

Introduction to Unity

Unity



Unity is a cross-platform game development system

Consists of a game engine and an IDE

Can be used to develop games and applications for many different AR/VR platforms

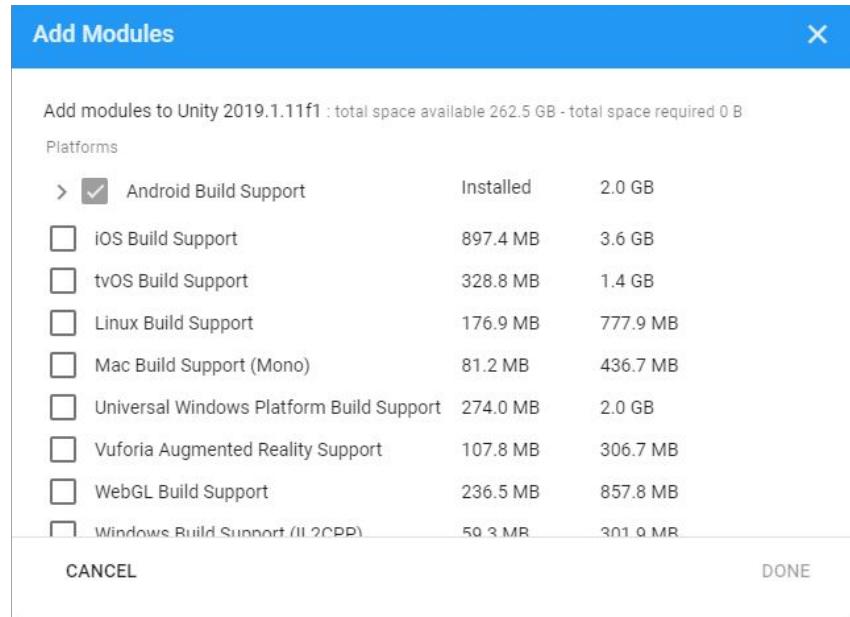


Installation

Unity is already installed on the lab computers in LWSN B131

If you wish to install your own computer:

- Download the ‘Personal Edition’: <https://unity3d.com/get-unity>
 - Make sure to get Unity version 2019.1.11f1
 - Make sure to add Android Build Support during installation.



Documentation

- Unity User Manual: <https://docs.unity3d.com/Manual/index.html>
- Scripting API: <http://docs.unity3d.com/ScriptReference/index.html>
- These pages should become your best friends.
- Also documentation on the OVR Utilities Plugin:
<https://developer.oculus.com/documentation/unity/unity-utilities-overview/>

Unity Official Scripting Videos: These also serve as a good introduction to C#.

- Beginner Scripting Playlist:
<https://www.youtube.com/watch?v=Z0Z7xc18CcA&list=PLX2vGYjWbI0S9-X2Q021GUtolTqbUBB9B>
- Intermediate Scripting Playlist:
<https://www.youtube.com/watch?v=HzlqrISbjjU&list=PLX2vGYjWbI0S8YpPPKKvXZayCjkKj4bUP>

Unity Basic Concepts

Project - The project contains all the elements that makes up the game, including models, assets, scripts, scenes, and so on.

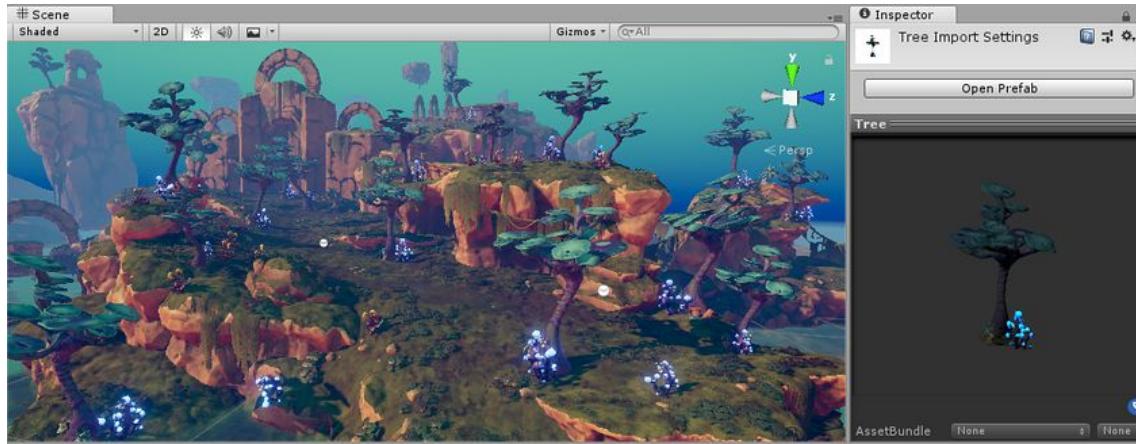
Scenes - A scene contains a collection of game objects that constitute the world that the player sees at any time.

Packages: A package is an aggregation of game objects and their associated metadata

Unity Basic Concepts (continued)

Prefabs: A prefab is a template for grouping various assets under a single header.

- Prefabs are used for creating multiple instances of a common object.
- For example, you may have a large number of copies of a single element (e.g., street lights, trees)
- Prefabs can be instantiated during runtime



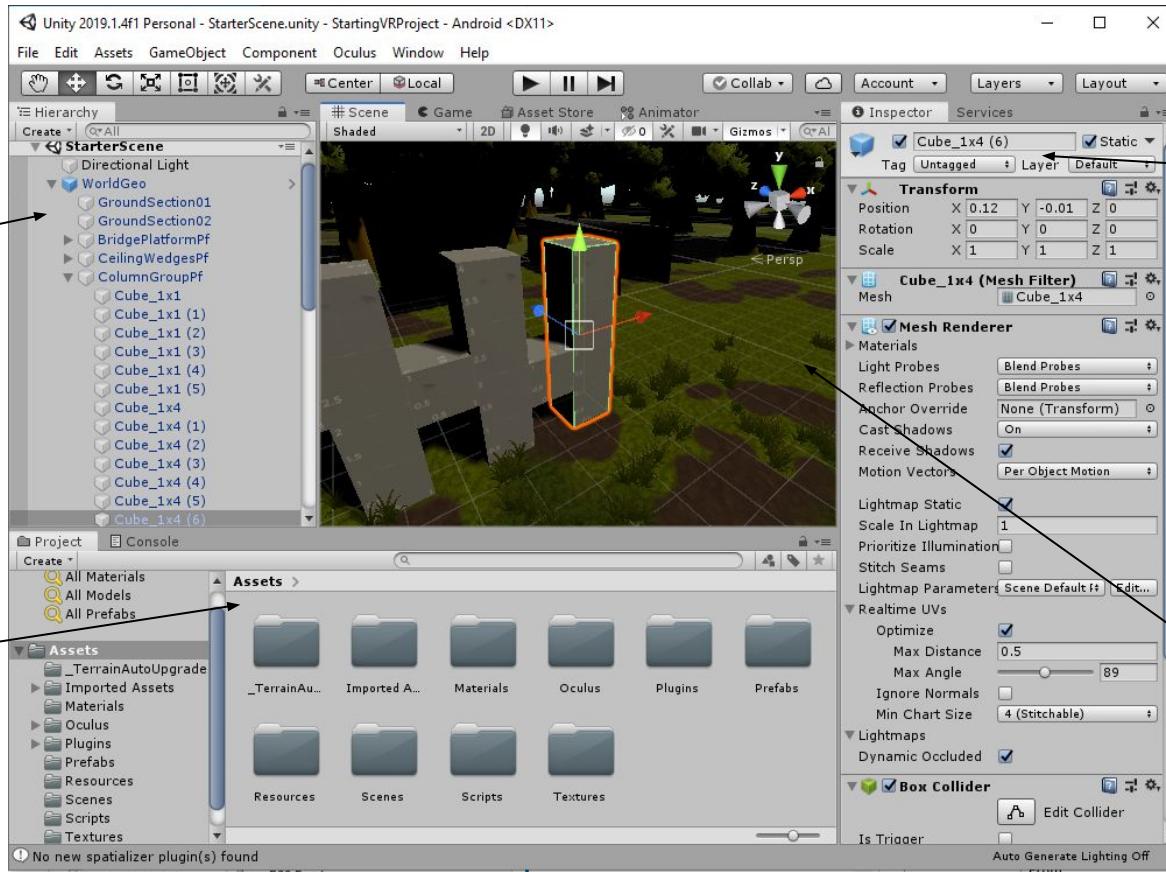
Overview of the Unity IDE:

Object hierarchy

Game object Inspector

Project assets

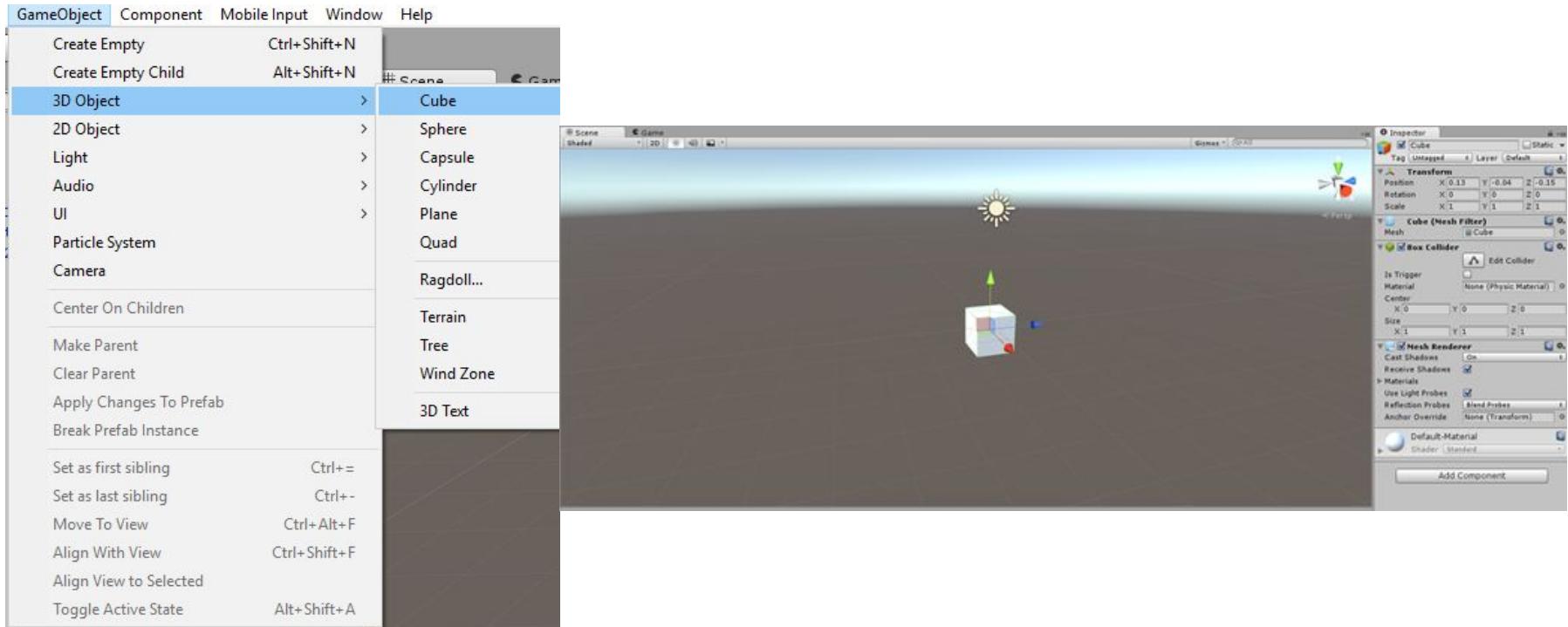
Scene View



Editor Camera Controls

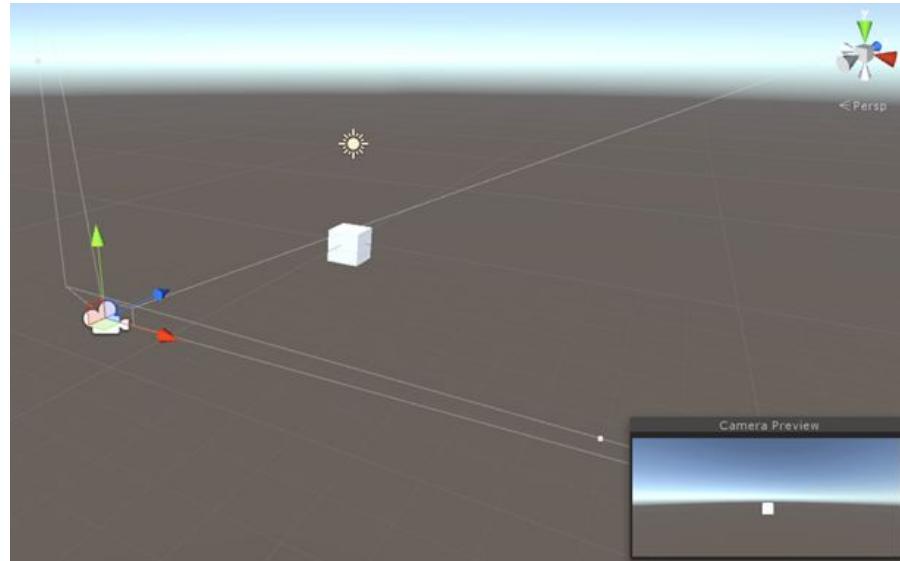
- Controls:
 - Alt + Left Click & Move: Rotate Camera
 - Alt + Right Click & Move (Or Scroll Up/Down): Zoom in and out
 - Alt + Middle Click & Move: Move camera up/down or left right
- Flythrough Mode:
 - Click and hold right mouse button and now you can use FPS-like controls to move around through the scene (WASD, Q/E to move up down).
- Unity Documentation:
<http://docs.unity3d.com/Manual/SceneViewNavigation.html>

