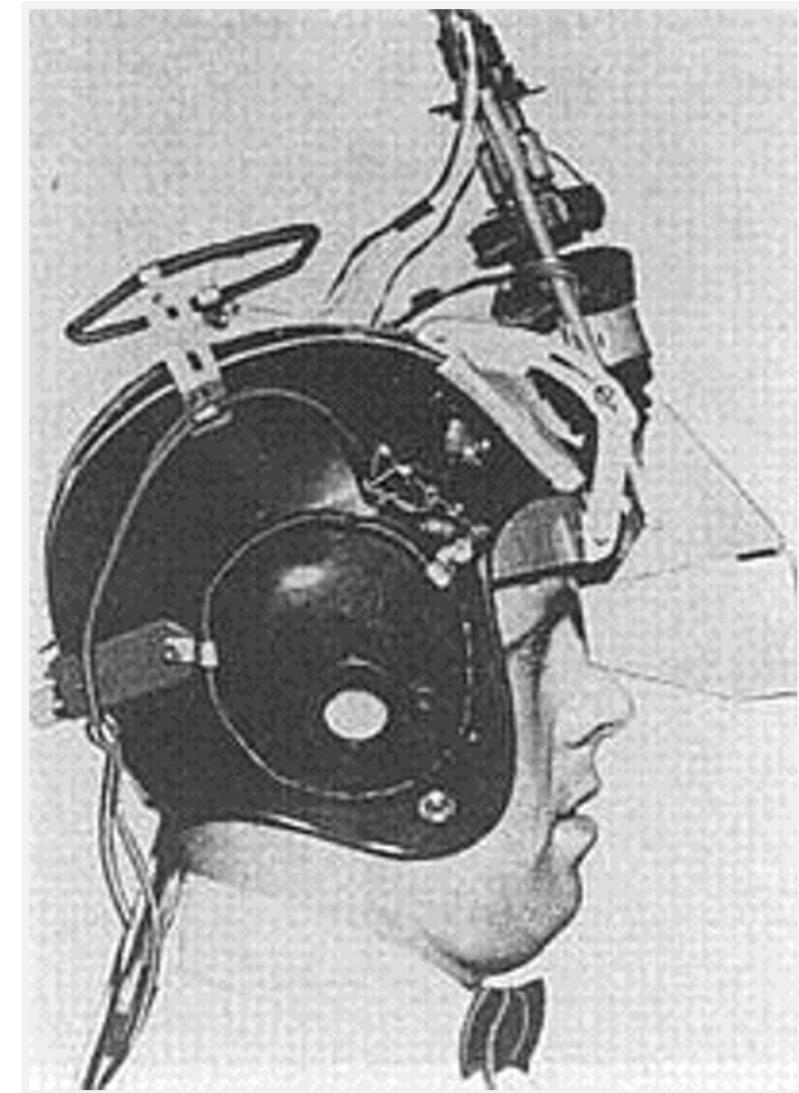
1956: Il sistema **Sensorama** (sinistra) viene inventato da Morton Heilig nel 1956, e consiste in una citta' virtuale visitabile su moto con visione stereo, audio, vibrazioni, e odori.

1960: Il sistema **Telesphere Mask** (destra) viene inventato da Morton Heilig nel 1960, per vedere film non-interattivi con visione stereo , audio, ma nessun motion tracking.



STORIA: VR NEL 1950-1975 (2)

1961: Il sistema Headsight, un HMD inventato dalla Philco Corporation nel 1961, era non-portatile ma includeva motion tracking magnetico (la visualizzazione in computer graphics ancora non era integrata).



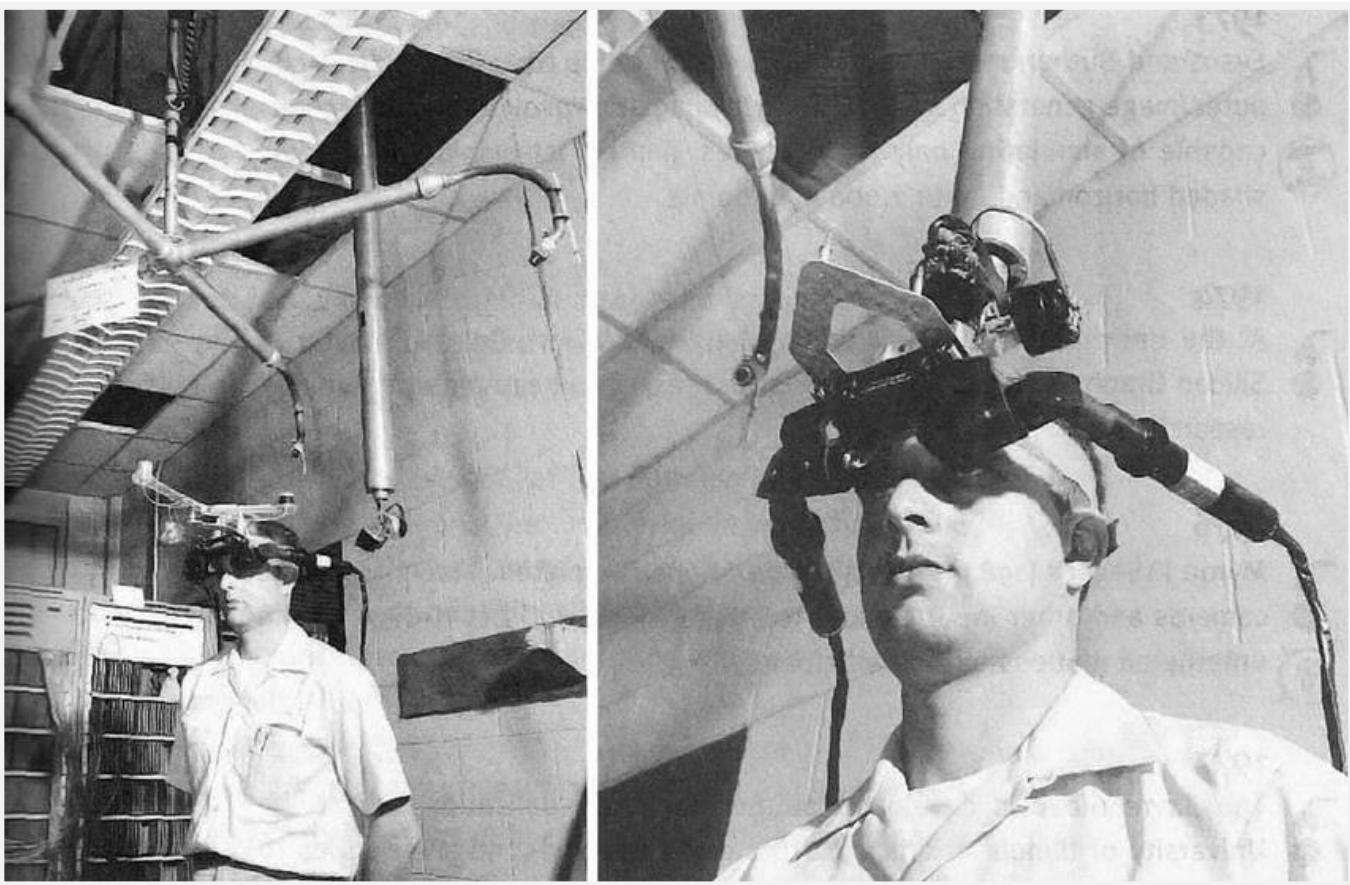
STORIA: VR NEL 1950-1975 (3)

1965: Il HMD "Ultimate Display"

viene inventato da Ivan Sutherland nel 1965 (non-portatile, connesso solo a camere stereo, e senza visualizzazione in computer graphics).

1968: Il sistema "Sword of

Damocles", un VR-AR HMD, viene inventato da Ivan Sutherland e Bob Sproull (non-portatile, include visualizzazione in computer graphics).



STORIA: VR NEL 1950-1975 (4)

1971: Il sistema "Visually Coupled Airborne Systems Simulator" (VCASS) viene co-sviluppato da Tom Furness e progettato per uso militare, ed è considerato il **primo simulatore di volo moderno**.



STORIA: VR NEL 1950-1975 (5)

1972: Un simulatore di volo completamente digitale (basato su computer graphics) viene sviluppato dalla General Electric (GE), e include 3-monitor in un setup a 180-gradi configurati in una cabina di pilotaggio per training.



STORIA: VR NEL 1950-1975 (6)

1969: "Artificial Reality" (in figura), un insieme di esperienze virtuali artistiche e interattive, viene ideata da Myron Krueger nel 1969.

1975: "VIDEOPLACE", il primo sistema VR interattivo, viene progettato da Myron Krueger nel 1975, e include visualizzazione computer-graphics, proiettori, camere, motion tracking, ma nessun HMD.



STORIA: VR NEL 1975-1985

1977: La "Aspen Movie Map", una mappa VR interattiva, viene progettata al MIT, permettendo agli utenti di visitare Aspen (CO, USA) in VR sfruttando video filmati da una auto (il sistema non include HMD).



STORIA: VR NEL 1975-1985 (2)

1979: L'elmetto VITAL, il primo VR HMD per l'industria (non prototipale), viene sviluppato alla McDonnell-Douglas, e include head tracking e visualizzazione computer graphics di base.



Vital trains more military pilots worldwide than any other visual system.

Every new U.S. Air Force pilot trains on our Vital visual system. Realistic takeoff and landing, air-to-ground, and air-to-air Vital simulation makes an even more effective pilot in the air.

For advanced training in Air Force F-4s, A-7s, or A-10s, Vital is there, too.

The Air Force pilot is not alone in relying on Vital. Navy F-14, S-3A, P-3C, SH-2, SH-3 and SH-60B pilots are doing the same thing. F-18 pilots in the Navy, Marine Corps and Canadian Forces soon will.

Dutch, Israeli, Danish, Egyptian, and Belgian F-16 pilots also will fly their aircraft with Vital. The Swedish Air Force JA-37 pilot gets his first

air-to-air engagement with Vital.

Vital trains more military pilots than all other visual systems combined. Thirteen services train pilots in 26 aircraft types with Vital. At five research centers, test pilots are learning to fly tomorrow's aircraft with Vital. We're very big with commercial pilots, too. In all, there are 200 Vital visual systems operating at 45 sites around the world.

But we're not finished yet. To find out what Vital is now, and what it's going to be, drop a note to Vital Marketing, McDonnell Douglas Electronics Company, Box 426, St. Charles, MO 63301. Phone (314) 925-4467. Telex 447369.

VITAL
MCDONNELL DOUGLASS

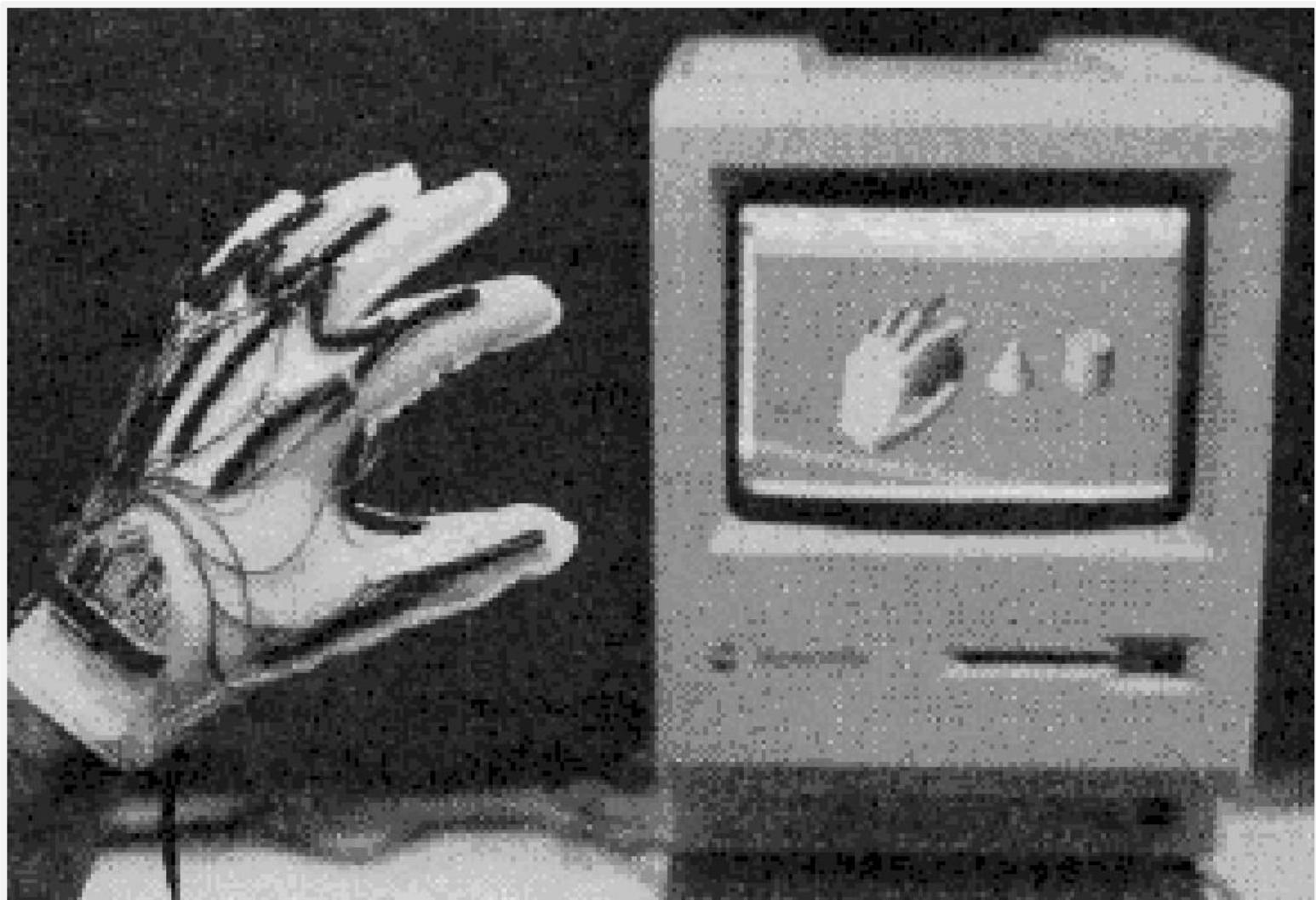
STORIA: VR NEL 1975-1985 (3)

1980: Il "Eye Tap", uno dei primi prototipi low-cost di OST-HMD, viene progettato da Steve Mann, e consiste in uno zaino-PC collegato ad un elmetto con camera e visore semitrasparente per applicazioni AR.