Creating Geometry via the Unity Editor



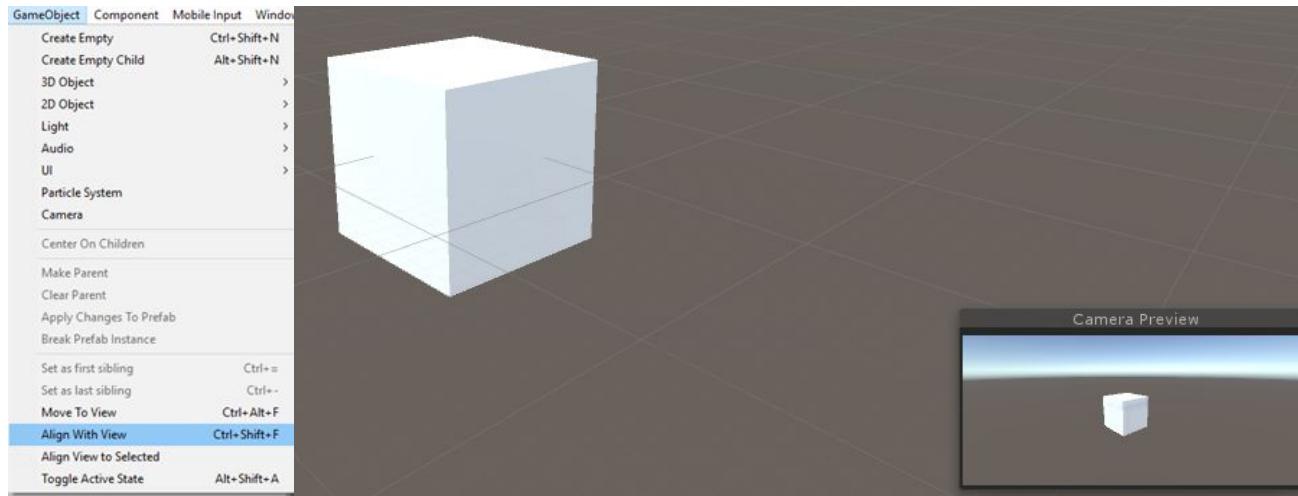
Setting Up The Scene Camera

- Do not confuse the scene camera with the editor camera.
- Unity scenes by default come with a “Main Camera.” Notice the tag of “MainCamera” in the inspector, this will be useful for accessing the camera from your scripts.
- “Camera Preview” box is useful to see what your camera can see.
- “Camera Preview” is what you will see when you hit Play.



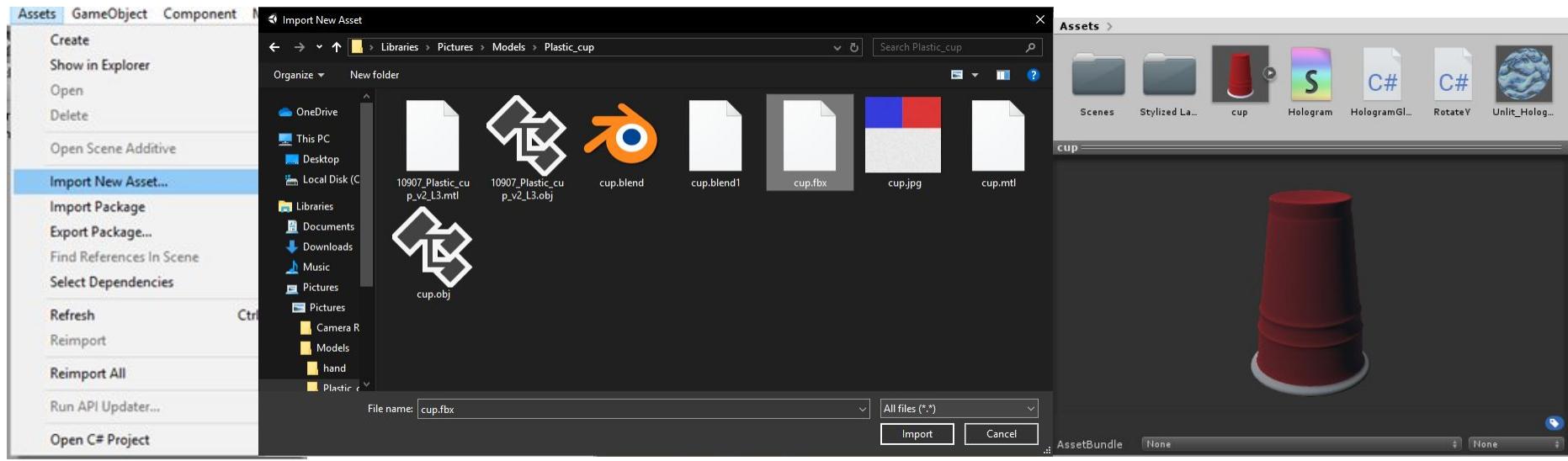
Setting Up The Scene Camera

- Moving the scene camera can be done manually by changing the position/rotation/scale in the Inspector.
- Or you can move the editor camera around as mentioned earlier, select the camera and align the camera to the view. Note that your camera should be selected before doing this.



Import External Objects

- Create and export an object from Maya/Blender/3ds Max as either an *.OBJ or a *.FBX. You can save this anywhere.
- Then import this asset into Unity. Unity will take care of everything for you.
- Alternatively, you can just save your *.OBJ or *.FBX inside the “Assets” folder.
 - You will need to right click on the folder it is in and click “Refresh” to get it to show up.
- Click and drag the object from the assets library into your scene hierarchy and it should now show up!



Game Objects

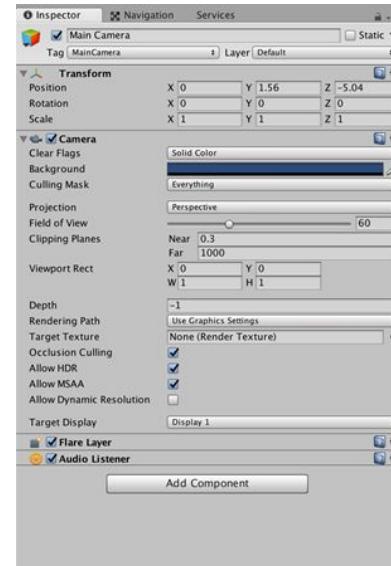
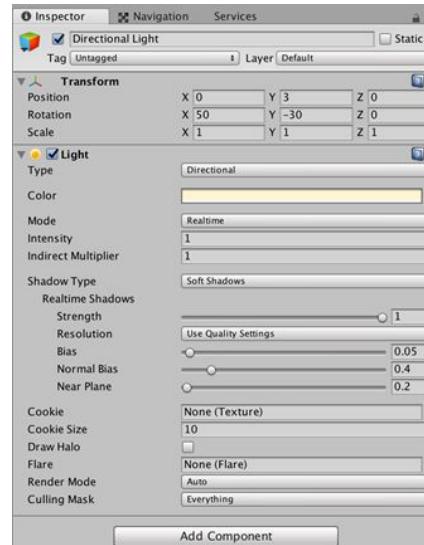
Game Objects: The *game objects* are all the “things” that constitute your scene.

- Light sources
- Audio sources
- Cameras
- Gameplay Logic
- User Interface
- Etc.

GameObject: <http://docs.unity3d.com/ScriptReference/GameObject.html>

Everything is a “GameObject”

- A *Game Object* does nothing on its own.
- *Game Objects* always have a *Transform* component which has a position/rotation/scale.
- Must add *Components* to the *Game Object* to give it some behavior.

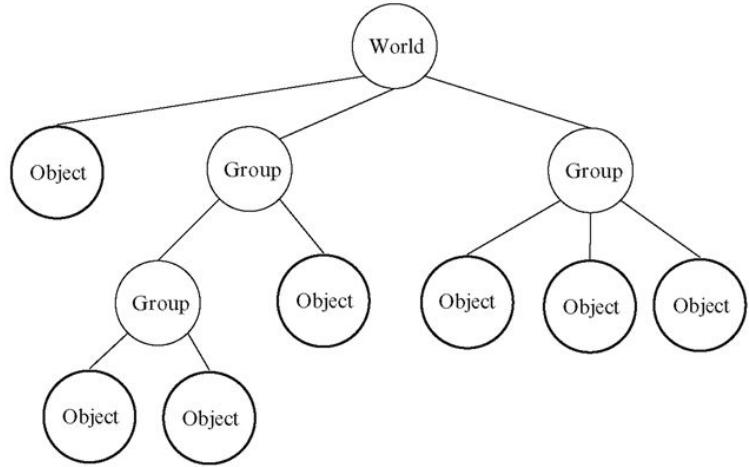


Scene graph

A scene graph is a collection of nodes in a graph or tree structure.

In Unity all tree nodes have only a single parent but may have many children.

Operations applied to a parent are applied to all its child nodes.



Components

Game Objects have *Components* to give it some behavior.

Many components already exist within Unity:

- Mesh Filter
- Mesh Renderer
- Rigidbody
- Colliders
- VideoPlayer

But you will also need create your own. These are your scripts that inherit from *MonoBehaviour*.



Scripts

- Many components already exist! But you will also need create your own.
These are your scripts that inherit from *MonoBehaviour*.
- Public variables will show up in the Inspector. A variable that is a Component can also be modified by the inspector

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

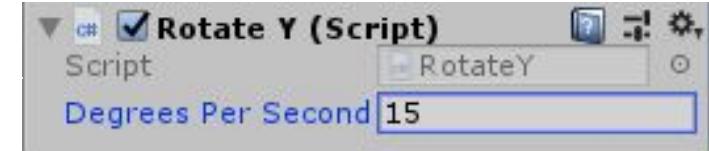
public class RotateY : MonoBehaviour
{
    // Degrees to rotate around Y-Axis
    public float rotationRate = 5.0f; // This value is overwritten by any changes to the script in the inspector!