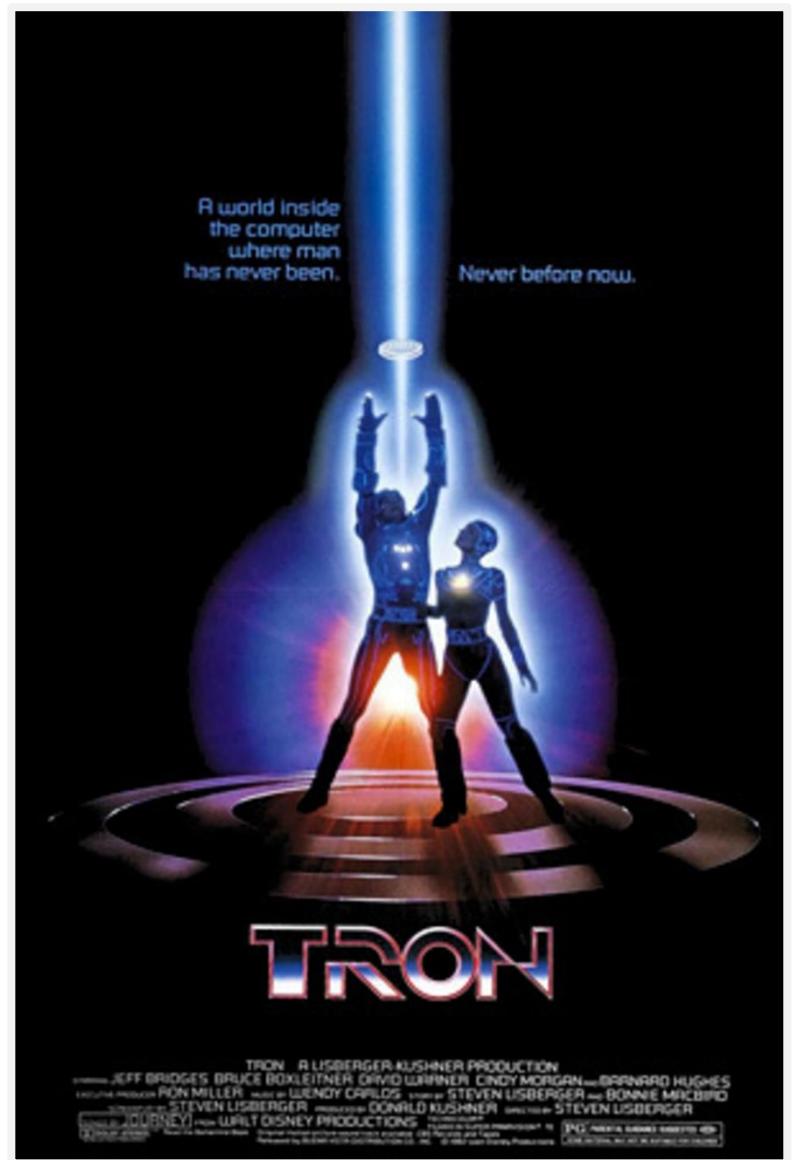
STORIA: VR NEL 1975-1985 (4)

1982: I primi guanti VR, i "Sayre Gloves", vengono inventati da Daniel Sandin e Thomas DeFanti, sono connessi a PC e integrano sensori ottici per il motion tracking delle dita.



STORIA: VR NEL 1975-1985 (5)

1982: "Tron" e' il primo film ad introdurre personaggi coinvolti in una avventura/videogioco in VR.



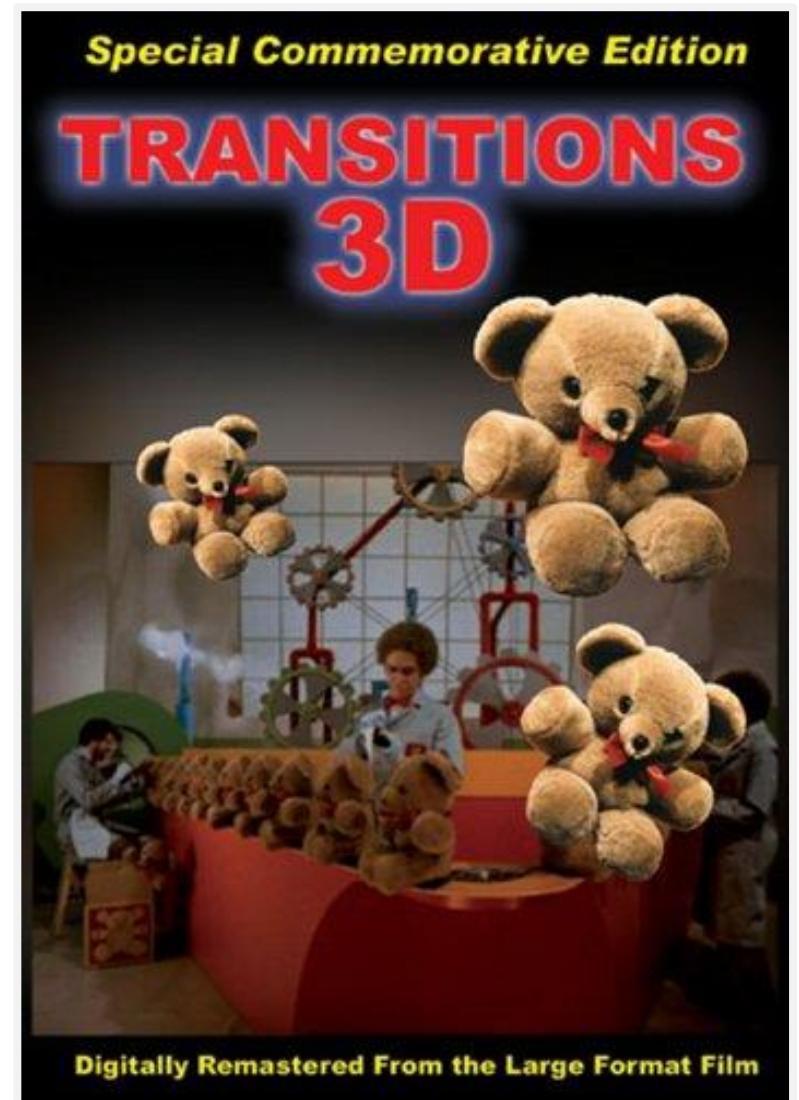
STORIA: VR NEL 1985-1995

1985: Il sistema VIEW, il primo HMD moderno, viene inventato alla NASA, inclusi i primi guanti aptici (i.e., con feedback tattile).



STORIA: VR NEL 1985-1995 (2)

1986: Il primo film 3D, "Transitions", viene pubblicato con tecnologia 3D IMAX sfruttando occhiali polarizzati per la visione degli anaglifi.



STORIA: VR NEL 1985-1995 (3)

1986: Il "Super Cockpit" viene inventato da Tom Furness e rappresenta il primo simulatore di volo VR con controllo del velivolo integrato.



STORIA: VR NEL 1985-1995 (4)

- 1985: La VPL Research viene fondata da Jaron Lanier e Thomas Zimmerman, costruisce VR HMDs e guanti VR (data gloves), e rappresenta la prima azienda del settore VR per il mercato consumer.
- 1987: Jaron Lanier (VPL Research) conia per primo il termine “virtual reality” e “data gloves”.



STORIA: VR NEL 1985-1995 (5)

1989: La stessa tecnologia usata nel sistema VIEW della NASA viene ora usata per progettare i guanti VR "Power Glove" della Nintendo (per il mercato consumer).



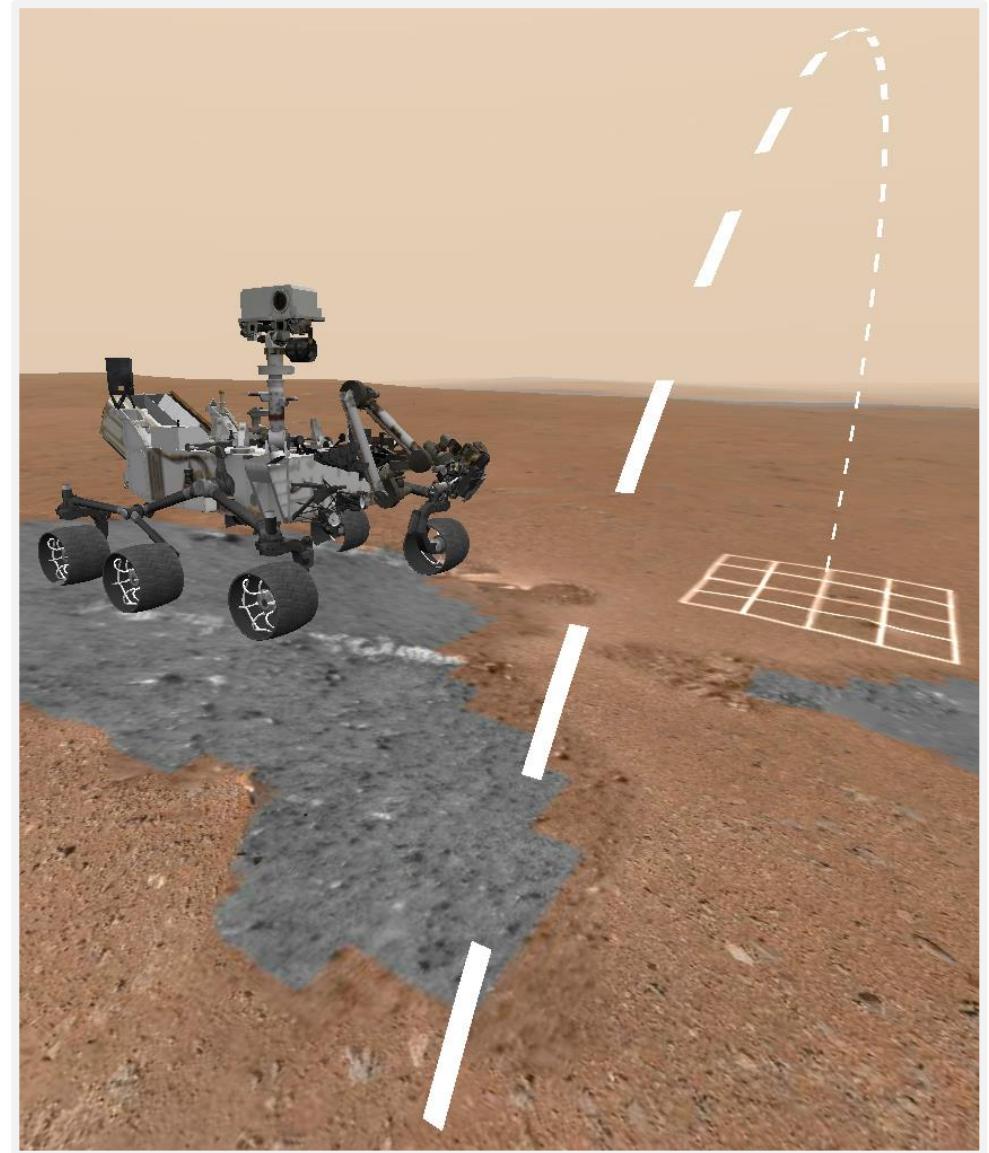
STORIA: VR NEL 1985-1995 (6)

1991: L'azienda Virtuality Group lancia diversi giochi VR arcade con inclusi occhiali VR, contenuto 3D generato in computer graphics, e multigiocatori (networked).



STORIA: VR NEL 1985-1995 (7)

1991: Un ingegnere NASA (Antonio Medina) sviluppa un sistema VR per pilotare un rover su Marte in teleoperazione (i.e., teleoperazione simulata al computer).



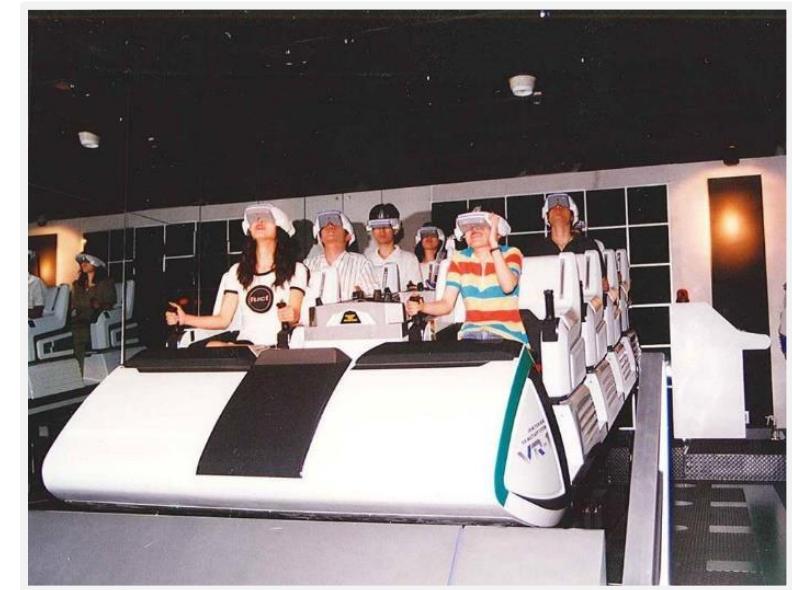
STORIA: VR NEL 1985-1995 (7)

1992: Il film "The Lawnmower Man" introduce il concetto di VR ad una audience più vasta.



STORIA: VR NEL 1985-1995 (8)

- 1993: L'azienda Sega annuncia il nuovo VR HMD (sinistra) per la console Sega Genesis, con inclusi head tracking, audio stereo, e schermi LCD.
- 1994: L'azienda Sega mette sul mercato il VR-1 (destra), un simulatore arcade con HMDs.



STORIA: VR NEL 1995-2005

1995: L'azienda Nintendo mette sul mercato il "Virtual Boy" (aka VR-32), la prima console portatile per videogiochi 3D VR (con visione stereo).



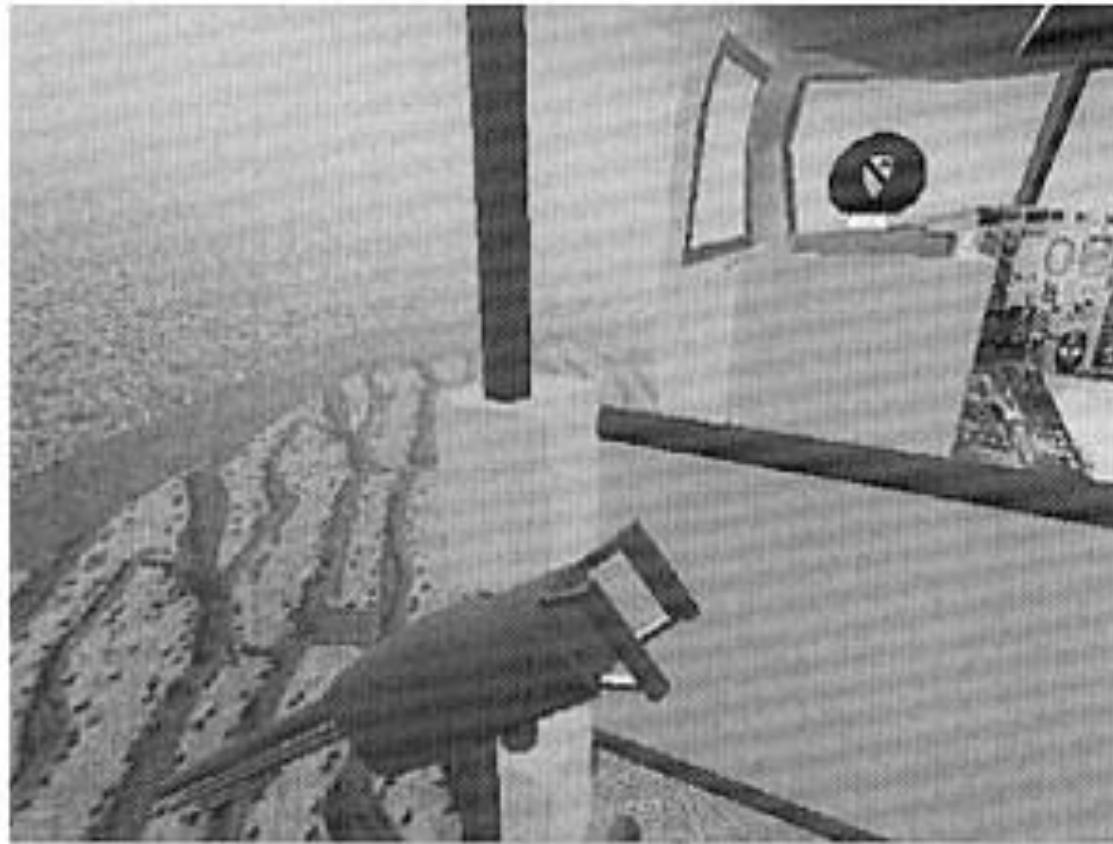
STORIA: VR NEL 1995-2005 (2)

1995: Il CAVE Automatic Virtual Environment viene inventato alla University of Illinois, e include shutter glasses LCD stereoscopici e pareti proiettate consentendo un'esperienza VR multi-utente interattiva.



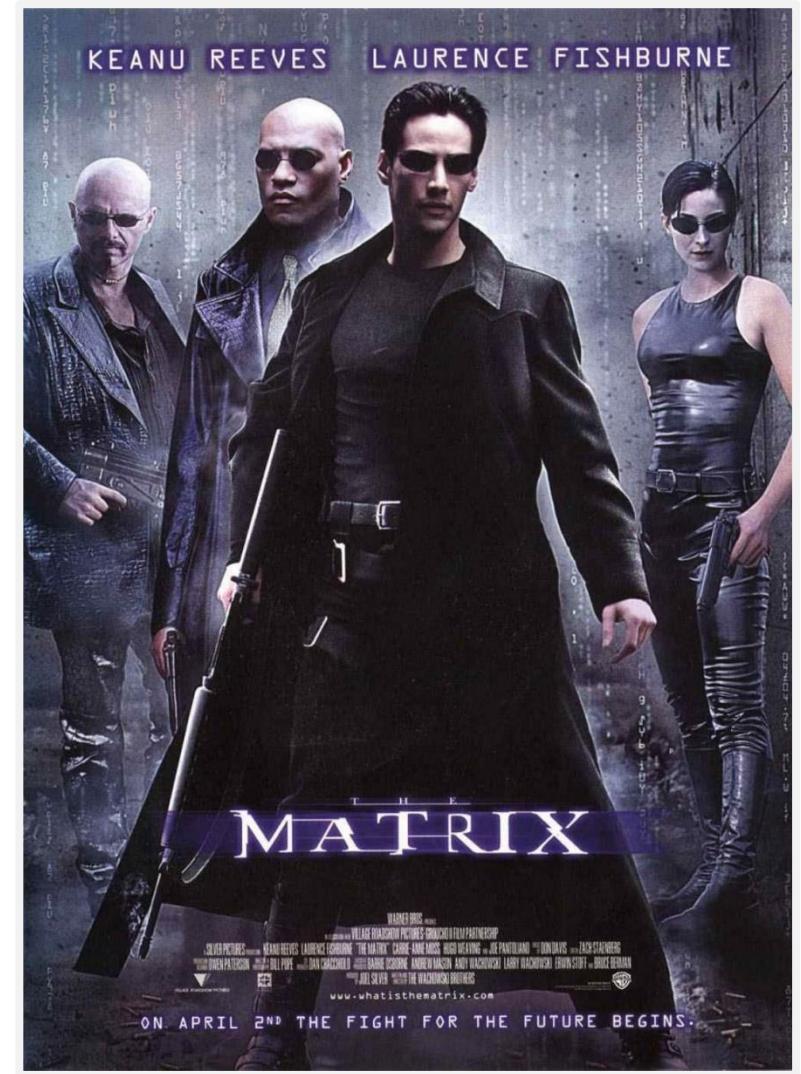
STORIA: VR NEL 1995-2005 (3)

1997: La Georgia Tech e Emory University sono tra le prime ad usare la VR nella ricerca, il sistema "Virtual Vietnam" permette di studiare e trattare la PTSD in veterani di guerra.



STORIA: VR NEL 1995-2005 (4)

1999: Il film "The Matrix" viene pubblicato, e rappresenta la prima pellicola con personaggi viventi in un mondo completamente simulato in VR.



STORIA: VR NEL 1995-2005 (5)

2001: Il sistema SAS Cube viene prodotto dalla azienda Z-A Production, e consiste in un sistema CAVE cubico commerciale.



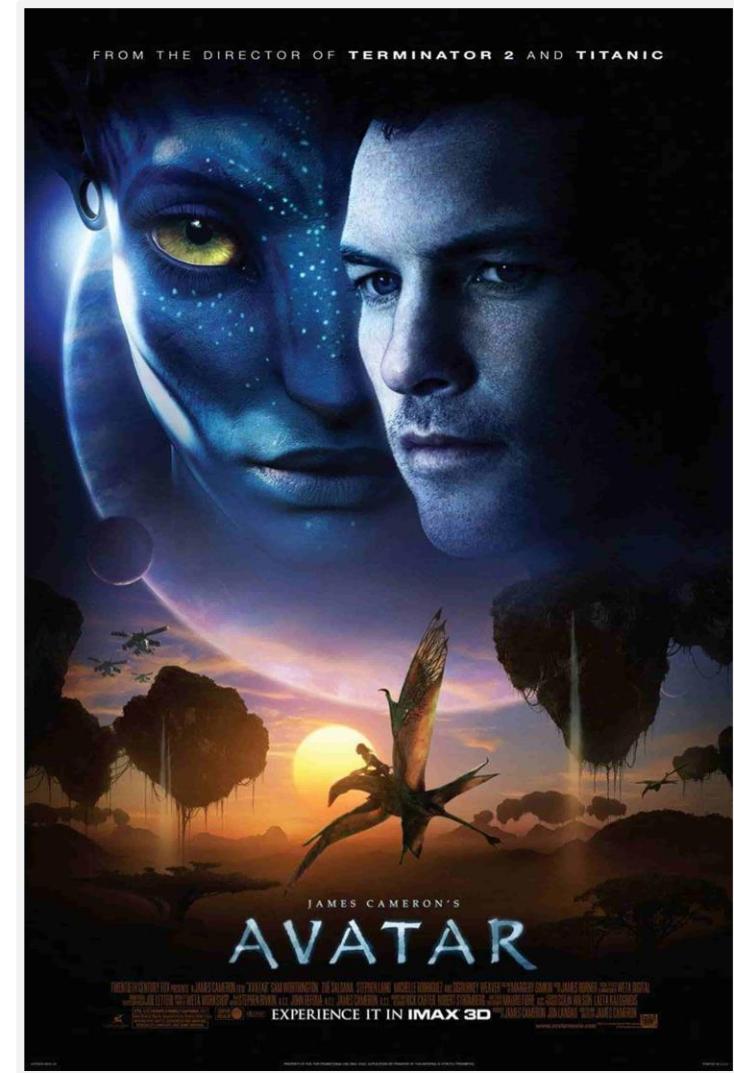
STORIA: VR DAL 2005

2007: L'azienda Google pubblica "Street View" per il suo servizio Maps, includendo immagini a 360-gradi a livello-strada generate da set di immagini catturate da speciali camere (dodecaedrali) montate sul tetto di auto.



STORIA: VR DAL 2005 (2)

2009: Il film 3D "Avatar" viene realizzato utilizzando videocamere stereo speciali e software specializzato per la computer graphics.



STORIA: VR DAL 2005 (3)

2010: Palmer Luckey inventa il primo prototipo del Oculus Rift, un HMD realizzato con componenti low-cost e consumer-grade, più tardi commercializzato collaborando con John Carmack.

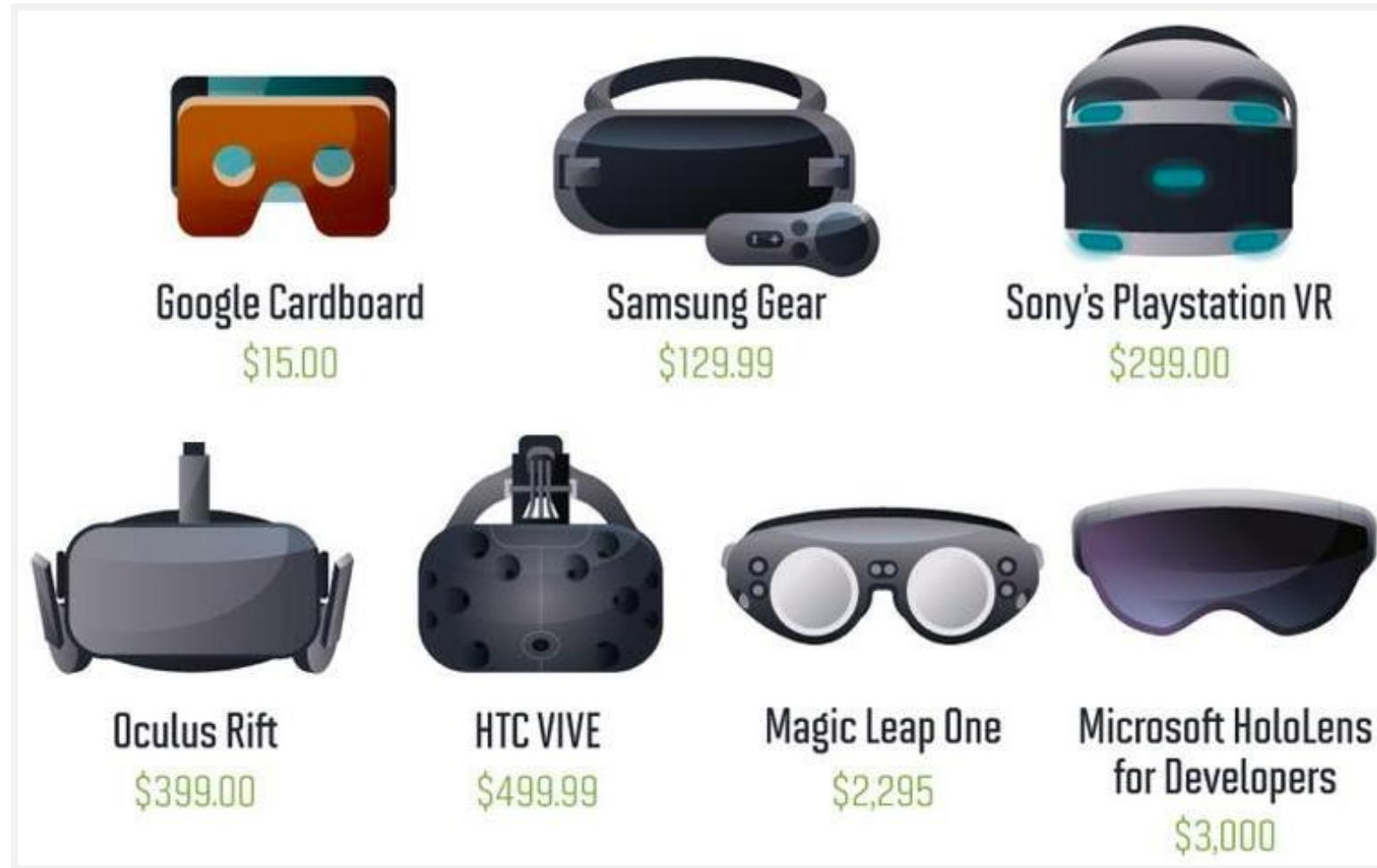


STORIA: VR DAL 2005 (4)

- 2012: Palmer Luckey lancia una campagna Kickstarter per finanziare lo sviluppo del prototipo HMD, Oculus Rift.
- 2014: Facebook acquisisce Oculus, Sony commercializza la PlayStation VR, Google lancia i Google Cardboard, e Samsung mette sul mercato il Samsung Gear VR.
- 2016: Ogni azienda tecnologica sviluppa i suoi prodotti VR, con Oculus Rift e HTC Vive che rappresentano gli HMD leader del mercato.
- 2018: Il HMD Oculus Half-Dome viene annunciato, e dovrebbe includere lenti varifocali, ampio FOV (140 gradi), face-tracking, eye-tracking, hand-tracking, etc.
- 2019: “Beat Saber”(VR) e’ la prima app a vendere oltre 1 miliardo di copie in meno di 1 anno.
- 2020: Standalone VR (non-cablata) e’ la tecnologia leader del mercato con i sistemi Oculus Go e Oculus Quest, mentre la mobile VR sta rapidamente declinando.