    // Update is called once per frame
    void Update()
    {
        // Define the axis of rotation
        Vector3 axis = new Vector3(0, 1, 0);

        // Calculate the amount to rotate the object this frame
        float amountToRotate = rotationRate * Time.deltaTime;

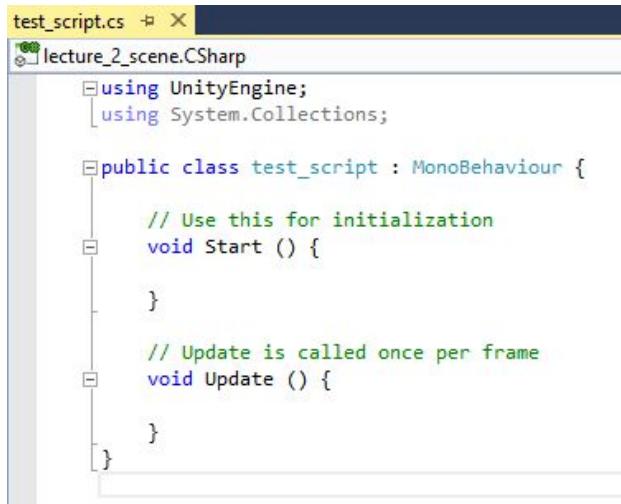
        // Access the transform component of this object and rotate
        this.transform.Rotate(axis, amountToRotate);
    }
}
```

overwrites



Adding Components to Game Objects

- GameObject: <http://docs.unity3d.com/ScriptReference/GameObject.html>
- MonoBehaviour: <http://docs.unity3d.com/ScriptReference/MonoBehaviour.html>
- Drag and Drop Script onto the GameObject in the “Inspector” or manually add it by going to:
 - Add Component > Scripts > YOUR_SCRIPT_NAME_HERE



```
test_script.cs ✘ X
lecture_2_scene.CSharp
using UnityEngine;
using System.Collections;

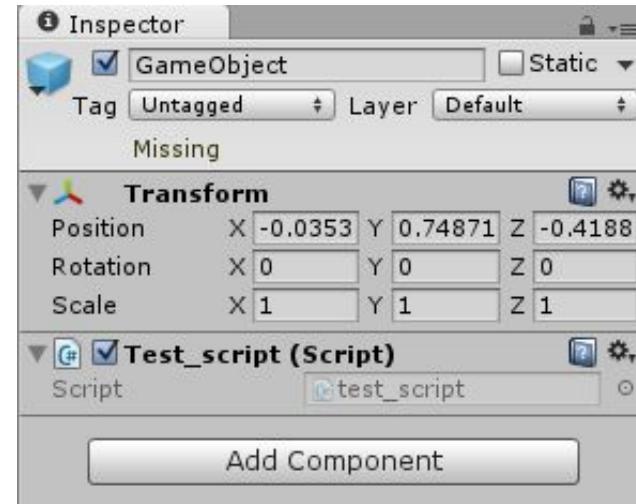
public class test_script : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

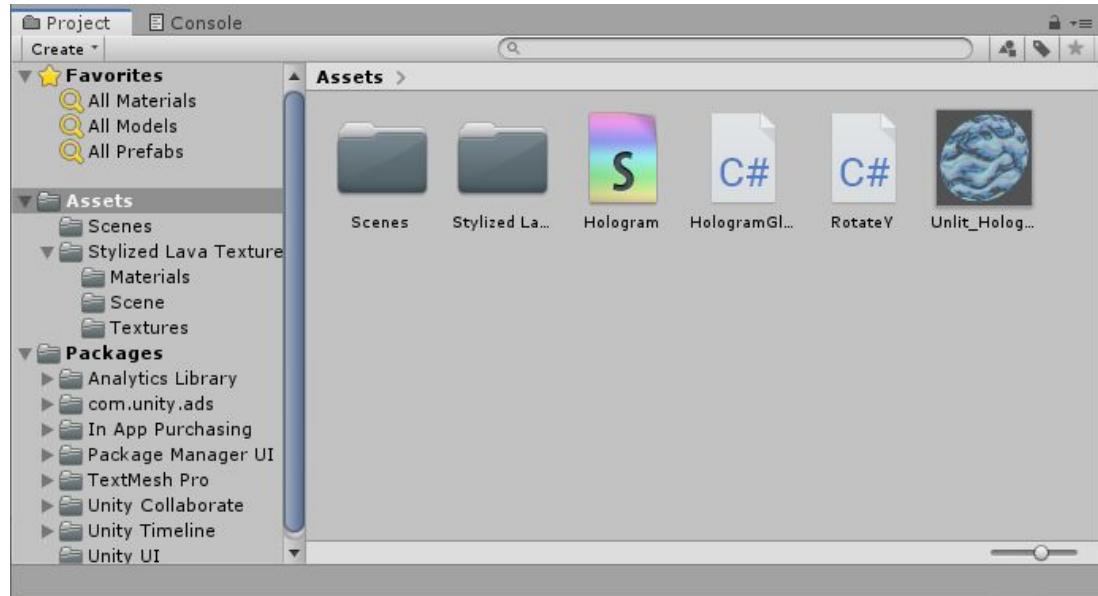
    }
}
```



Assets

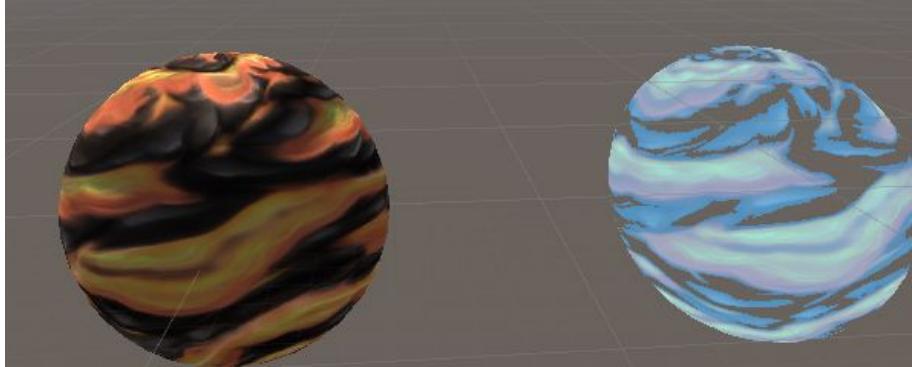
An asset is any resource that will be used as part of an object's component

- Scenes
- “Prefabs”
- Scripts
- Textures
- Animations
- Models
- Particles
- Sprites
- Etc.



Shading and Materials

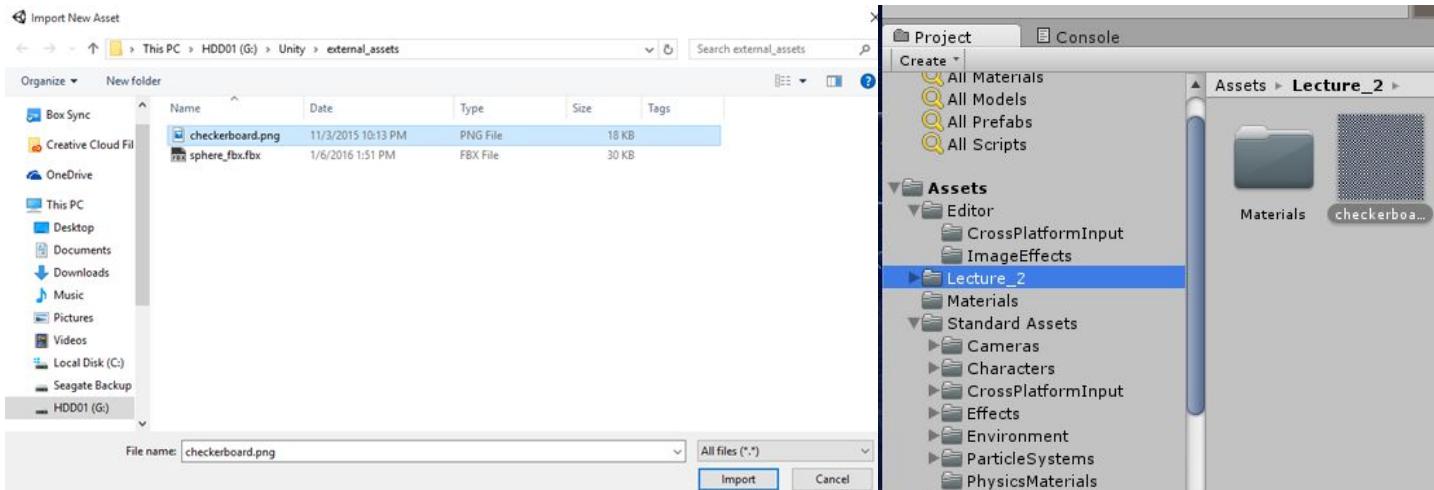
- Unity provides several built-in shaders
 - Unity Standard shader
 - Can also write your own shader
 - Shaders are written in Cg/HLSL and wrapped in ShaderLab
- Manual Shader Documentation: <http://docs.unity3d.com/Manual/ShadersOverview.html>
- Standard Shader Documentation:
<http://docs.unity3d.com/Manual/shader-StandardShader.html>
- Materials Documentation: <http://docs.unity3d.com/Manual/Materials.html>



The image shows two separate Unity Inspector windows. The top window is for a material named "Stylize_Lava", which uses a "Standard (Specular setup)" shader. It includes sections for Main Maps (Albedo, Specular, Smoothness, Normal Map, Height Map, Occlusion, Detail Mask), Emission, Tiling, Offset, Secondary Maps (Detail Albedo x2, Normal Map), Forward Rendering Options (Specular Highlights, Reflections checked), Advanced Options (Enable GPU Instancing, Double Sided Global Illumination), and a preview window showing a brown, textured surface. The bottom window is for a material named "Unitl_Hologram", which uses a "Unitl/Tutorial/Hologram" shader. It includes sections for Albedo Texture (Tiling X 1, Y 1; Offset X 0, Y 0), Tint Color (0.5, 0.22, 0.68), Transparency, Cutout Threshold, Distance, Amplitude, Speed, Amount, Render Queue (From Shader, 3000), and Double Sided Global Illumination. Both windows have tabs for "Inspector" and "Services".

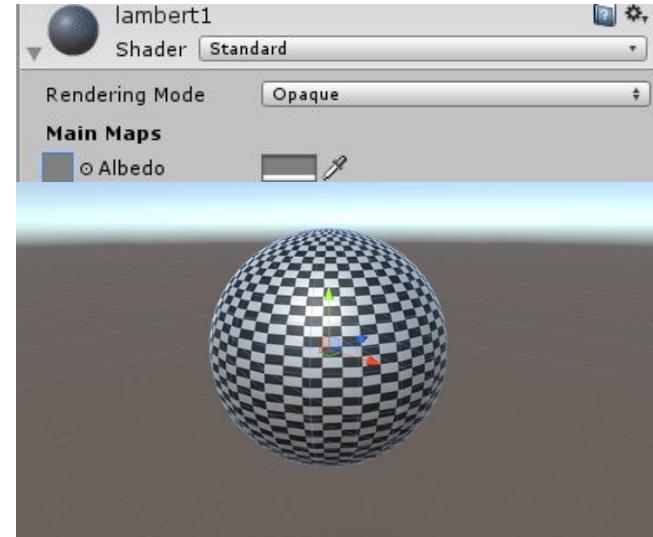
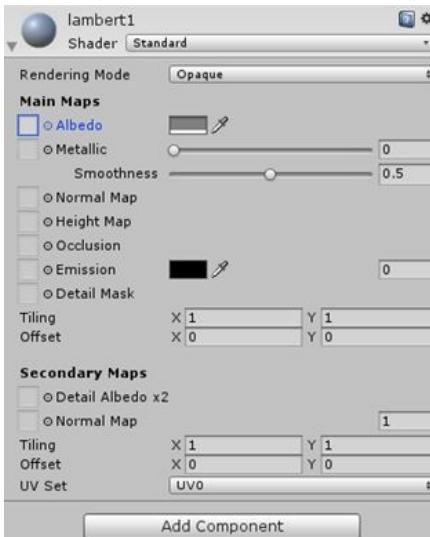
Importing Textures

- Process is the same as importing an external object. This time, instead of selecting an *.FBX or *.OBJ, select a *.PNG, *.JPG, etc. You can also place the images inside the Assets folder manually.



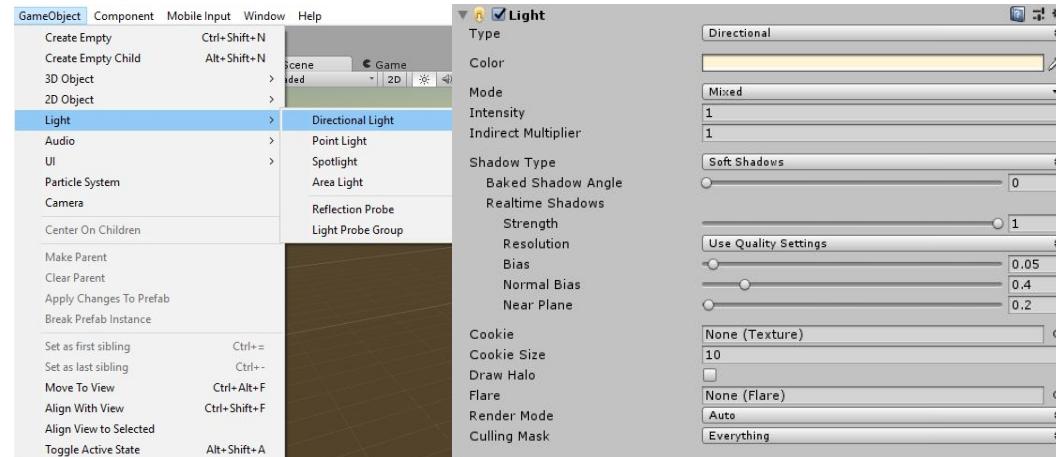
Using Textures

- Click on the object you imported in the scene hierarchy and expand the shader properties in the Inspector.
- Click and drag the imported texture onto the square next to “Albedo” and your object should now have a texture on it.



Lighting

- Lighting Documentation: <http://docs.unity3d.com/Manual/Lighting.html>
- Global Illumination Documentation:
<http://docs.unity3d.com/Manual/GlobalIllumination.html>
- Lighting is accomplished with the “Light” component.
 - Directional
 - Point
 - Spot
 - Area (baked only)



Scripting in Unity

Scripting in Unity is done in C#

Scripts are an example of a component that is associated with a game object

The skeletal structure of a typical script called *MyGameObject* is shown below:

Script example

```
using UnityEngine;           // basic Unity-Engine objects
using System.Collections; // basic structures (ArrayList, HashTable,...)

public class MyGameObject : MonoBehaviour {
    void Start () {
        // ... initializations (like a constructor in Java)
    }
    void Update () {
        // ... insert code to be repeated every update cycle
    }
}
```

Fundamental Classes: MonoBehaviour

When you create a script in Unity, Unity creates a class that extends MonoBehaviour.

Contains functions and events that are available to standard scripts attached to Game Objects

- Awake, Start, Update, FixedUpdate
- OnCollisionEnter, OnCollisionStay, OnCollisionExit
- GetComponent, SendMessage, BroadcastMessage
- Destroy, Instantiate

For a full list of methods and documentation, see:

<https://docs.unity3d.com/ScriptReference/MonoBehaviour.html>

Fundamental Classes: GameObject

GameObject: A generic type from which all game objects are derived. This corresponds to anything that could be placed in your scene hierarchy.

GameObjects have an associated *name* and *tag*. You can find other gameObjects with *Find*, *FindWithTag*, *FindGameObjectsWithTag*, etc.

Here is an example of how to obtain the main camera reference by its name:

```
GameObject camera = GameObject.Find ( "Main Camera" );
```

Suppose that we assign the tag “Player” with the player object and “Enemy” with the various enemy objects. We could access the object(s) through the tag using the following commands:

```
GameObject player = GameObject.FindWithTag ( "Player" );
GameObject [] enemies = GameObject.FindGameObjectsWithTag ( "Enemy" );
```

Fundamental Classes: Transform

Transform: Every *game object* in Unity is associated with an object called its transform.

This object stores the position, rotation, and scale of the object. You can use the transform object to query the object's current position (*transform.position*) and rotation (*transform.eulerAngles*)

Vector3

Structure in Unity for representing 3D vectors and points.

This structure is used throughout Unity to pass 3D positions and directions around. It also contains functions for doing common vector operations.

Other classes can be used to manipulate vectors and points as well. For example the Quaternion and the Matrix4x4 classes are useful for rotating or transforming vectors and points.

Common methods: Cross, Dot, Normalize, Lerp, Reflect, Distance

For more information, see the documentation:

<https://docs.unity3d.com/ScriptReference/Vector3.html>

Quaternion

Quaternions are used internally by Unity to represent rotations.

There are some advantages to using quaternions over euler angles (gimbal lock, can be interpolated easily, etc)

Have x,y,z,w components and are non-commutative. Likely will never need to modify these components individually.

Instead use these to create/manipulate Quaternions: Quaternion.LookRotation, Quaternion.Angle, Quaternion.Euler, Quaternion.Slerp, Quaternion.FromToRotation, Quaternion.identity

For more information, see: <https://docs.unity3d.com/ScriptReference/Quaternion.html>

Matrix4x4

Structure for a 4x4 transformation matrix

Can perform translation, rotation, scale, shear, and perspective transformations using homogeneous transformations.

Column major: for the expression ***mat[a, b]***, ***a*** refers to the row index, while ***b*** refers to the column index

In Unity, Matrix4x4 is used by several Transform, Camera, Material and GL functions.

Common methods/properties: determinant, inverse, transpose, LookAt, Ortho, Perspective, Rotate, Scale, Translate, TRS

For more information, see: <https://docs.unity3d.com/ScriptReference/Matrix4x4.html>

Accessing Components:

It is often desirable to modify the values of components at run time.

Unity defines class types for each of the possible components, and you can access and modify this information from within a script.

To access public variables/methods from a component, use *GetComponent*.

Example:

```
// Get rigidbody component of this game object
Rigidbody rb = GetComponent <Rigidbody>();

// change this body's mass
rb.mass = 10f;
```

Accessing Members of Other Scripts

Often, game objects need to access members variables in other game objects.

Can use *GetComponent* to access public variables/methods in other scripts.

```
public class PlayerController : MonoBehaviour {
    public void DecreaseHealth () { ... } // decrease player 's health
}

public class EnemyController : MonoBehaviour {
    public GameObject player; // the player object
    void Start () {
        GameObject player = GameObject.Find( "Player" );
    }
    void Attack () { // inflict health loss on player
        player.GetComponent<PlayerController>().DecreaseHealth();
    }
}
```

Colliders and Triggers:

Some events are generated by the user (e.g., input), some occur at regular time intervals (e.g., `Update()`), and finally others are generated within the game itself.

Typically, colliders are physical objects that should not overlap, whereas triggers are invisible barriers that send a signal when crossed.

There are various event functions for detecting when an object enters, stays within, or exits, collider/trigger region. These include, for example:

- For colliders: `void OnCollisionEnter()`,`void OnCollisionStay()`,`void OnCollisionExit()`
- For triggers: `void OnTriggerEnter()`,`void OnTriggerStay()`,`void OnTriggerExit()`

Example: Rotate script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RotateYFinal : MonoBehaviour
{
    // Degrees to rotate around Y-Axis per second
    public float rotationRate = 5.0f;

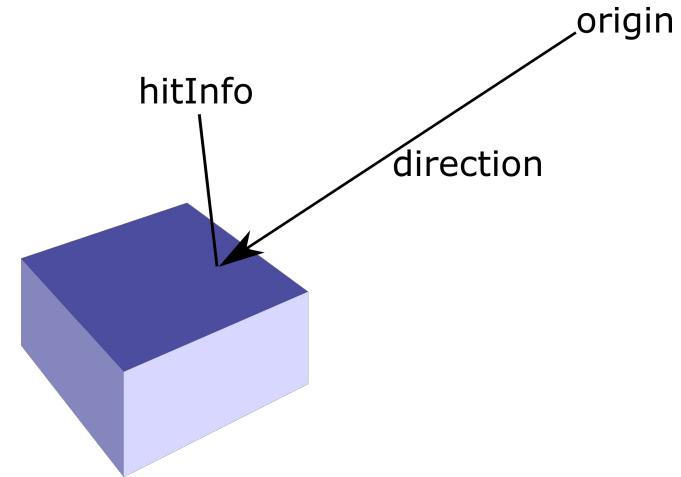
    // Update is called once per frame
    void Update()
    {
        // Define the axis of rotation
        Vector3 axis = new Vector3(0, 1, 0);
        // Equivalently you could use Vector3.up

        // Calculate the amount to rotate the object this frame
        float amountToRotate = rotationRate * Time.deltaTime;

        // Access the transform component of this object and rotate
        this.transform.Rotate(axis, amountToRotate);
    }
}
```

Raycasting:

```
public static bool Raycast(  
    Vector3 origin,  
    Vector3 direction,  
    out RaycastHit hitInfo,  
    float maxDistance,  
    int layerMask,  
    QueryTriggerInteraction queryTriggerInteraction  
) ;
```



Casts a ray, from point *origin*, in direction *direction*, of length *maxDistance*, against all colliders in the Scene.

You may optionally provide a LayerMask, to filter out any Colliders you aren't interested in generating collisions with.

Documentation: <https://docs.unity3d.com/ScriptReference/Physics.Raycast.html>

Oculus Utilities for Unity

OVRCameraRig is a Component that controls stereo rendering and head tracking. It maintains three child “anchor” Transforms at the poses of the left and right eyes, as well as a virtual “center” eye that is halfway between them.

This Component is the main interface between Unity and the cameras. It is attached to a prefab that makes it easy to add comfortable VR support to a scene.

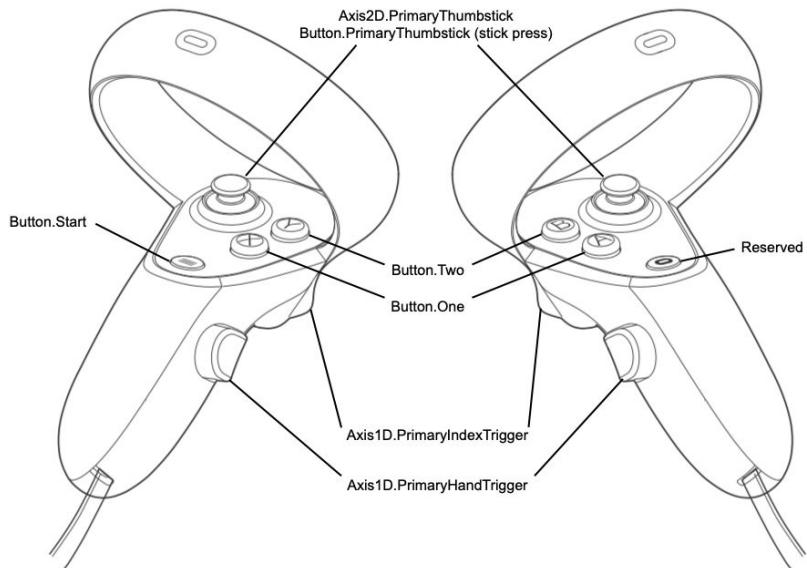
Public Members:

1. Updated Anchors - Allows clients to filter the poses set by tracking. Used to modify or ignore positional tracking.

Game Object Structure:

1. TrackingSpace - A Game Object that defines the reference frame used by tracking. You can move this relative to the OVRCameraRig for use cases in which the rig needs to respond to tracker input. For example, OVRPlayerController changes the position and rotation of TrackingSpace to make the character controller follow the yaw of the current head pose.

OVRInput



Virtual Reality and Its Applications

Pavan Kumar B N

Department of Computer Science and Engineering

Virtual Reality and Its Applications

What is Virtual Reality?

Inducing **targeted behavior** in an **organism** by using **artificial sensory stimulation**, while the organism has little or no **awareness** of the interference.