STORIA: VR DAL 2005 (5)

2020: Il mercato VR-AR attuale include molti HMD differenti, progettati per diversi scopi (mercato consumer, ricerca, etc.) e commercializzati con prezzi molto differenti.



DESIGN: ECOSISTEMA VR

Questa e' una breve lista dei componenti principali di un sistema VR:

- **VR Engines:** software e librerie per facilitare lo sviluppo di applicazioni VR.
- **Dispositivi di Visualizzazione:** VR HMDs, CAVEs, anaglifi, etc.
- **VR Controllers:** bacchette/wands, guanti-VR/data gloves, interfacce aptiche, etc.
- **VR Motion Tracking Systems:** integrati in dispositivi VR, e sistemi esterni.

DESIGN: VR ENGINE

Un VR engine fornisce software e librerie per i programmatori di applicazioni VR al fine di facilitare lo sviluppo.

Attualmente i VR engines esistenti sono game engines estesi col supporto di dispositivi VR (e in alcuni casi anche col supporto per lo sviluppo di applicazioni AR-MR).

Solitamente, un VR engine dovrebbe fornire:

- Funzionalita' per progettare/sviluppare/testare applicazioni VR.
- Funzionalita' per progettare/creare/modificare il contenuto 3D per VR.
- Funzionalita' per integrare dispositivi VR (HMDs, controllers, tracking systems, etc.).

Al momento, i principali VR engine sul mercato sono: Unity, e Unreal Engine.

DESIGN: VR HMD

Questa e' una sintesi delle specifiche tecniche principali di un HMD moderno:

- **Costo:** da ~20 \$ (e.g., Google Cardboard) a ~500 \$ (e.g., Oculus Quest 2).
- **Tipo Display:** da cellulare a OLED.
- **Risoluzione Display:** da cellulare in split-screen a 1920×1080 per ciascun occhio.
- **FOV:** da ~60 gradi a ~120 degrees (FOV orizzontale).
- **Audio:** da mobile audio a 360-gradi noise-cancelling headphones.
- **Refresh Rate:** da 60Hz a 120 Hz.
- **Latenza:** da ~20 ms (smartphones) a ~ 0.01 ms (high-end HMD).
- **Ottiche:** da lenti fisse a IPD e focus configurabili.

DESIGN: CAVE

Questa e' una sintesi delle principali specifiche tecniche di un sistema CAVE moderno:

- **Costo:** a partire da 50000 \$.
- **Dimensioni:** a partire da 3×3×3 m.
- **Displays:** 4-5 schermi retro-proiettati, con occhiali stereoscopici (shutter glasses).
- **Numero Utenti:** fino a 5 utenti simultaneamente.
- **Interazioni:** con controllers VR(wands, data gloves, joysticks, etc.).



DESIGN: VR CONTROLLER

I dispositivi di input standard (mouse, tastiera, etc.) non sono appropriati in VR.

Attualmente, i VR controller sono i dispositivi di input standard usati in VR.

Questo e' un breve elenco dei VR controller piu' comuni:

- **Bacchetta/Wand:** un joystick senza fili (e.g., la Nintendo Wii), spesso con tracking.
- **VR Gloves/Data Gloves:** guanti con tracking, spesso con feedback tattile.
- **Hand-Finger Trackers:** sistemi per motion tracking di mani/dita (e.g., Leap Motion).
- **VR Controllers:** console controller senza fili, con 6-DOF tracking e feedback tattile.
- **Haptic Stylus:** stilo connesso a braccio robotico, con feedback aptico e tracking.

DESIGN: MOTION TRACKING SYSTEM

L'obiettivo di un sistema di motion tracking e' quello di misurare la posizione e orientamento di un oggetto in 3D: 3-DOF (solo posizione), 6-DOF (posizione e orientamento).

In molti sistemi VR i principali targets del tracking sono il HMD e i VR controllers.

A seconda del target del motion tracking, possono essere necessari diversi requisiti di performance. Ad esempio: il tracking del HMD deve avere una frequenza > 20 Hz con una latenza < 50 ms.

TEORIA: STEREO RENDERING

Un sistema per VR e' per necessita'/definizione un sistema stereoscopico.

I sistemi stereoscopici presentano all'utente immagini di contenuti 3D in modo che l'utente percepisca il contenuto visualizzato come avente una "reale profondita'" (**depth perception**).

Tutte le diverse tecniche stereoscopiche esistenti si basano sul presentare coppie di viste (**stereo pairs**) del contenuto 3D virtuale/digitale in modo indipendente a ciascun occhio dell'utente, stimolando in questo modo la **stereopsis**: la percezione della profondita' come se il contenuto 3D virtuale fosse visto con la vista binoculare nel mondo reale.

I sistemi stereoscopici tradizionali utilizzano hardware specifico per "dirigere" le immagini destra e sinistra di una coppia stereo (stereo pair) solamente all'occhio appropriato (e.g., HMDs, active shutter glasses, anaglyph glasses, etc.).

Il "santo graal" dei sistemi stereoscopici sarebbe un sistema autostereoscopico: una tecnica che non richieda nessun strumento e/o ambiente particolare (e.g., holografie, parallax barriers, lenticular sheets, etc.).

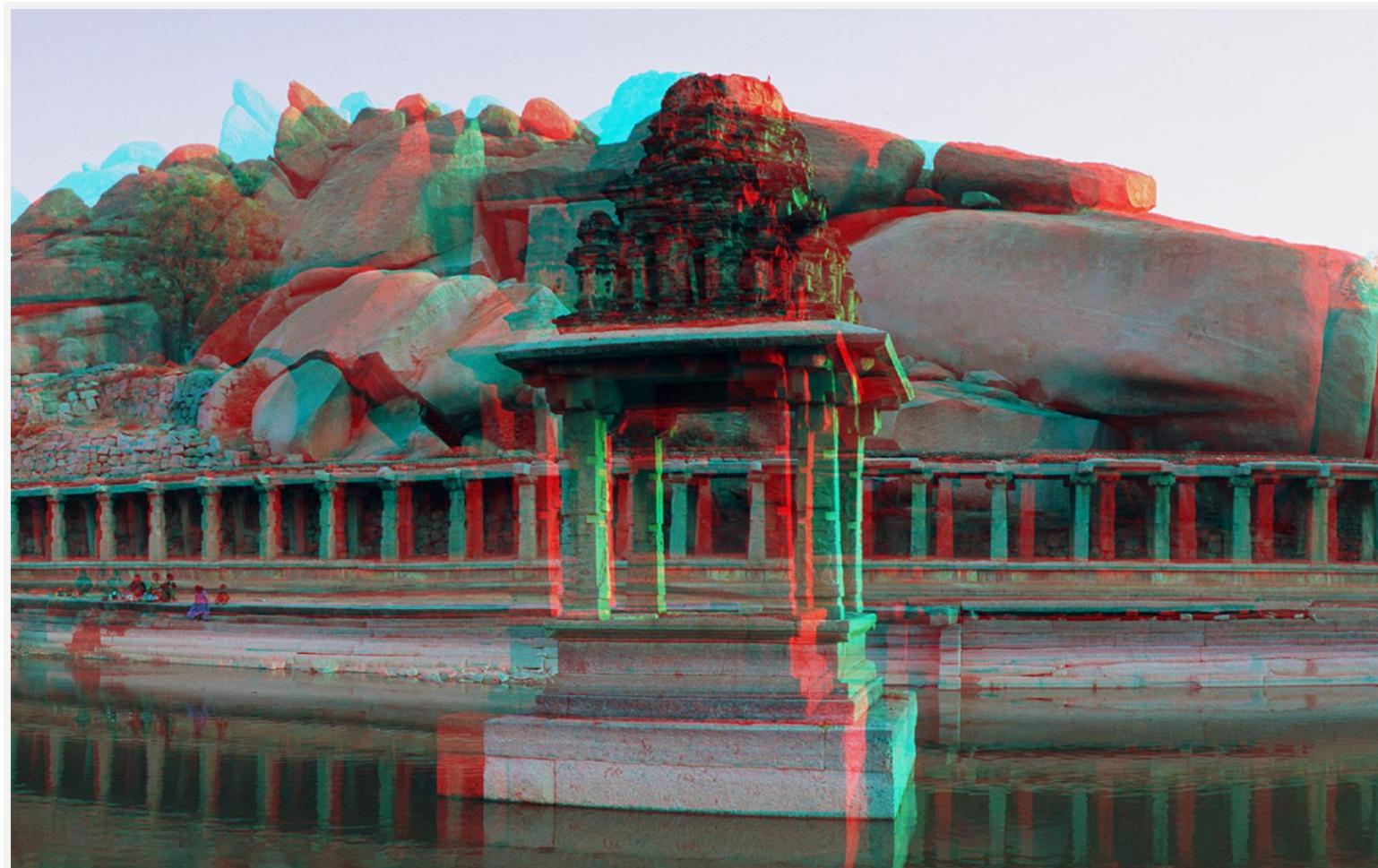
HARDWARE: ANAGLYPH GLASSES

Un anaglifo 3D consiste nell'effetto stereoscopico ottenuto attraverso l'integrazione (coding) di una coppia stereo in una singola immagine utilizzando filtri di colore, e sfruttando occhiali speciali (anaglyph glasses) che includono gli appropriati filtri di colore per bloccare/dirigere (decoding) l'immagine appropriata all'occhio relativo.



HARDWARE: ANAGLYPH GLASSES (2)

Un esempio di anaglyph red-cyan.



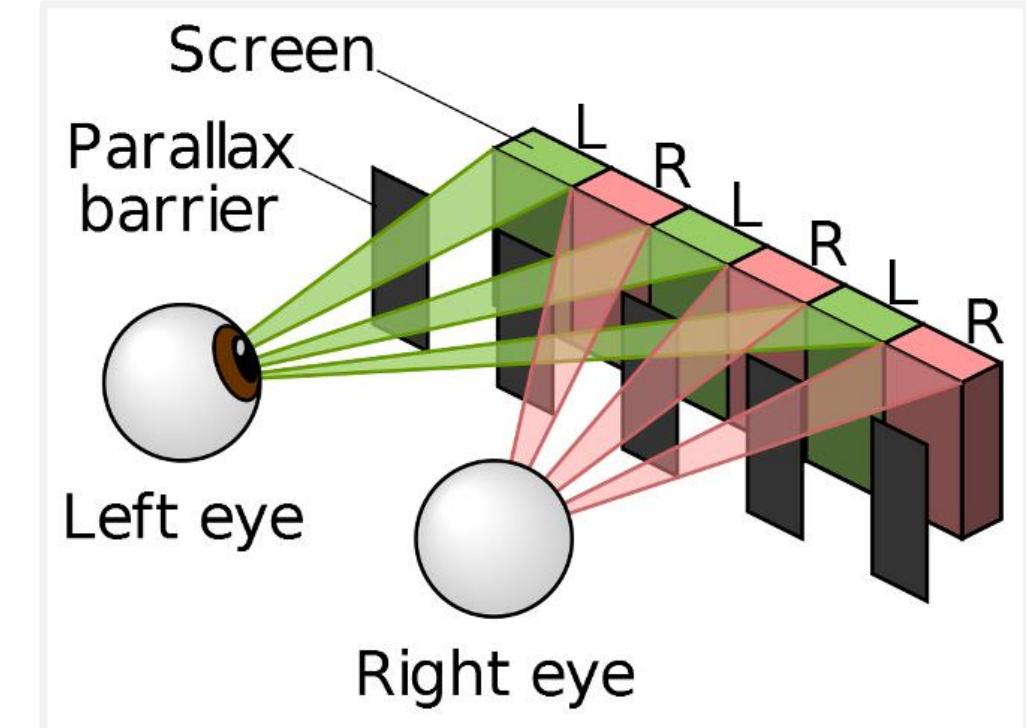
HARDWARE: ACTIVE SHUTTER GLASSES

Un sistema stereoscopico basato su "active shutter glasses" consiste nella visualizzazione di coppie stereo in maniera alternata su video, utilizzando degli occhiali speciali in grado di bloccare la visione ad ogni occhio individualmente. La visualizzazione a video e il blocco della visione sugli occhiali vengono effettuate in modo sincrono e ad alta frequenza.