The organism is having an “experience” that was designed by the creator

life form such as a fruit fly, cockroach, fish, rodent, or monkey

Senses of the organism become co-opted, and their inputs are enhanced by artificial stimulation

“fooled” into feeling present in a virtual world. unawareness leads to a sense of presence





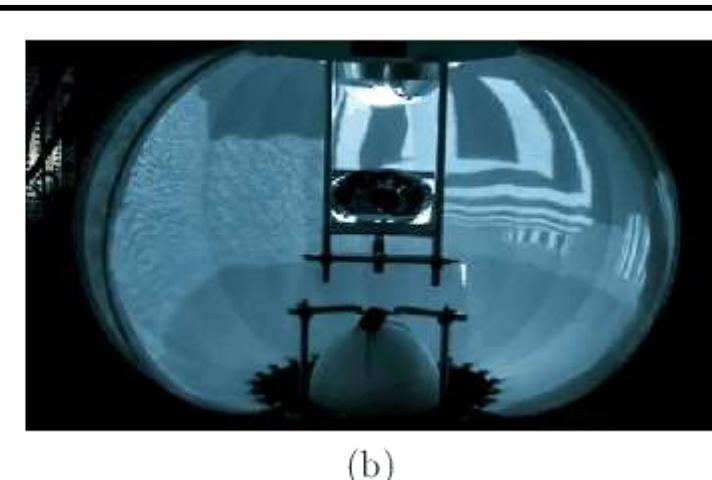
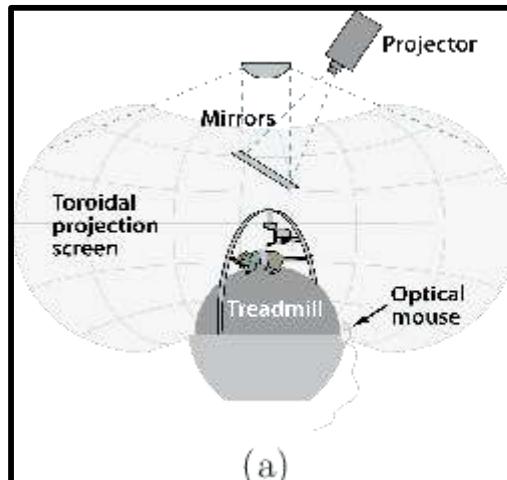
Virtual Reality and Its Applications

What is Virtual Reality?



In the Birdly experience from the Zurich University of the Arts, the user, wearing a VR headset, flaps his wings while flying over virtual San Francisco

An experimental setup used by neurobiologists at LMU Munich to present visual stimuli to a gerbil while it runs on a spherical ball that acts as a treadmill

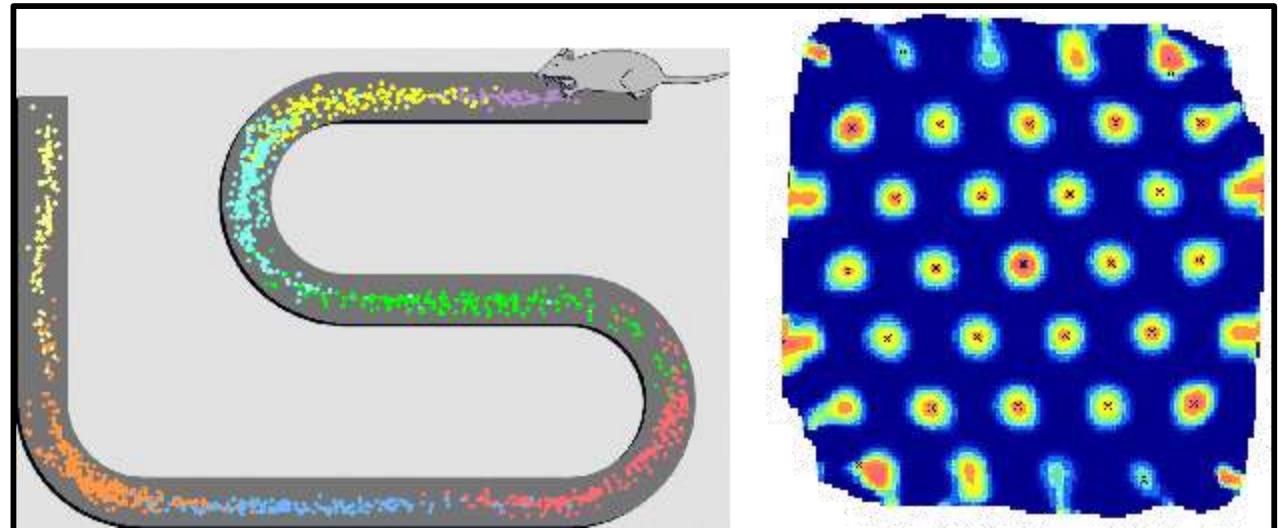


Virtual Reality and Its Applications

Testing the boundaries

- Listening to music through headphones
- Watching a movie at a theater
- Anyone with modern VR headset and enjoying a session
- Portrait or painting on the wall
- Reading a novel

spatial firing patterns of eight place cells in a rat brain as it runs back and forth along a winding track

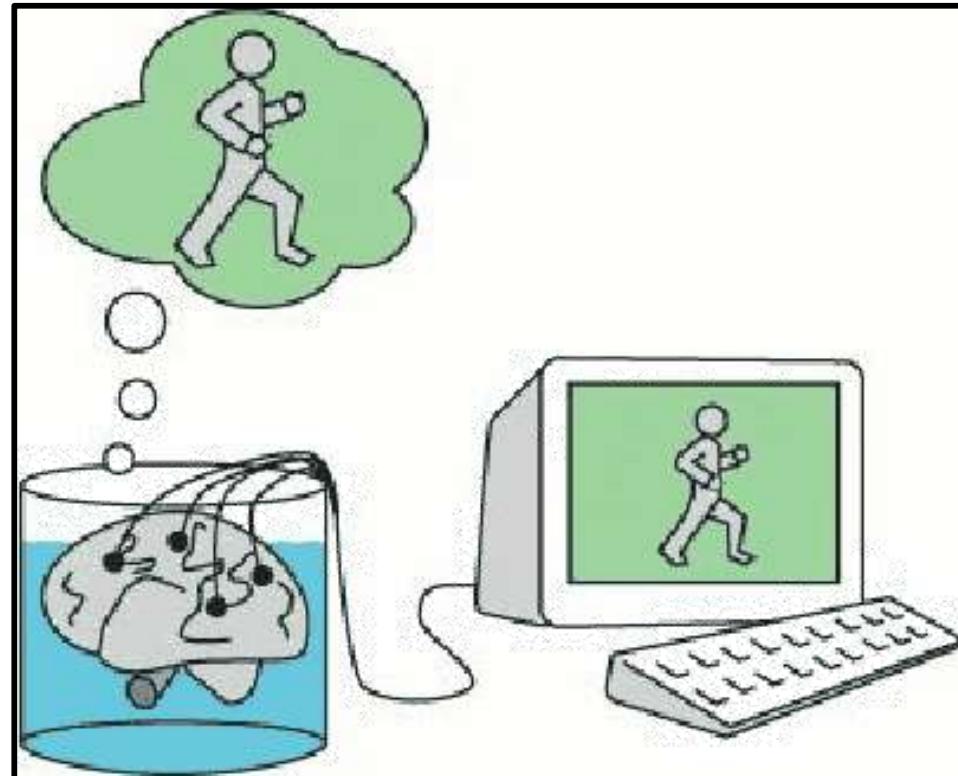


Virtual Reality and Its Applications

Who is the fool

- Neural structures composed of place cells are formed that encode spatial information about its surroundings
- Each place cell is activated precisely when the organism returns to a particular location that is covered by it
- Our brains may form place cells for places that are not real!
- This is a clear indication that VR is fooling our brains, at least partially

A VR thought experiment: The brain in a vat, by Gilbert Harman



Virtual Reality and Its Applications

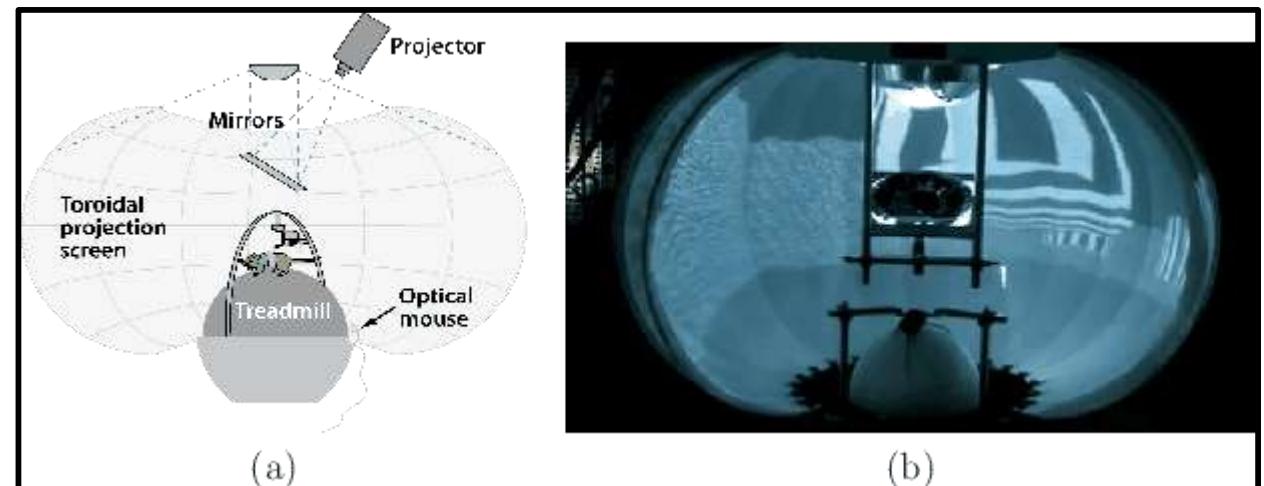
Interactivity

- Most VR experiences involve another crucial component: interaction.
- Does the sensory stimulation depend on actions taken by the organism? If the answer is “no”, then the VR system is called open-loop;
- If the organism has partial control over the sensory stimulation, which could vary as a result of body motions, including eyes, head, hands, or legs, then we are in a closed-loop

Virtual Reality and Its Applications

First Person Vs Third Person

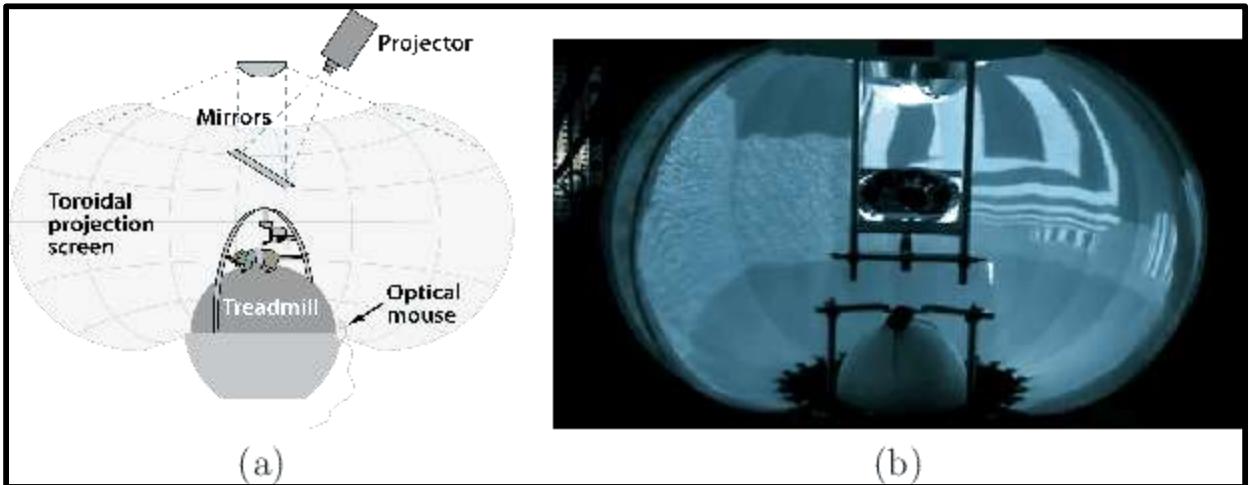
- Subjects have First Person Experiences
- Observers have third-person experience
- Switching back and forth between being the Subjects and observers while evaluating and refining their VR is a **bad Idea!**



Virtual Reality and Its Applications

Synthetic Vs Captured

- We may program a *synthetic* world, which is completely invented from geometric primitives and simulated physics
- The world may be *captured* using modern imaging techniques.
- Factor affecting captured
 - What happens when the user changes her head position and viewpoint?
 - What are their facial expressions while wearing a VR headset?
 - What can we infer about their emotional state?
 - Do we need to know their hand gestures?
 - Are their eyes focused on me?



Virtual Reality and Its Applications

Modern VR Experiences – Video Games



(a)



(b)



(c)



(d)

Virtual Reality and Its Applications

Modern VR Experiences – Immersive cinema



In VR, viewers can look in any direction, and perhaps even walk through the scene.

Some questions for the makers

- What should they be allowed to do?
- How do you make sure they do not miss part of the story?
- Should the story be linear, or should it adapt to the viewer's actions?
- Should the viewer be a first-person character in the film, or a third-person observer who is invisible to the other characters?
- How can a group of friends experience a VR film together?
- When are animations more appropriate versus the capture of real scenes?

Virtual Reality and Its Applications

Modern VR Experiences – Telepresence

- The first step toward feeling like we are somewhere else is capturing a panoramic view of the remote environment
- Simple VR apps that query the Street View server directly enable to user to feel like he is standing in each of these locations, while easily being able to transition between nearby locations
- By connecting panoramic cameras to robots, the user is even allowed to move around in the remote environment



(a)



(b)



(a)



(b)

Virtual Reality and Its Applications

Modern VR Experiences – Virtual societies



Virtual Reality and Its Applications

Modern VR Experiences – Empathy



In *Clouds Over Sidra*, 2015, offered a first-person perspective on the suffering of Syrian refugees

The Machine to Be Another



Virtual Reality and Its Applications

Modern VR Experiences – Education



A flight simulator in use by the US Air Force



A tour of the Nimrud palace

Virtual Reality and Its Applications

Modern VR Experiences – Advertisement



VReddo VR
Virtual Reality Classrooms

Virtual Reality and Its Applications

Modern VR Experiences – Virtual prototyping



Virtual Reality and Its Applications

Modern VR Experiences – Health care



Toward
convenience
and
portability



To go some
where to
see



See it from
your home



Being able
to carry
anywhere





What is Virtual Reality (VR)?

A high-end **user interface** that involves **real-time simulation** and **interactions** through **multiple sensorial channels** for a **targeted behavior** in an **organism**.



Visuals

So vibrant that they are eventually indistinguishable from the real world



Interactions

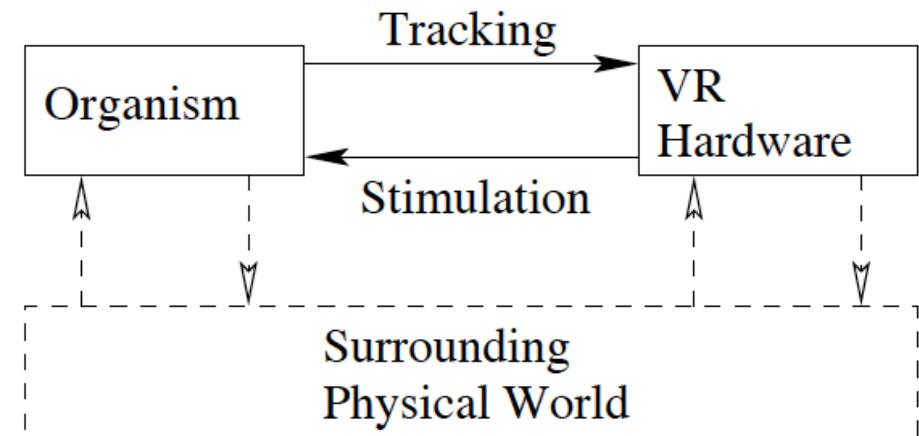
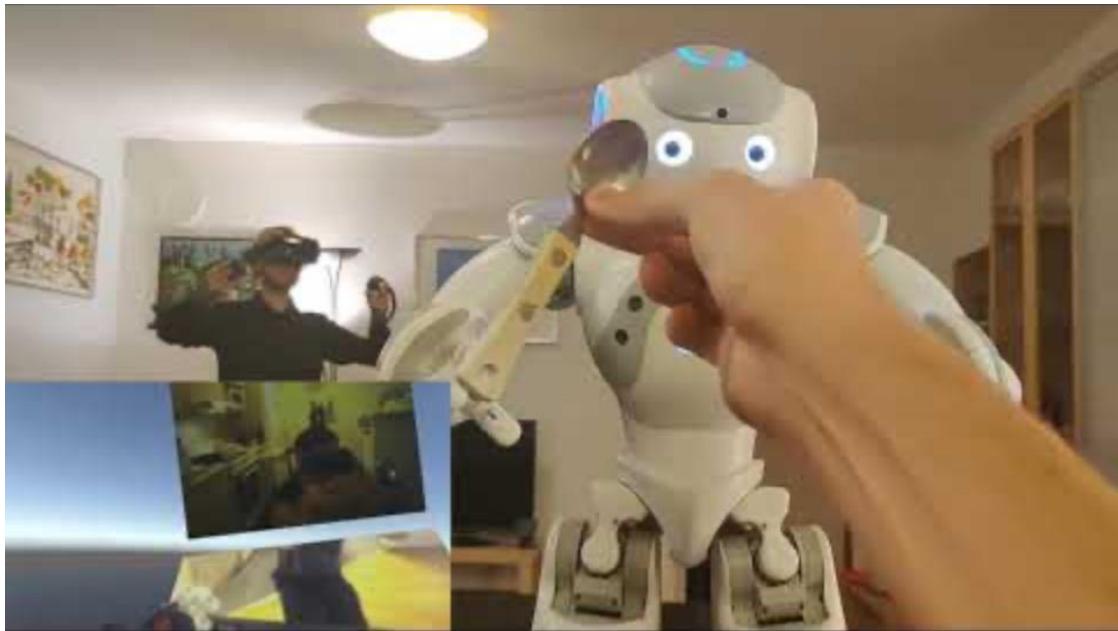
So intuitive that they become second nature

Sounds

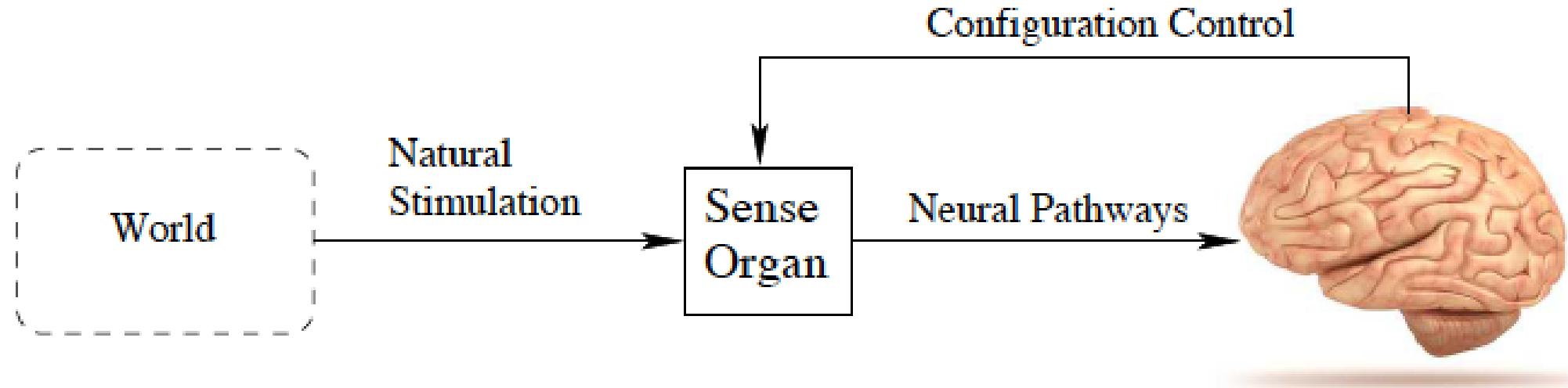
So accurate that they are true to life



VR System



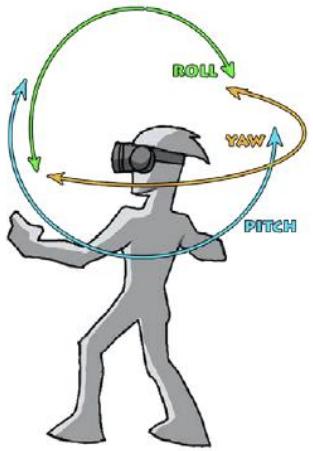
A third-person perspective



Sensors and Sense Organs

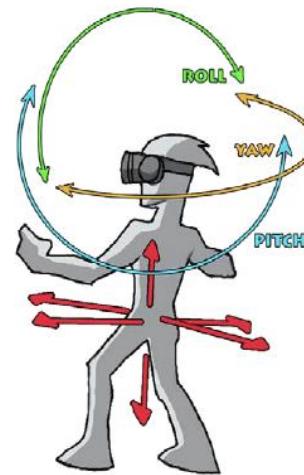
DEGREES OF FREEDOM (DOF)

3 degrees of freedom (3-DoF)



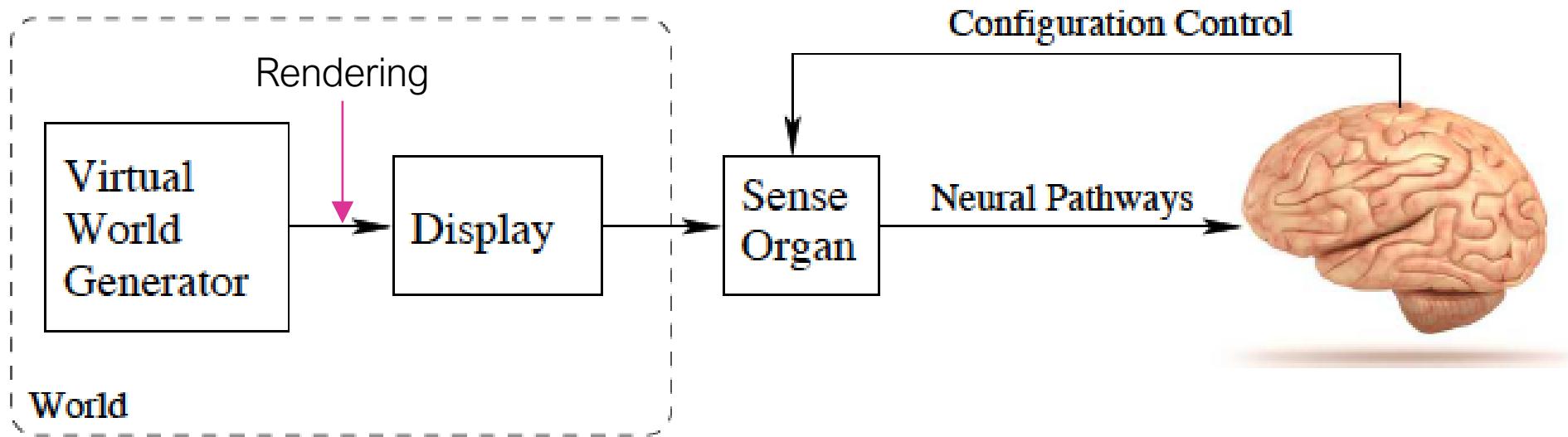
- “In which direction am I looking”
- Detect rotational head movement
- Look around the virtual world from a fixed point

6 degrees of freedom (6-DoF)



- “Where am I and in which direction am I looking”
- Detect rotational movement and translational movement
- Move in the virtual world like you move in the real world