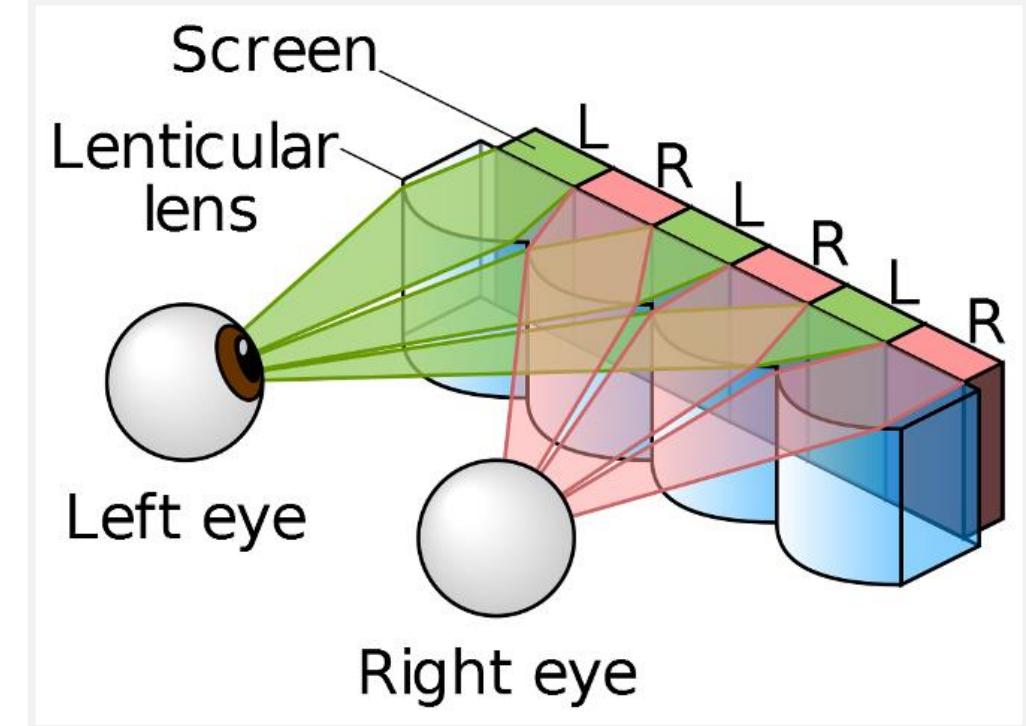
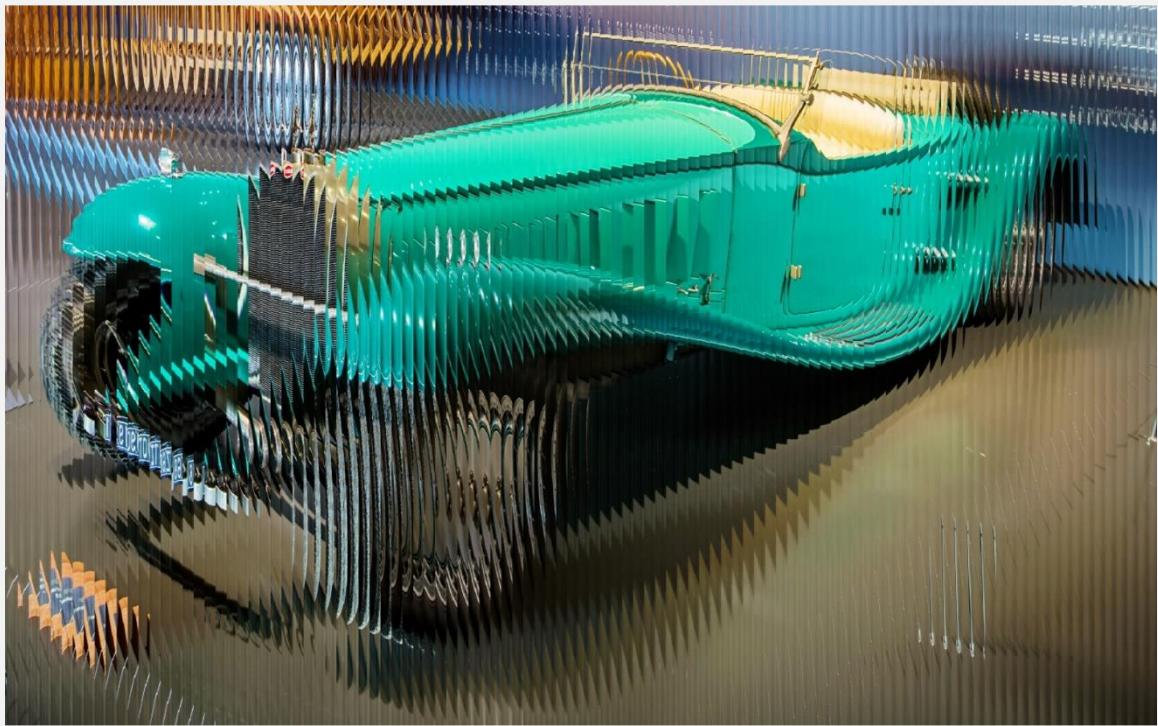
HARDWARE: PARALLAX BARRIER

La barriera di parallasse (parallax barrier) e' un sistema autostereoscopico che permette di visualizzare coppie stereo senza bisogno di utilizzare dispositivi ottici speciali, grazie all'immagine visualizzata integrante la coppia stereo in modo interallacciato e ad una barriera semi-trasparente che implementa in modo appropriato il parallasse necessario.



HARDWARE: LENTICULAR LENS

Le lenticular lens sono un sistema autostereoscopico che permette di visualizzare coppie stereo senza bisogno di utilizzare dispositivi ottici speciali, grazie all'immagine visualizzata integrante la coppia stereo in modo interallacciato e ad una barriera trasparente di lenti che indirizza la vista appropriatamente.



TEORIA: DEPTH PERCEPTION

La percezione di profondità (depth perception) essenziale nei sistemi VR si basa sulla corretta generazione (computer-based) di coppie stereo.

Per capire completamente come funziona la depth perception occorre evidenziare che ci sono molti "indizi di profondità" (depth cues) che il nostro cervello utilizza...

Depth Cues Presenti in Immagini 2D:

- **Prospettiva:** oggetti (proiezioni) rimpiccioliscono più distanti sono (dal punto di vista).
- **Dimensioni Relative:** ci aspettiamo che oggetti noti abbiano determinate dimensioni.
- **Dettaglio:** oggetti ravvicinati appaiono con più dettagli di oggetti distanti.
- **Occlusione:** oggetti occludenti sono in primo piano rispetto a oggetti occlusi.
- **Illuminazione:** oggetti ravvicinati sono più illuminati di oggetti distanti.
- **Movimento Relativo:** oggetti distanti si muovono più lentamente di oggetti ravvicinati.

TEORIA: DEPTH PERCEPTION (2)

Depth Cues **Non** Presenti in Immagini 2D:

- **Binocular Disparity:** differenza tra immagine sinistra e destra in una coppia stereo.
- **Accommodation:** tensione muscolare necessaria negli occhi per mettere a fuoco.
- **Convergence:** tensione muscolare necessaria per ruotare gli occhi verso il fuoco.

La **binocular disparity** è considerata il depth cue dominante (di solito).

Altri depth cue, se incorretti, possono avere un forte effetto dannoso.

TEORIA: DEPTH PERCEPTION (3)

Affinche' la depth perception venga ottenuta correttamente, la coppia stereo deve essere visualizzata appropriatamente in modo da essere "fusa" dal cervello dell'utente (visual cortex).

Coppia Stereo Creata con Conflitto nei Depth Cues:

- Uno qualsiasi dei depth cue puo' diventare dominante.
- La depth perception puo' diventare esagerata o ridotta (rispetto a quella corretta).
- Il contenuto 3D virtuale puo' essere "scomodo" da vedere.
- La coppia stereo puo' non venire "fusa" dalla visual cortex (vista doppia).

Coppia Stereo Creata Correttamente:

- La binocular disparity e la convergence sono corretti.
- La accomodation e' sempre inconsistente, ma e' un problema tollerato (di solito).

TEORIA: DEPTH PERCEPTION (4)

Parallasse Orizzontale: La distanza tra le proiezioni dello “stesso punto 3D” nelle immagini destra e sinistra di una coppia stereo.

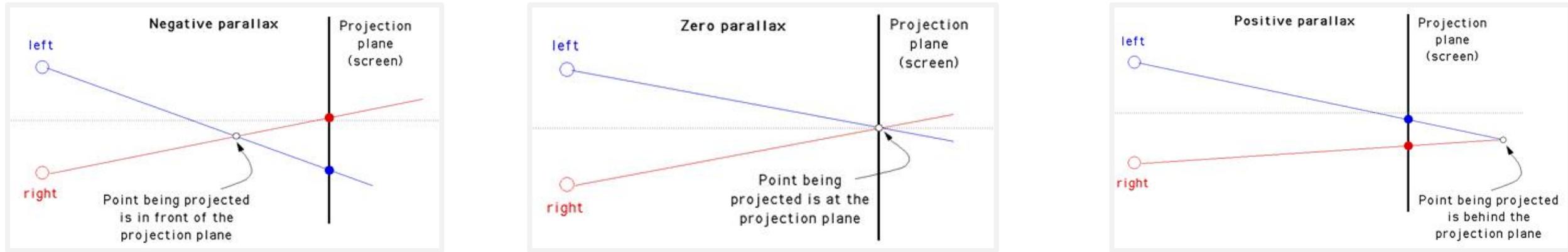
In una coppia stereo, c’è un **parallasse orizzontale positivo** quando le proiezioni dello “stesso punto 3D” sono dallo stesso lato del rispettivo occhio. Il massimo valore del parallasse orizzontale positivo è relativo ad un “punto 3D” situato all’infinito.

In una coppia stereo, c’è un **parallasse orizzontale negativo** quando le proiezioni dello “stesso punto 3D” sono dal lato opposto rispetto all’occhio corrispondente. Questo valore è uguale alla IPD se il “punto 3D” è a metà strada tra il piano di proiezione e il centro degli occhi.

In una coppia stereo, c’è un **parallasse orizzontale nullo (zero)** se il “punto 3D” è situato sul piano di proiezione e quindi le sue proiezioni sono coincidenti.

TEORIA: DEPTH PERCEPTION (5)

In figura le viste top-down della configurazione delle proiezioni di un “punto 3D” in una coppia stereo nei casi di parallasse orizzontale: negativo (sx), nullo/zero (centro), e positivo (dx).

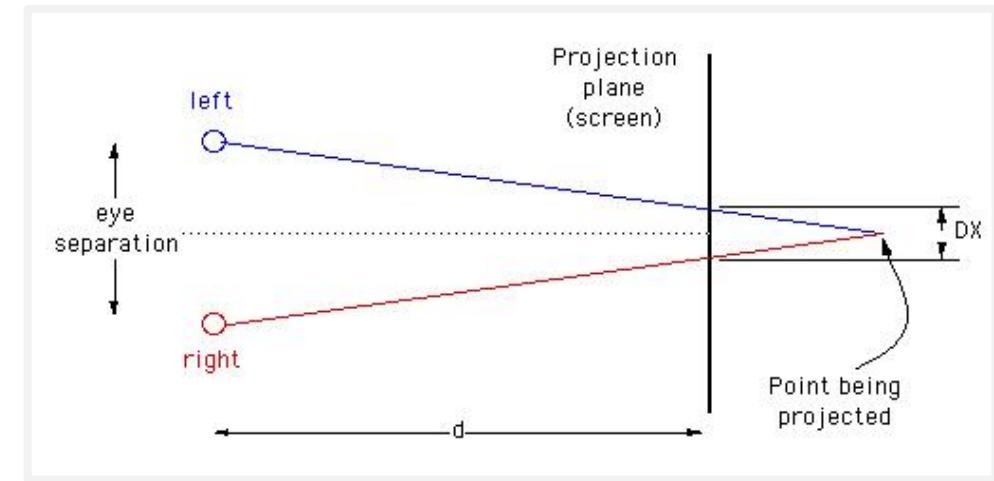


Nota: Una coppia stereo corretta non deve presentare alcun parallasse verticale; se e' presente la configurazione della camera stereo virtuale (stereo rig) e' errata.

TEORIA: DEPTH PERCEPTION (6)

Nel discutere gli effetti del parallasse orizzontale e' utile considerare l'angolo di parallasse.

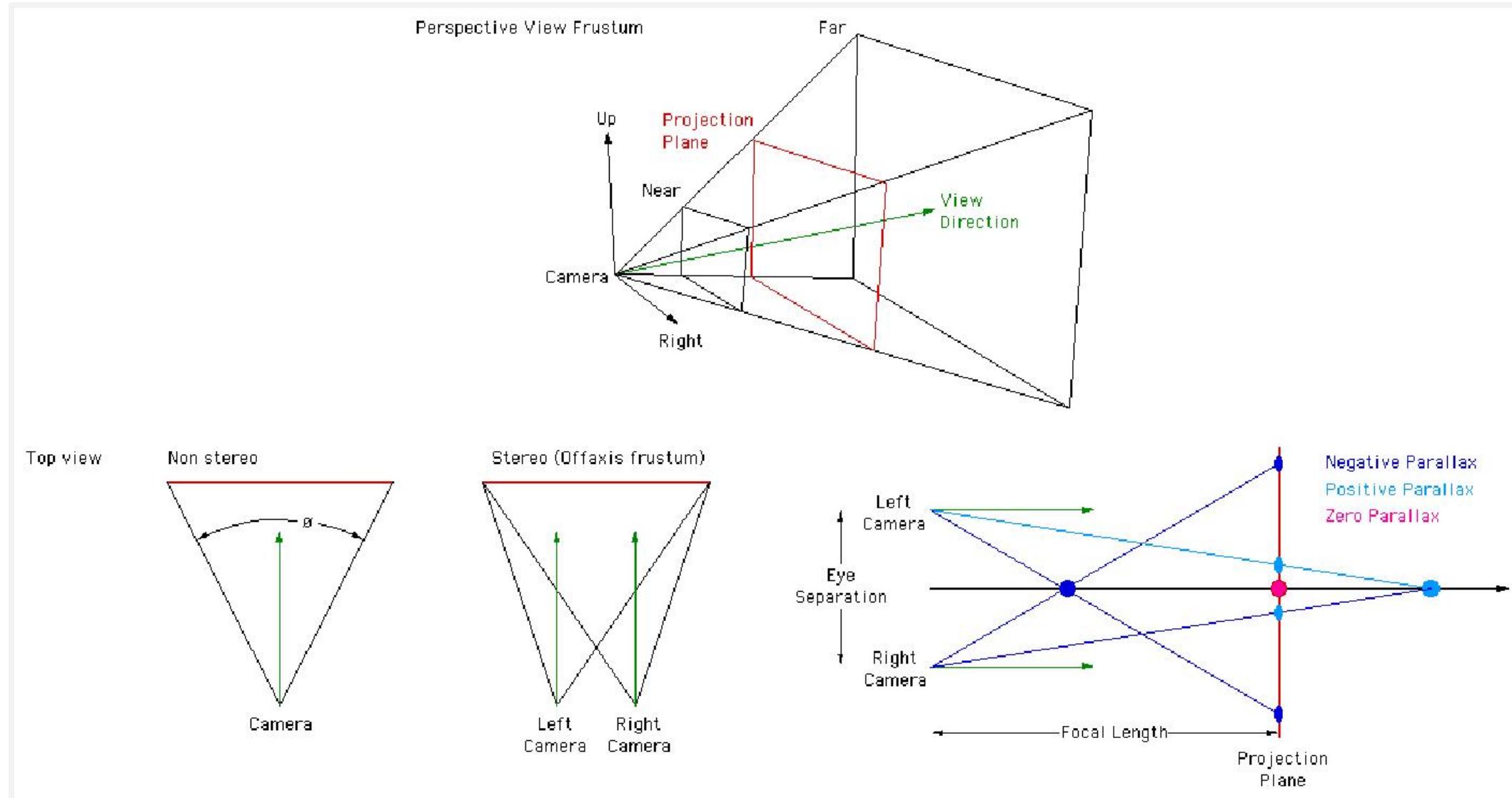
$$\text{angolo di parallasse} = \theta = 2 \operatorname{atan} \left(\frac{DX}{2d} \right)$$



Nell'equazione dell'angolo di parallasse theta: DX e' il parallasse orizzontale, e d e' la distanza tra gli occhi/camere e il piano di proiezione.

Nota: L'angolo di parallasse theta e' positivo per punti 3D "dietro" al piano di proiezione in 3D, e negativo per punti 3D "davanti" al piano di proiezione in 3D.

TEORIA: CAMERA VIRTUALE STEREO



TEORIA: CAMERA VIRTUALE STEREO (2)

Camera Virtuale Stereo: Configurazione della camera virtuale che include 2 camere (stereo rig) ognuna relativa ad un occhio dell'utente.

La posizione/orientamento e configurazione delle 2 camere e del loro viewing frustum devono essere impostate in modo appropriato per poter generare coppie stereo corrette.

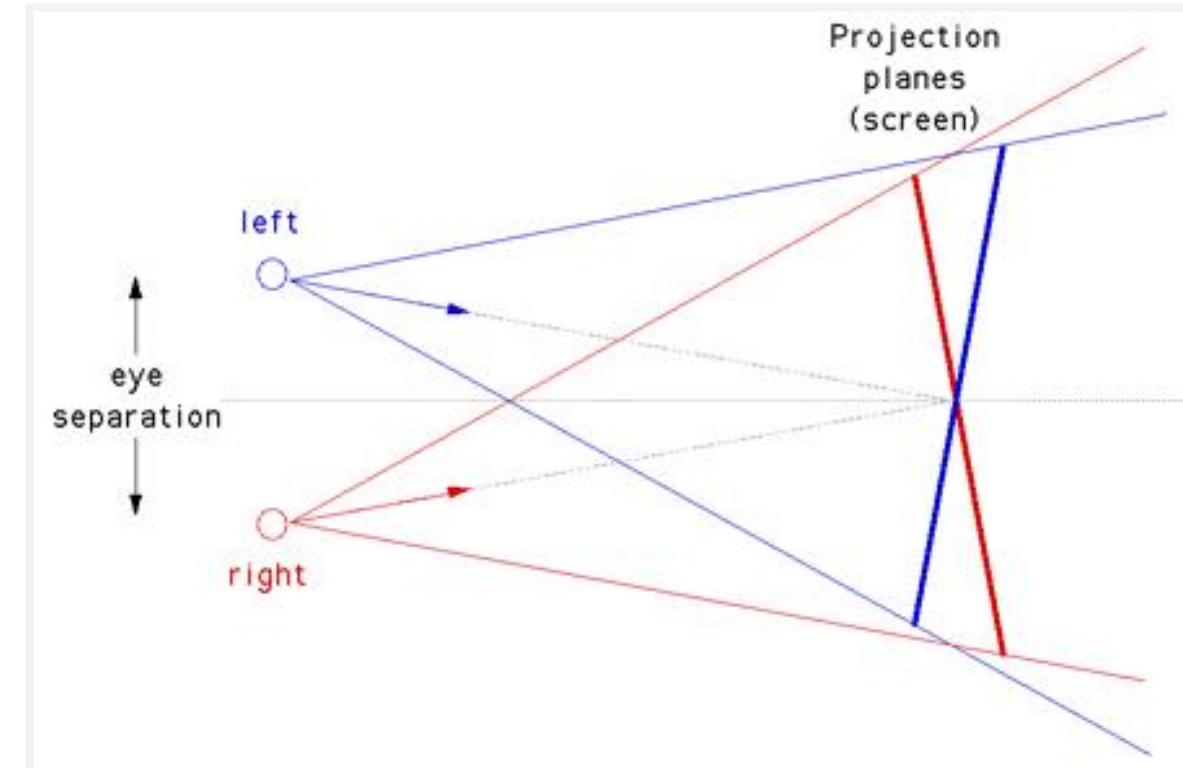
TEORIA: CAMERA VIRTUALE STEREO (3)

Toe-In: Configurazione della camera virtuale stereo **incorrecta** dal punto di vista ottico, ma comunque utile nei casi in cui non e' possibile configurare le camere appropriatamente per limitazioni tecniche (game engine, computer graphics, etc.).

Le camere usano apertura simmetrica, e puntano allo stesso punto focale.

Le coppie stereo create col metodo toe-in sono stereoscopiche ma non confortevoli, perche' includono parallasse verticale (a causa dei piani di proiezione non coplanari).

Il parallasse verticale incrementa dal centro verso i bordi dell'immagine, e se si incrementa l'apertura delle camere.



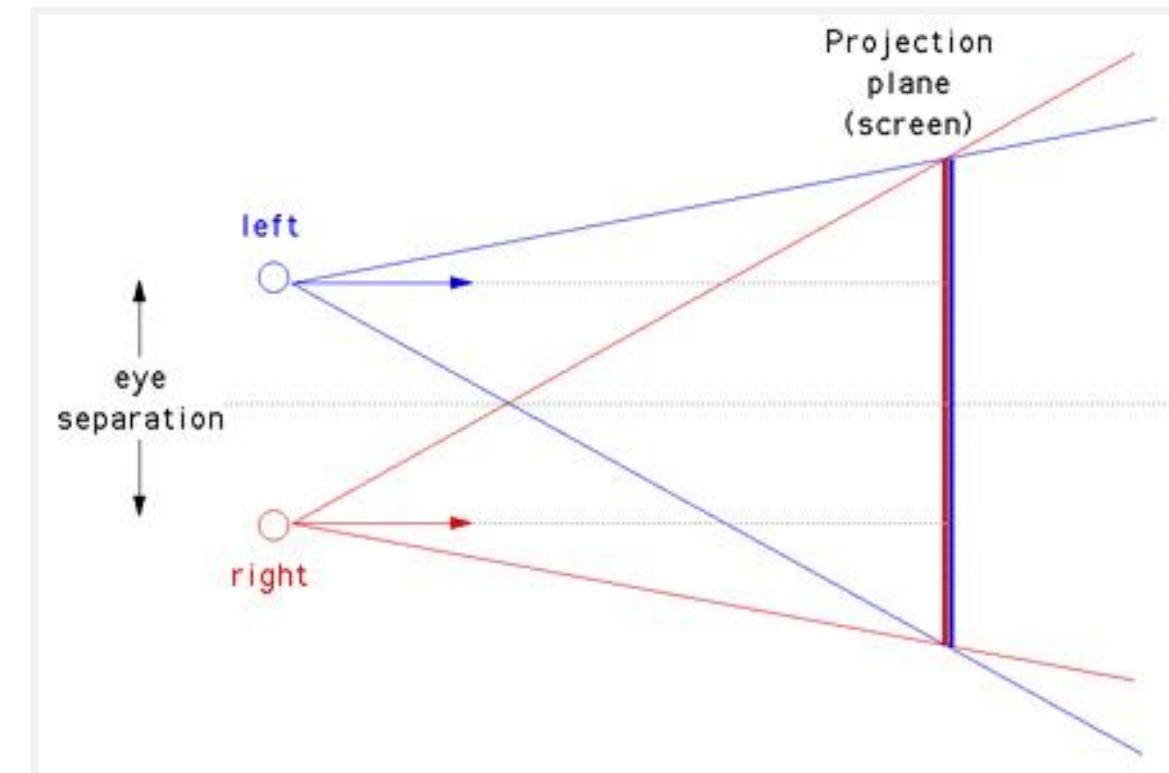
TEORIA: CAMERA VIRTUALE STEREO (4)

Off-Axis: Configurazione della camera virtuale stereo corretta dal punto di vista ottico.

Le camere usano apertura asimmetrica, e puntano a punti focali differenti.

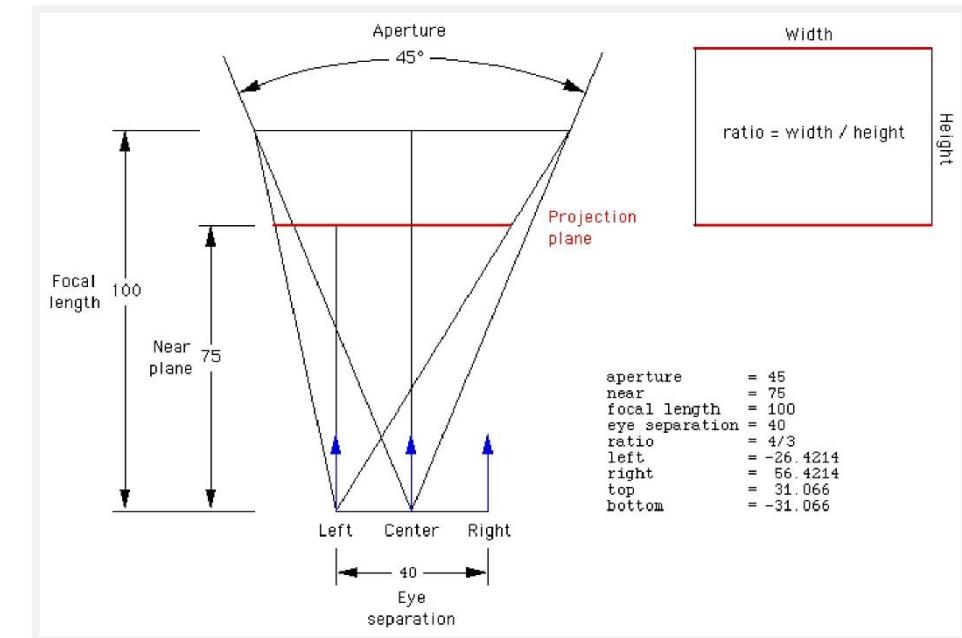
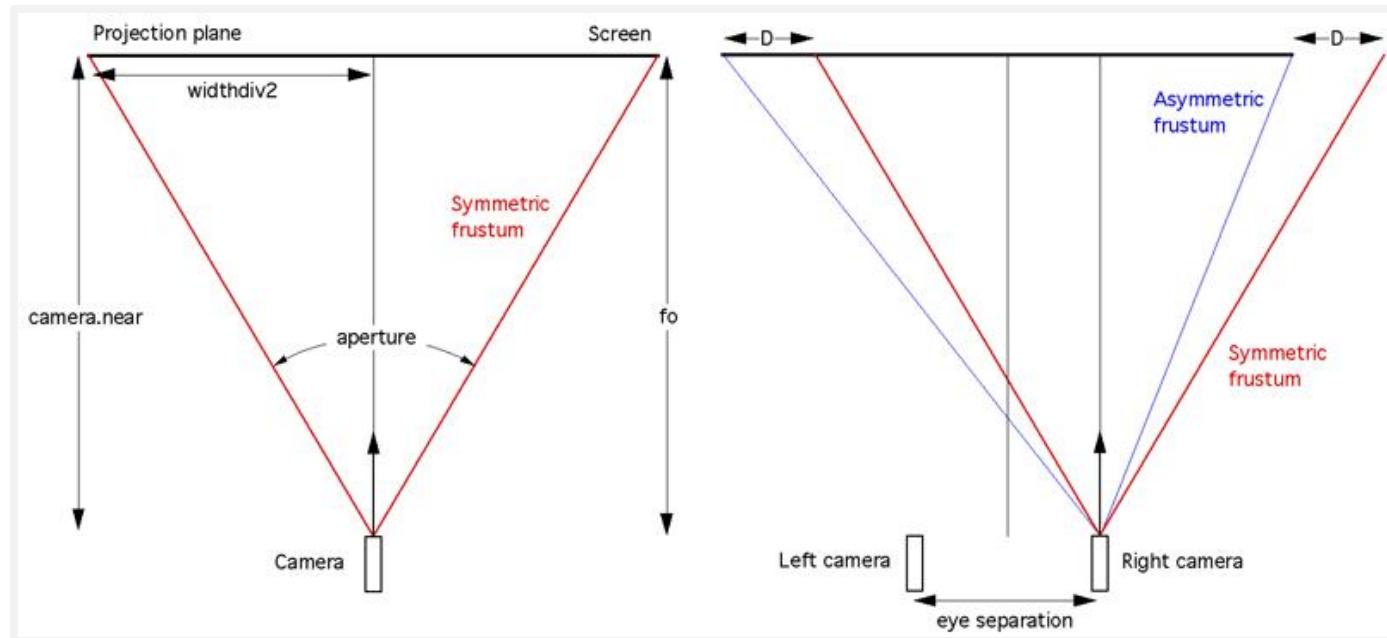
Le coppie stereo create col metodo off-axis sono stereoscopiche e non includono parallasse verticale (a causa dei piani di proiezione coplanari).

Questa configurazione necessita della capacita' di impostare un viewing frustum asimmetrico.



TEORIA: CAMERA VIRTUALE STEREO (5)

In figura un esempio di configurazione off-axis per una camera virtuale stereo.



TEORIA: VISIONE STEREO

Questa e' una sintesi dei concetti principali per l'applicazione pratica della visione stereo basata su generazione computerizzata di coppie stereo (continua):

- Gli oggetti 3D di fronte al piano di proiezione 3D appaiono "di fronte" al monitor fisico.
- Gli oggetti 3D dietro al piano di proiezione 3D appaiono "dentro" al monitor fisico.
- Di solito e' piu' "facile" vedere coppie stereo di oggetti 3D che appaiono "dentro" al monitor fisico; quindi il punto focale dovrebbe essere posto piu' vicino alla camera virtuale invece che agli oggetti 3D target.
- Per facilitare la "fusione" dovremmo mantenere il valore assoluto dell'angolo di parallasse theta al di sotto dei 1.5 gradi per tutti i punti 3D.

TEORIA: VISIONE STEREO (2)

Questa e' una sintesi dei concetti principali per l'applicazione pratica della visione stereo basata su generazione computerizzata di coppie stereo (continua):

- Il grado dell'effetto stereo (depth perception) dipende sia dalla distanza della camera dal piano di proiezione che dalla separazione tra le camere dx e sx.
- Una separazione tra le camere dx e sx troppo grande (aka hyperstereo) puo' generare coppie stereo "difficili" da fondere.
- Una buona approssimazione per la separazione delle camere dx e sx e' $1/20$ della distanza dal piano di proiezione; questa puo' essere considerata la massima separazione per una visione stereo confortevole.
- E' buona pratica assicurarsi che il parallasse orizzontale negativo non ecceda mai la separazione tra le camere.