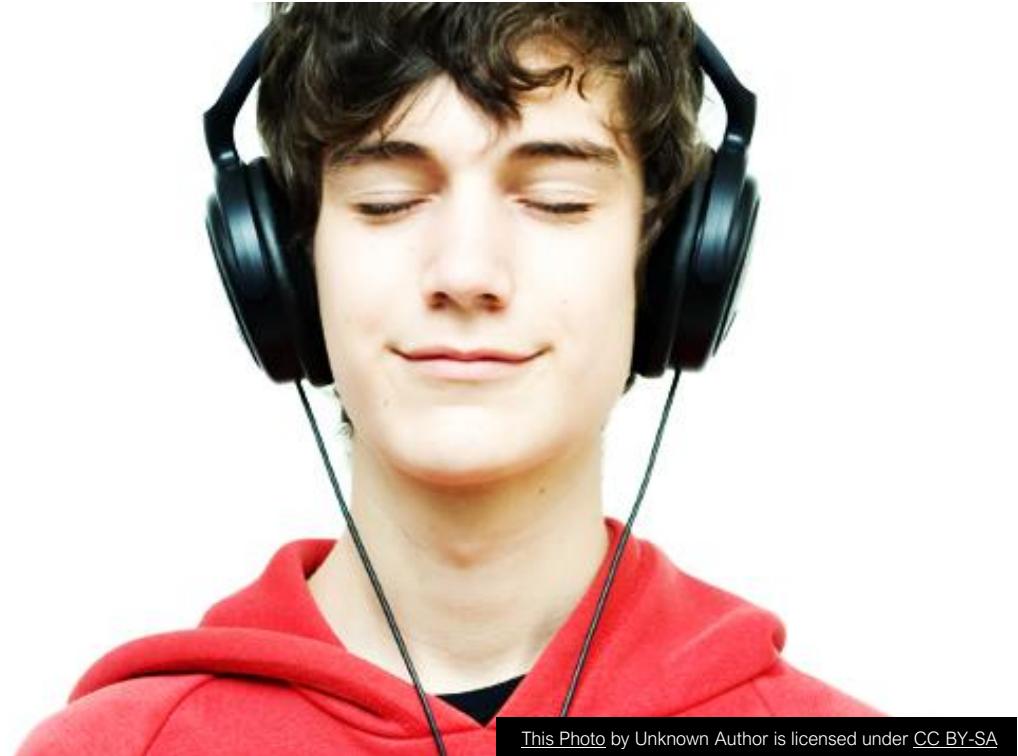
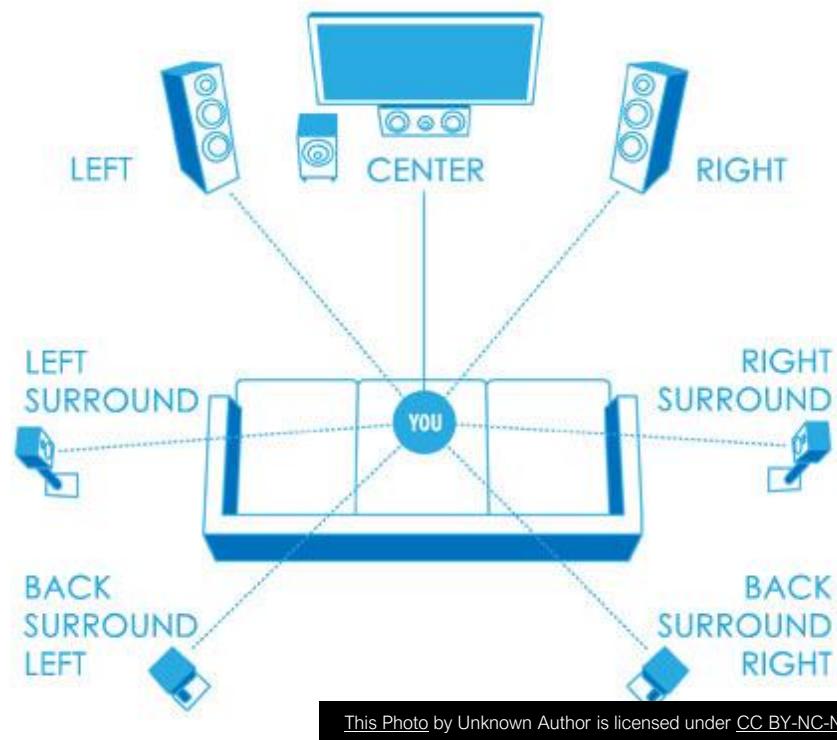
An illustration |



Artificial simulation

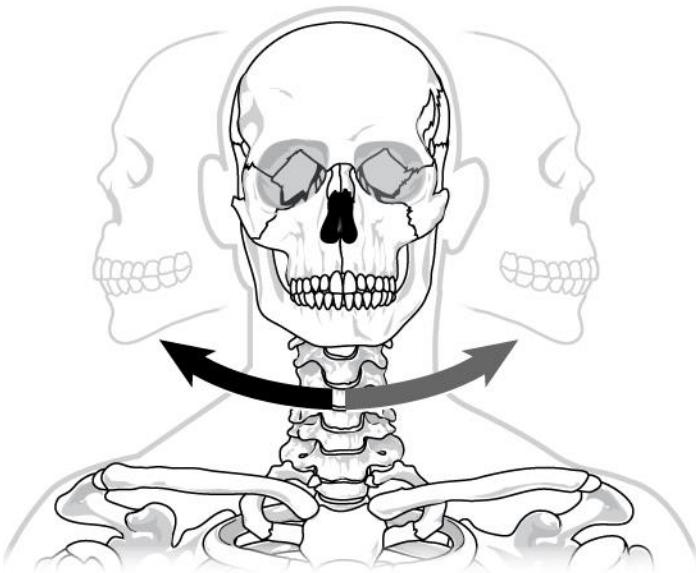
World-fixed vs. User-fixed

Trend of having to go somewhere for an experience, to having it in the home, and then finally to having it be completely portable



Use-case 1: Aural

Key Differences



[This Photo](#) by Unknown Author is licensed under [CC BY](#)

Distance of the source (or stimuli) from the ears

Power requirements to generate the stimuli of equal strength

Degree of privacy

No of users having the immersive experience

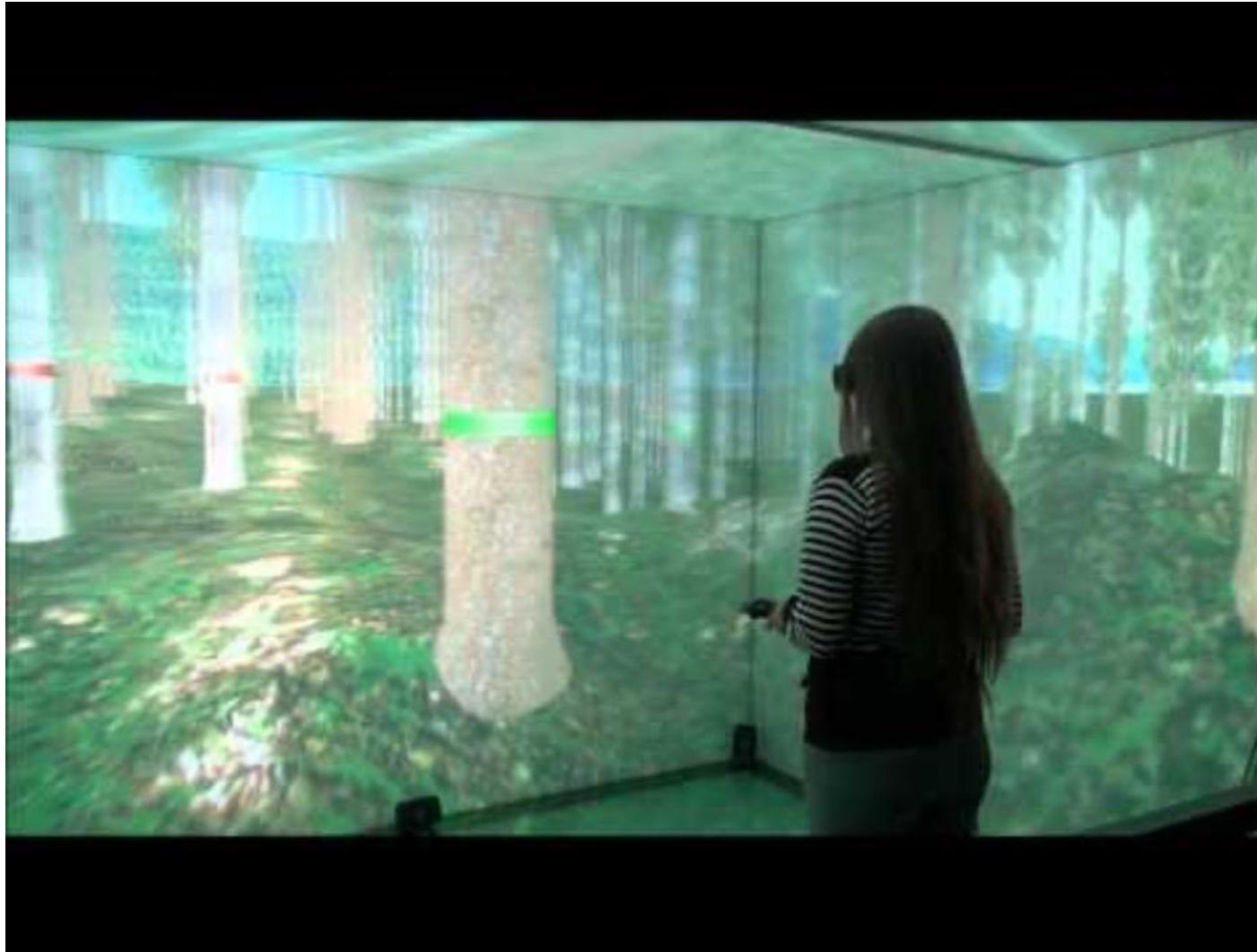
Cost of the system

Wearing electronics

How are these factors going to affect a VR system design?