TEORIA: VISIONE STEREO (3)

Questa e' una sintesi dei concetti principali per l'applicazione pratica della visione stereo basata su generazione computerizzata di coppie stereo (continua):

- L'apertura della camera virtuale dovrebbe essere uguale allo "sweet spot" del sistema di visualizzazione VR (di solito 45-60 gradi).
- Scegliere appropriatamente la lunghezza focale considerando il contenuto 3D (i.e., la distanza associata con il parallasse orizzontale nullo), onde evitare che oggetti 3D appaiano piu' vicini della meta' della lunghezza focale.
- La separazione delle camere stereo (eye separation) dovrebbe essere circa 1/30 della lunghezza focale.
- Una "eye separation" simile a quella umana garantisce (1) un realistico senso di scala e distanza, ma (2) puo' risultare in poca depth perception per contenuto 3D distante.

TEORIA: VR SICKNESS

VR Sickness: (aka "simulator sickness") e' il discomfort associato all'utilizzo di ambienti virtuali/simulati, e dovuto a conflitti tra i sensi corporei (visivi, auditivi, etc.).

La "VR/simulator sickness" puo' essere causata dai fattori seguenti:

- **accelerazione** (minimizzare ogni accelerazione).
- **controllo** (assicurare che l'utente abbia un buon grado di controllo in VR).
- **durata sessione** (permettere/incoraggiare pause nelle sessioni VR).
- **flusso visivo** (evitare flussi visivi intensi).
- **binocular disparity** (non tutti sono in grado di fondere coppie stereo).
- **FOV** (ridurre il campo di vista puo' ridurre il comfort).
- **latenza** (minimizzarla perche' i "lags" causano discomfort in VR).

Nota: Considerare che l'esperienza rende resistenti alla "VR/simulator sickness"!

HARDWARE: ANATOMIA DI UN VR HMD

In questa sezione viene discussa l'anatomia di un moderno HMD per VR generico. Lo sviluppo di applicazioni VR per HMD puo' non richiede la conoscenza dettagliata del hardware target, ma di solito queste informazioni sono utili per il design e lo sviluppo/debugging.

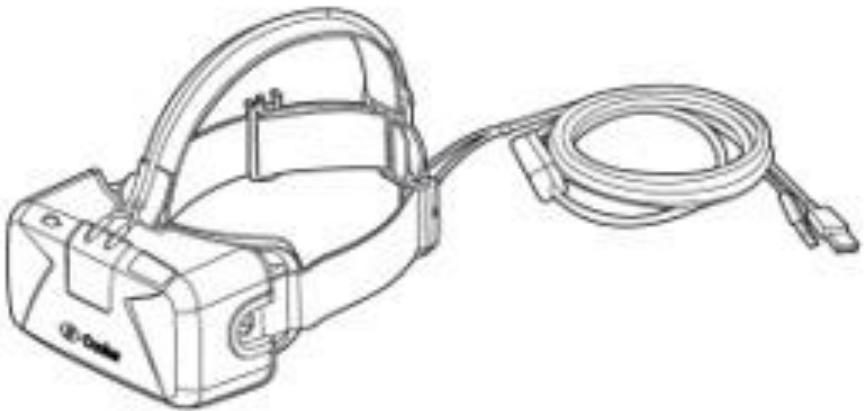
In figura 3 HMDs moderni per VR, con design diversi ma molte similitudini.



HARDWARE: ANATOMIA DI UN VR HMD (2)

In figura alcuni dei componenti principali di un moderno HMD per VR:

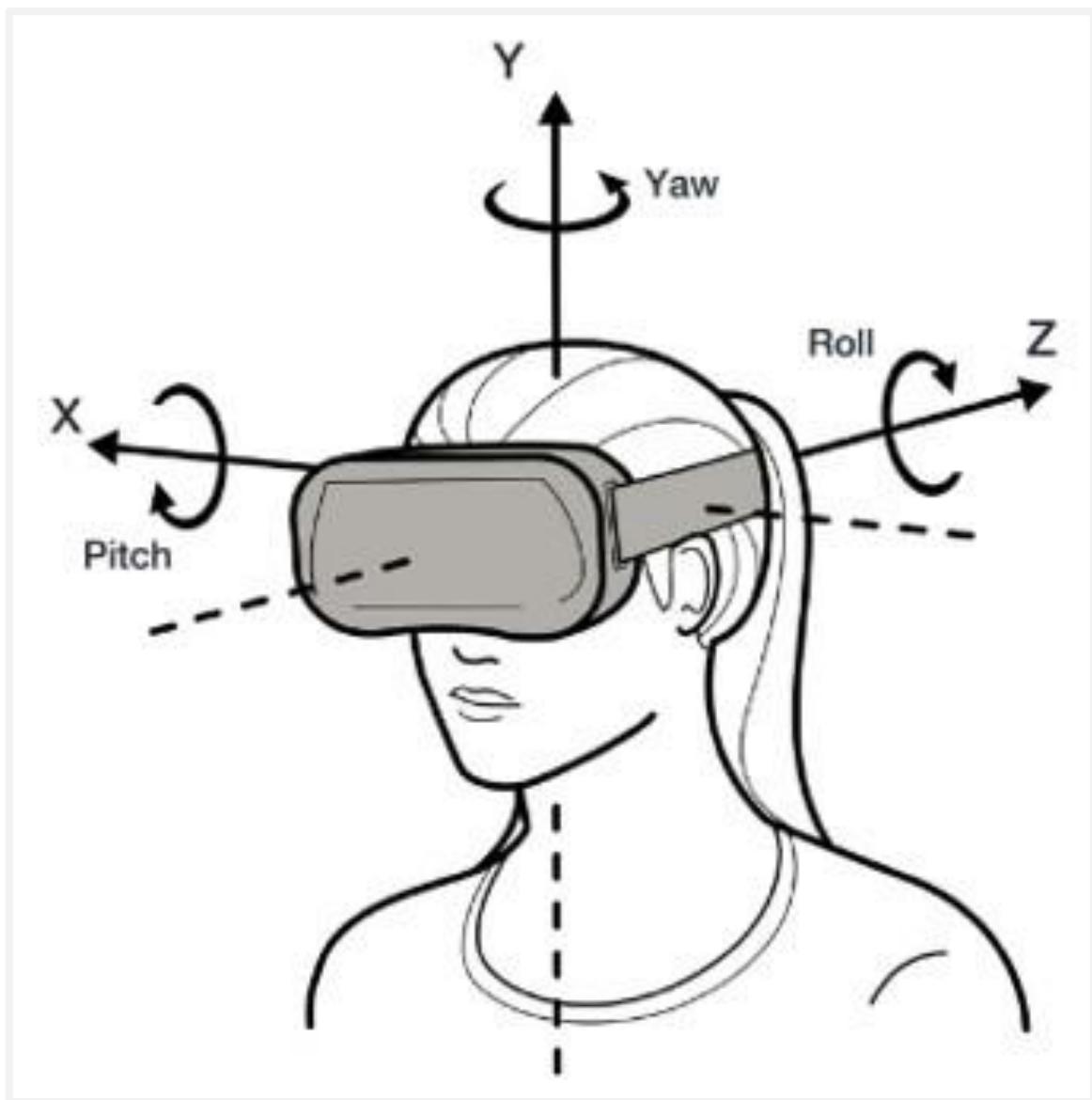
- L'elmetto (in alto) di solito contiene una IMU per parte dell tracking (rotazionale).
- Le lenti interne (al centro) per la messa a fuoco del display interno (di tipo Fresnel per limitare peso e volume).
- Sistema di tracking esterno (in basso) per parte del tracking (posizionale).



HARDWARE: ANATOMIA DI UN VR HMD (3)

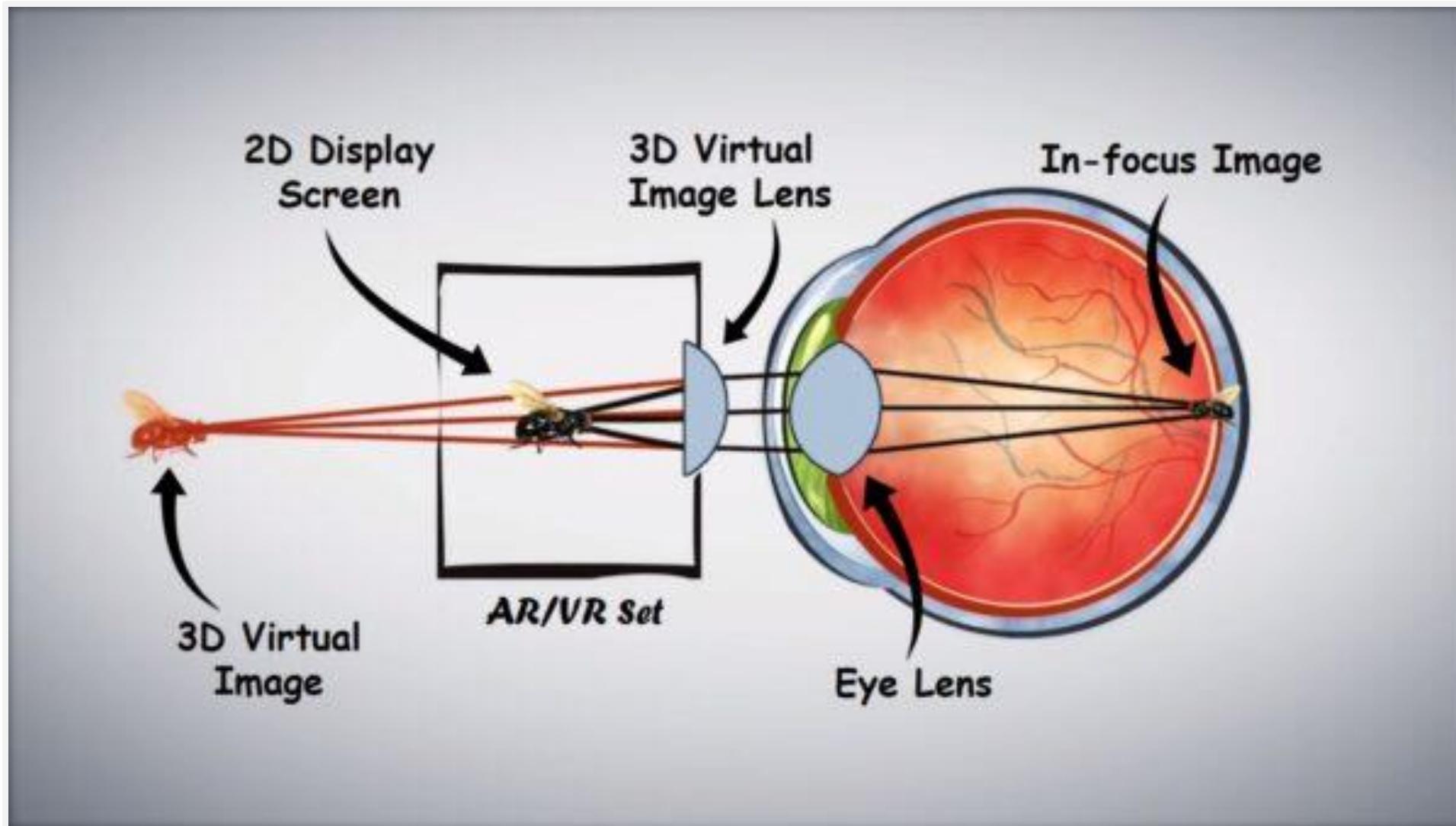
In figura un esempio di sistema di riferimento interno di un HMD per VR:

- **Pitch:** rotazione con sguardo verso alto/basso (e.g., rotazione su asse X).
- **Yaw:** rotazione con sguardo dx/sx e testa in verticale (e.g., rotazione su asse Y).
- **Roll:** rotazione con sguardo di fronte e testa in diagonale dx/sx(e.g., rotazione su asse Z).



HARDWARE: ANATOMIA DI UN VR HMD (4)

Il diagramma in figura illustra il ruolo delle lenti interne in un HMD per VR moderno.



HARDWARE: ANATOMIA DI UN VR HMD (5)

In figura la visualizzazione interna in modalita' split-screen di un HMD per VR moderno.



TEORIA: RENDERING PER UN VR HMD

Il rendering per un HMD per VR moderno deve tener conto che di solito il display interno dell'elmetto è configurato in split-screen stereo (vedi figura). Questo significa che:

- La configurazione della camera virtuale stereo deve combaciare con il setup del display/ottiche del HMD.
- Il software deve processare il contenuto 3D due volte: una volta per generare la vista per ciascun occhio.



TEORIA: RENDERING PER UN VR HMD (2)

L'apertura (FOV) della camera virtuale stereo (camera FOV, o CFOV) deve eguagliare quella dell'area visibile a schermo (display FOV, o DFOV).

Display FOV (o DFOV): La parte del campo di vista fisico dell'utente che e' occupata dal contenuto 3D virtuale. E' una caratteristica fisica del VR HMD (del suo HW e delle sue ottiche).

Camera FOV (o CFOV): L'apertura (FOV) della camera virtuale stereo.

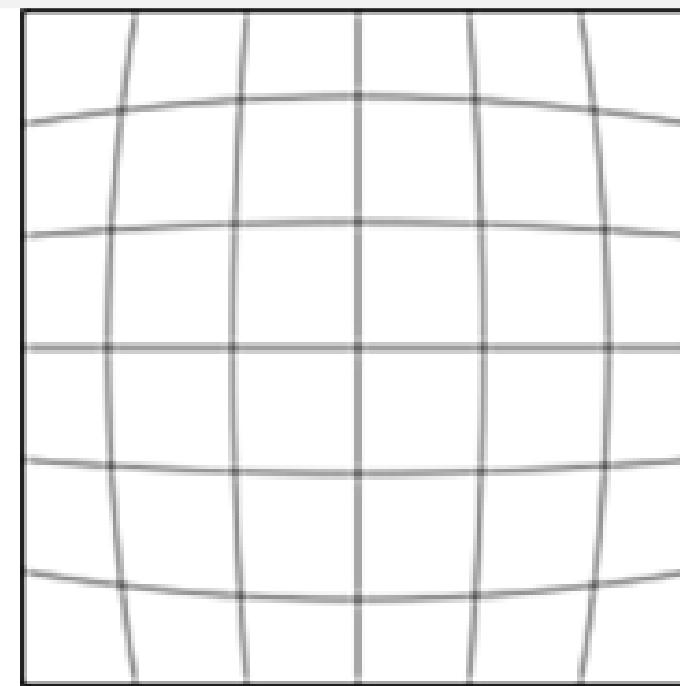
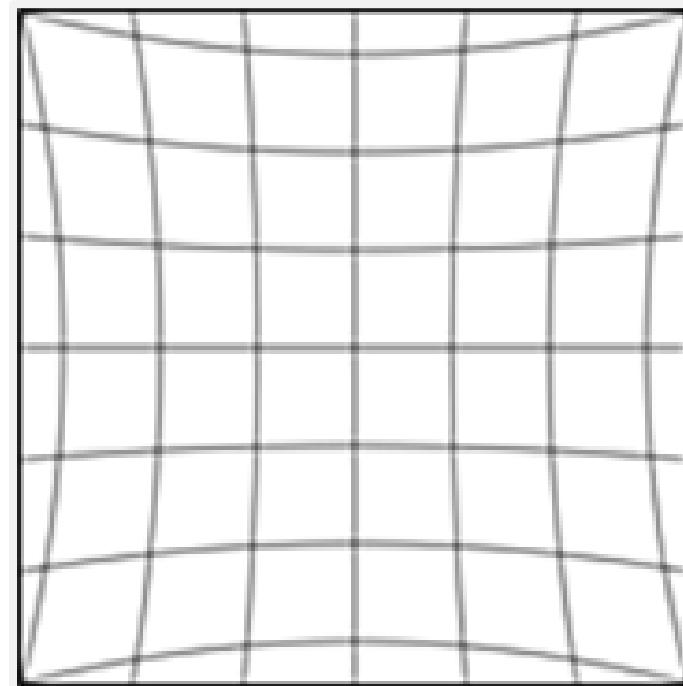
In VR non c'e' visione dell'ambiente reale esterno, e l'ambiente virtuale occupa gran parte della visione periferica dell'utente. Per questo motivo e' fondamentale che il CFOV e il DFOV combacino perfettamente.

Scale: Il rapporto tra CFOV e DFOV si chiama "scale", e in VR dovrebbe essere sempre 1.0.

TEORIA: RENDERING PER UN VR HMD (3)

L'utilizzo di lenti interne in ogni HMD per VR moderno richiede una correzione della distorsione introdotta dalle lenti.

Le lenti interne causano una “**pincussion distortion**” (in figura, a sx), che viene neutralizzata visualizzando il contenuto 3D sullo split-screen stereo applicando una distorsione opposta: una “**barrel distortion**” (in figura, a dx).



TEORIA: RENDERING PER UN VR HMD (4)

Le lenti interne in un VR HMD causano anche una “chromatic aberration” da correggere.

Chromatic Aberration: Dislocamento dei colori di una immagine causata dalla visione della stessa attraverso lenti.

Questo effetto è docuto all'indice di rifrazione delle lenti che è variabile per ogni lunghezza d'onda della luce.

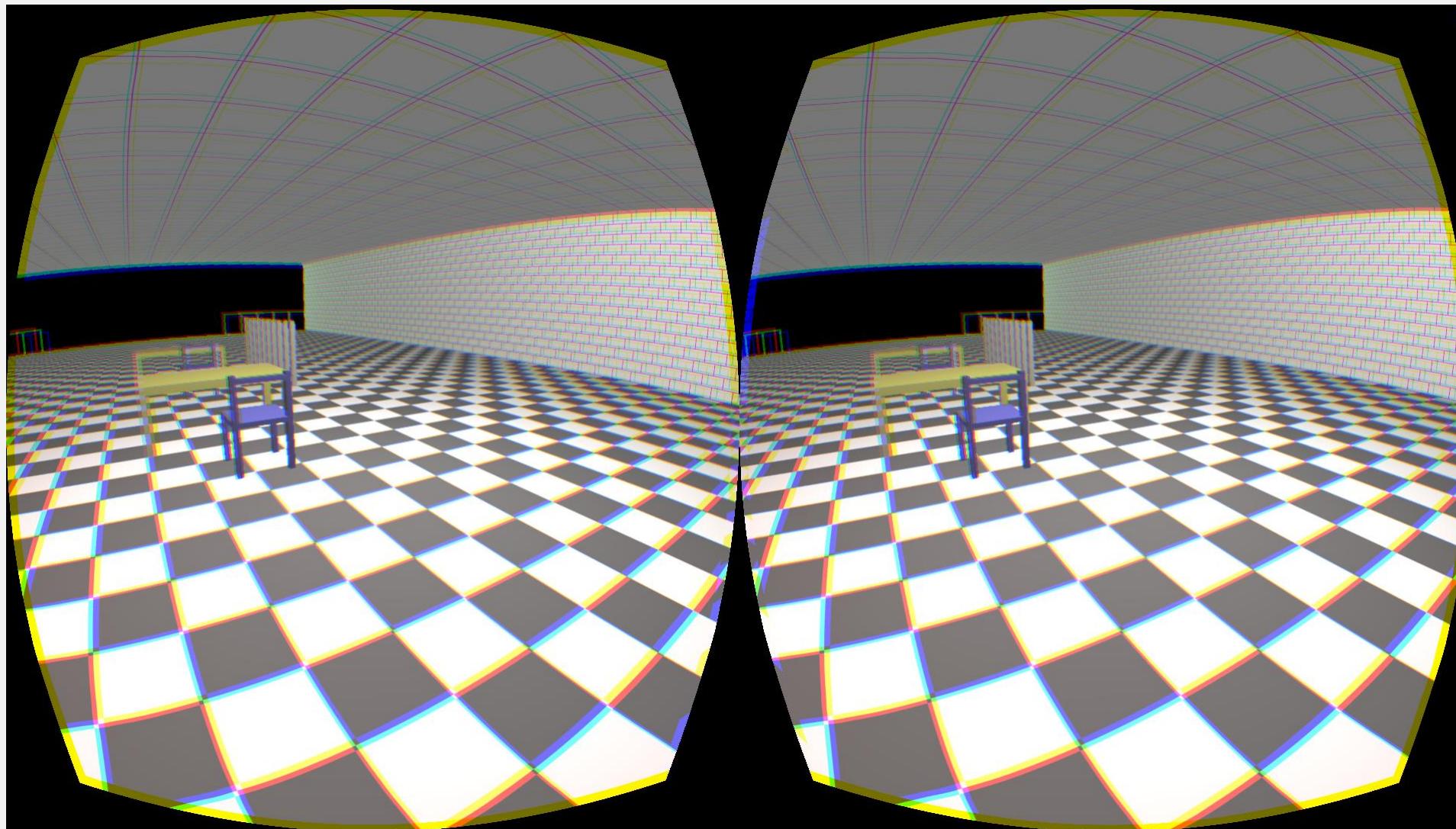
Le immagini visualizzate da un display sono composte da elementi di colore individuali (red, green, e blu) che subiscono rifrazioni diverse dalle lenti.

Questo effetto è più significativo ai bordi rispetto al centro, e colori diversi sono affetti in modo diverso (e.g., blu meno di rosso).

Nota: La “chromatic aberration” va corretta a basso livello (nel device driver o in uno shader).

TEORIA: RENDERING PER UN VR HMD (5)

In figura la “chromatic aberration” dovuta alle lenti interne di un HMD per VR moderno.



TEORIA: USER INTERFACE IN VR

Il design e l'implementazione di user interfaces per VR include:

- **Menus:** in viste specifiche (settings), menus in-game (pause), integrati in 3D, etc.
- **Heads-Up Displays (HUDs):** minimaps, timers, etc.
- **Controlli:** tastiera-mouse, point-and-click, joypad, etc.
- **Interaction Feedback:** visivo (highlighting), auditivo (clicks), tattile (vibration), etc.

Il design di UI per VR include la progettazione di quali informazioni/interazioni vanno rese disponibili all'utente, e come queste informazioni/interazioni devono essere implementate (di solito cercando di massimizzare il comfort e l'usabilità).

TEORIA: USER INTERFACES IN VR (2)

In figura una UI (non-diegetica) di un videogame standard (con first-person controller).



TEORIA: USER INTERFACES IN VR (3)

Queste sono delle linee guida per il design e l'implementazione di UI per VR:

- Integrare UI nell'ambiente virtuale (**UI diegetica**) e' il design migliore.
- Disegnare i **reticles/crosshairs** sul target (e non a una distanza prefissata).
- Nascondere **strumenti virtuali** quando non in uso.
- Fare attenzione con le **inconsistenze dei virtual avatar**.
- Nessun input device standard e' ideale per la VR, **l'opzione migliore sono i gamepads**.
- Usare un **input device familiare** (l'utente non vede l'input device in VR).
- Attenzione con l'uso intenso di **gestures e gaze** (causa affaticamento utente).
- La **locomozione e' inesplorata in VR** (puo' creare problemi mai affrontati).

TEORIA: VR BEST PRACTICES

Questa e' una sintesi delle "best practices" per il design e l'implementazione di sistemi/applicazioni per VR (continua):

- Non trascurare i "depth cues" monoculari (e.g., illuminazione, dettaglio, etc.).
- L'intervallo di profondita'(depth range) piu' confortevole in VR e' 0.75-3.50 metri.
- Assicurarsi che gli elementi testuali in VR siano ben leggibili ed evitare elementi piccoli e non necessari in aree dove l'utente concentra l'attenzione.
- Le esperienze VR piu' confortevoli non integrano alcun "self-motion" per l'utente, ad esclusione dei movimenti di testa e corpo limitati al guardarsi intorno.
- Se in VR e' necessario implementare il "self-motion", i movimenti lenti dell'utente sono i piu' confortevoli.
- Assicurare che ogni forma di accelerazione (dell'utente o di altri elementi) sia il piu' possibile breve ed infrequente.

TEORIA: VR BEST PRACTICES (2)

Questa e' una sintesi delle "best practices" per il design e l'implementazione di sistemi/applicazioni per VR (continua):

- Assicurare che ogni forma di accelerazione (dell'utente o di altri elementi) sia il più possibile breve ed infrequente.
- I movimenti dell'utente e della camera virtuale stereo non devono mai essere disaccoppiati.
- Non integrare "head bobbing" nei movimenti dell'utente in VR.
- Per esperienze VR più confortevoli minimizzare i movimenti all'indietro e laterali.
- Usare cautela per le situazioni che inducono (visivamente) forti sensazioni di movimento (e.g., salire scale, accelerazioni, etc.).

TEORIA: VR BEST PRACTICES (3)

Questa e' una sintesi delle "best practices" per il design e l'implementazione di sistemi/applicazioni per VR (continua):

- Minimizzare (se possibile) sia la "lag" che la latenza.
- Se la latenza e' inevitabile, un "lag" variable e' peggio di un "lag" costante.

Seguire queste linee guida significa massimizzare il comfort e l'usabilita' dell'esperienza VR, minimizzando: l'affaticamento visivo, la sensazione di disorientamento, la nausea, problemi al funzionamento visivo-motor post-sessione VR, etc.

PARTE 2C: VR APP IN UNITY

Unity fornisce un **template** per progetti VR che puo' essere usato come punto d'inizio.

Il template **pre-installa alcuni packages** necessari per lo sviluppo VR e **pre-configura** il progetto con il tracking per VR HMD e VR controllers nella scena.

Unity supporta direttamente queste piattaforme VR:

- Oculus
- Windows Mixed Reality
- (altre piattaforme VR sono supportate attraverso specifici plugins).

Vedi: docs.unity3d.com/manual/xr-template-vr

UNITY: USARE IL TEMPLATE VR

Per usare il template VR fornito da Unity:

- Creare un nuovo progetto Unity, selezionando il template VR.
- Selezionare la piattaforma VR target nel editor usando:
menu > Edit > Project Settings > XR Plug-in Management > ...
- Assicurarsi che i settings siano configurati correttamente per ottimizzare la build per la piattaforma VR target selezionata.

Vedi: docs.unity3d.com/manual/xr-template-vr

RIFERIMENTI

- Sito web di Unity: unity.com
- Manuale online di Unity: docs.unity3d.com/manual
- Scripting reference online di Unity: docs.unity3d.com/scriptreference
- Asset Store di Unity: assetstore.unity.com
- “Virtual Reality” by Steven M. LaValle: lavalle.pl/vr

APPENDICI

"Experience is the hardest kind of teacher. It gives you the test first and the lesson afterwards."

OSCAR WILDE.

UNITY: ALTRE EDITOR WINDOWS

Questo e' un breve elenco di alcune delle altre windows disponibili nello Unity Editor:

- **Device Simulator:** View per simulare un mobile device (cellulare, tablet, etc.).
- **Animation:** Window per creare/editare/ispezionare clip di animazione.
- **Profiler:** Window usata per investigare o ottimizzare la performance dell'app.
- **Lighting:** Window usata per configurare i settaggi di illuminazione di progetto.
- **Animator:** Window usata per configurare una finite-state machine (FSM) per il controllo di animazioni (sistema Unity Mecanim).
- **Audio Mixer:** Window usata per effettuare il mixaggio di audio clips.
- **Navigation:** Window usata per configurare tecniche di pathfinding (game AI) usando il sistema Unity Navmesh.

Vedi: docs.unity3d.com/manual/usingtheeditor

SCRIPTING: ATTRIBUTI (TODO)

Work in progress...

Vedi: [docs.unity3d.com/ScriptReference/SerializeField](https://docs.unity3d.com/ScriptReference/SerializeField.html)

Vedi: [docs.unity3d.com/ScriptReference/HideInInspector](https://docs.unity3d.com/ScriptReference/HideInInspector.html)

SCRIPTING: COROUTINES (TODO)

Work in progress...

Vedi: docs.unity3d.com/manual/coroutines

SCRIPTING: INSTANZIARE PREFABS (TODO)

Work in progress...

Vedi: docs.unity3d.com/manual/instantiatingprefabs

SCRIPTING: ORDINE ESECUZIONE SCRIPT (TODO)

Work in progress...

Vedi: docs.unity3d.com/manual/class-monomanager

SCRIPTING: MESH PROCEDURALI

La classe **Mesh** permette allo sviluppatore di creare/modificare modelli 3D a runtime. Questa classe memorizza tutti i dati (vertici, triangoli, etc.) in arrays interni separati.

Nota: Le mesh triangolari sono le piu' usate, ma Unity supporta anche altri tipi di topologie.

In generale, ci sono 3 tipi di operazioni per le quali viene usata la classe Mesh:

- Creare una mesh da 0 via scripting a runtime (o nell'editor).
- Modificare gli attributi dei vertici (ma non i triangoli) a runtime ogni frame.
- Modificare gli triangoli e vertici a runtime ogni frame.

Vedi: [docs.unity3d.com/scriptreference/mesh](https://docs.unity3d.com/ScriptReference/Mesh.html)