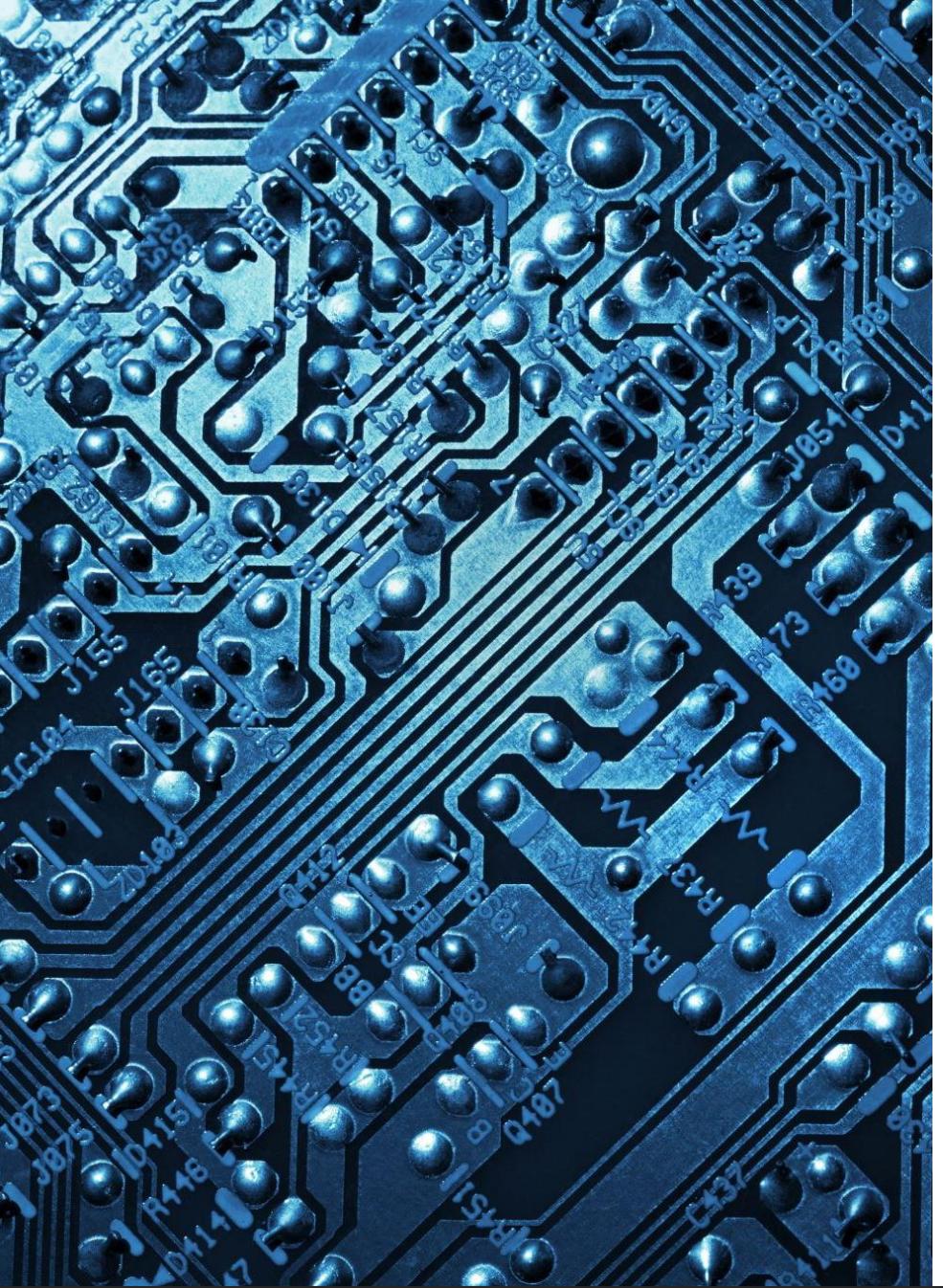


Use-case 2:Visual

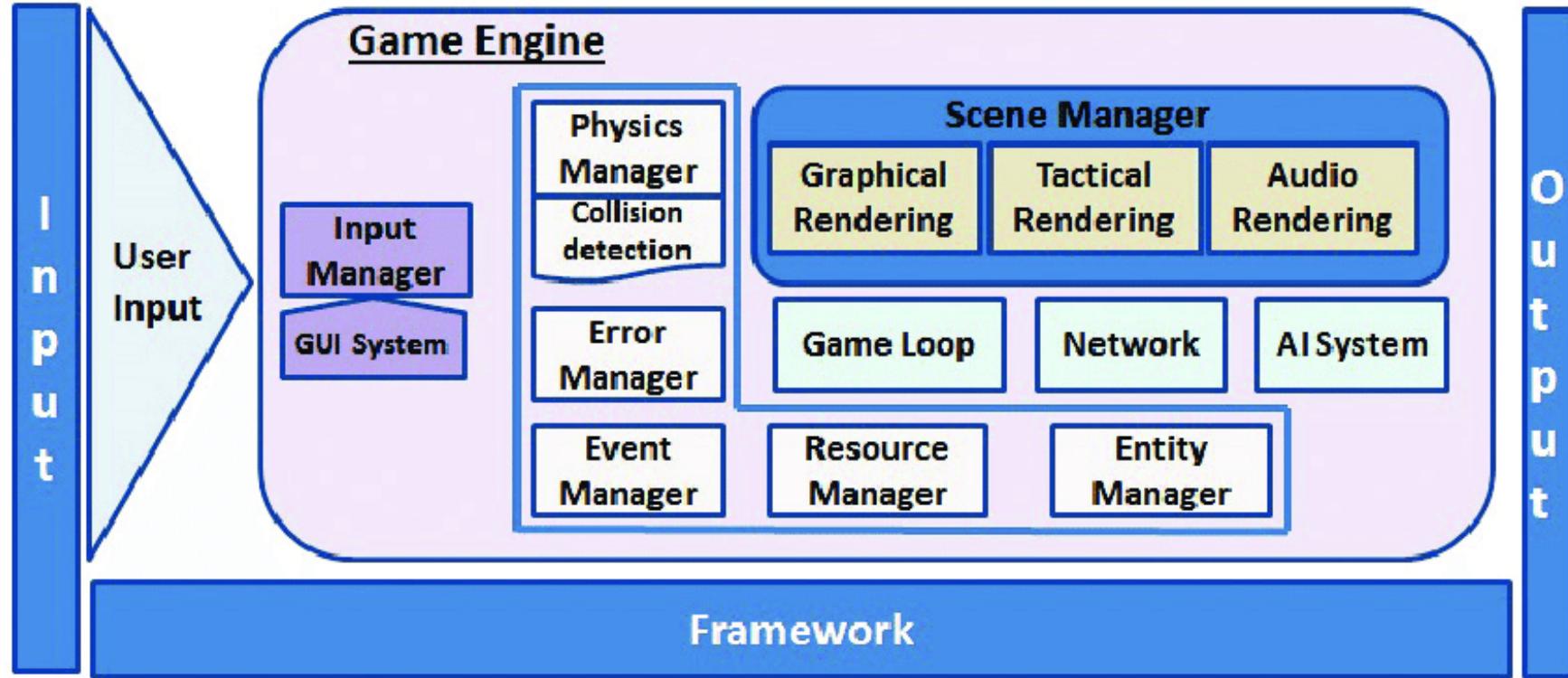


An Example

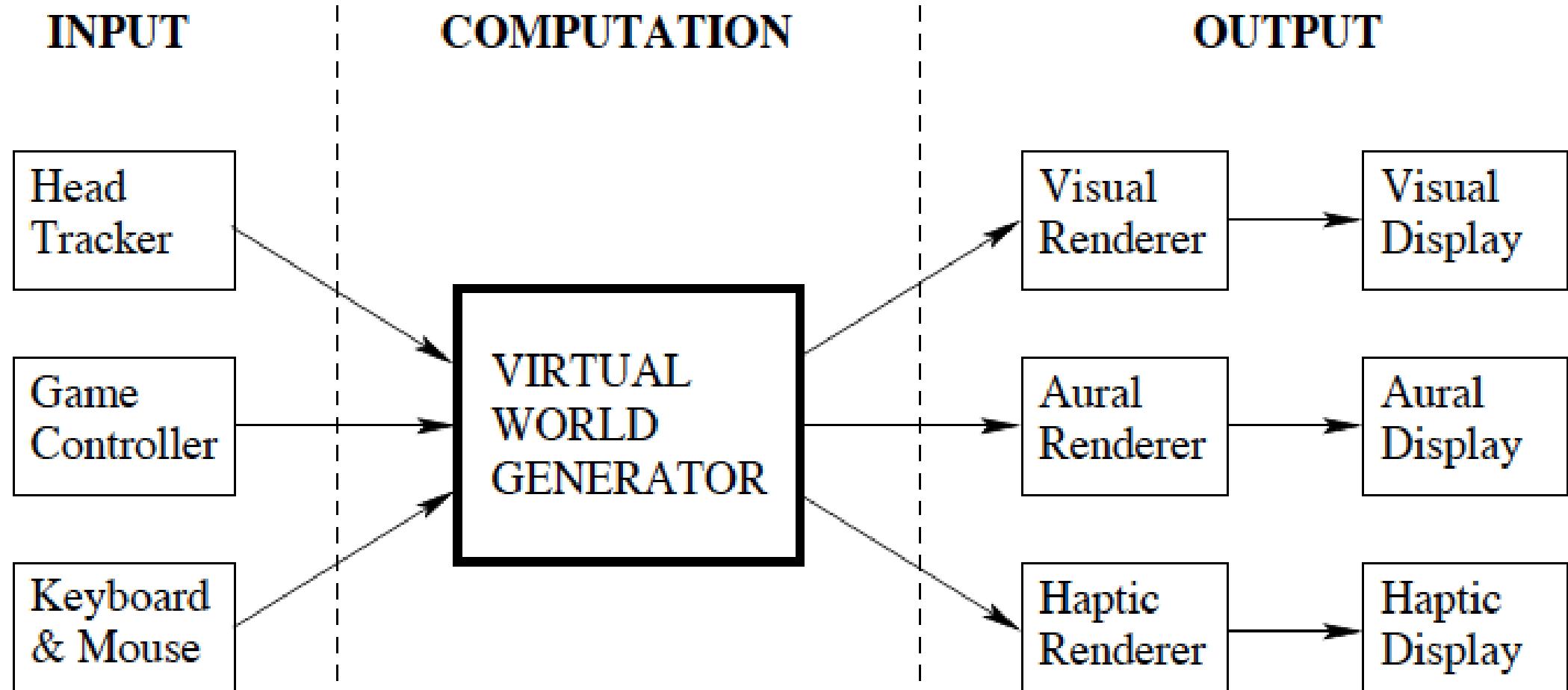
Virtual CAVE



Hardware Software Framework



Game Engine



Software Stack

Drivers

- System level interactions

Operating System

- To better manage the peripheral

Middleware

- Processing engine (Game, Audio, Video)

Tools and SDKs

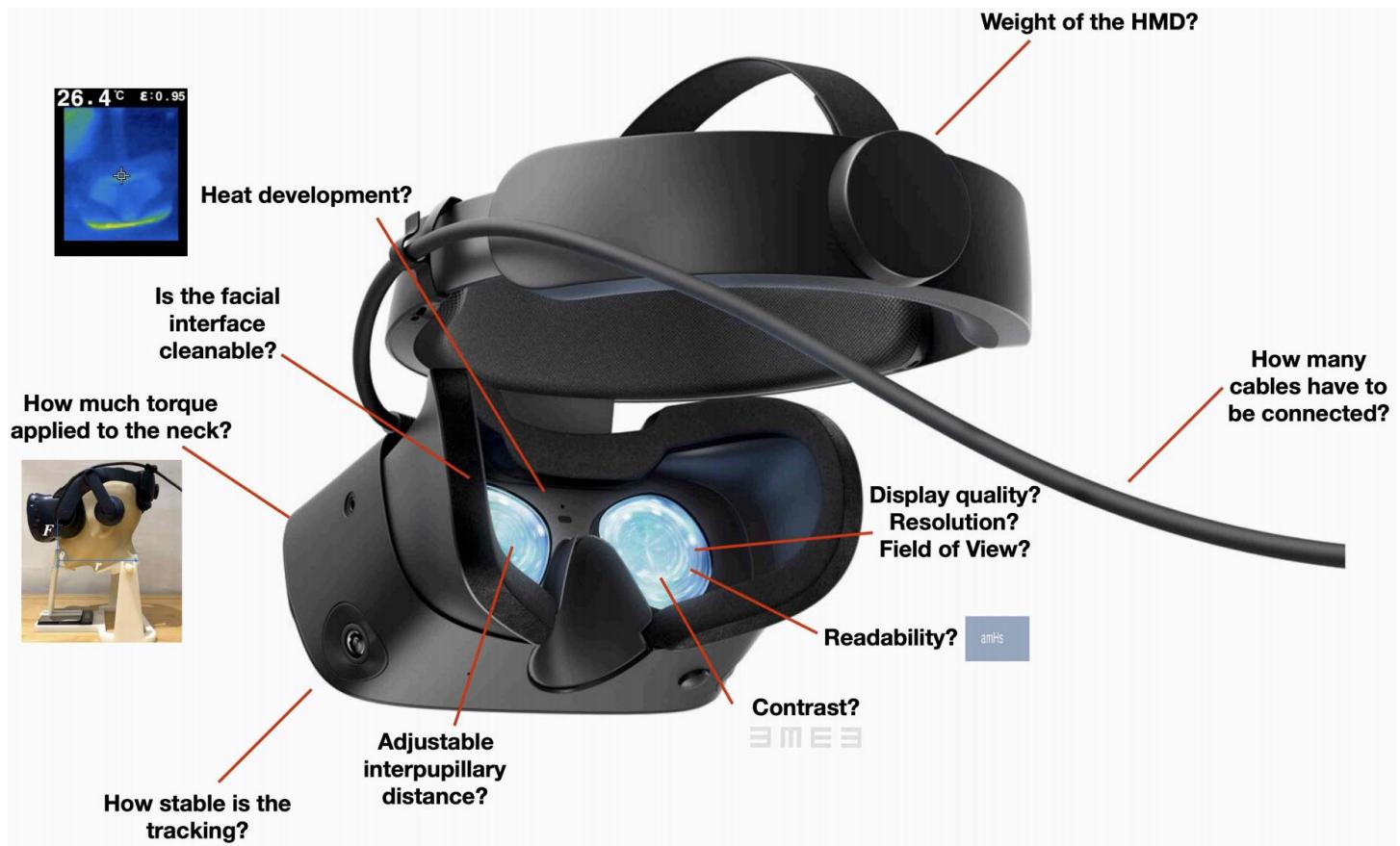
- To generate the content

Head-Mounted Display (HMDs)

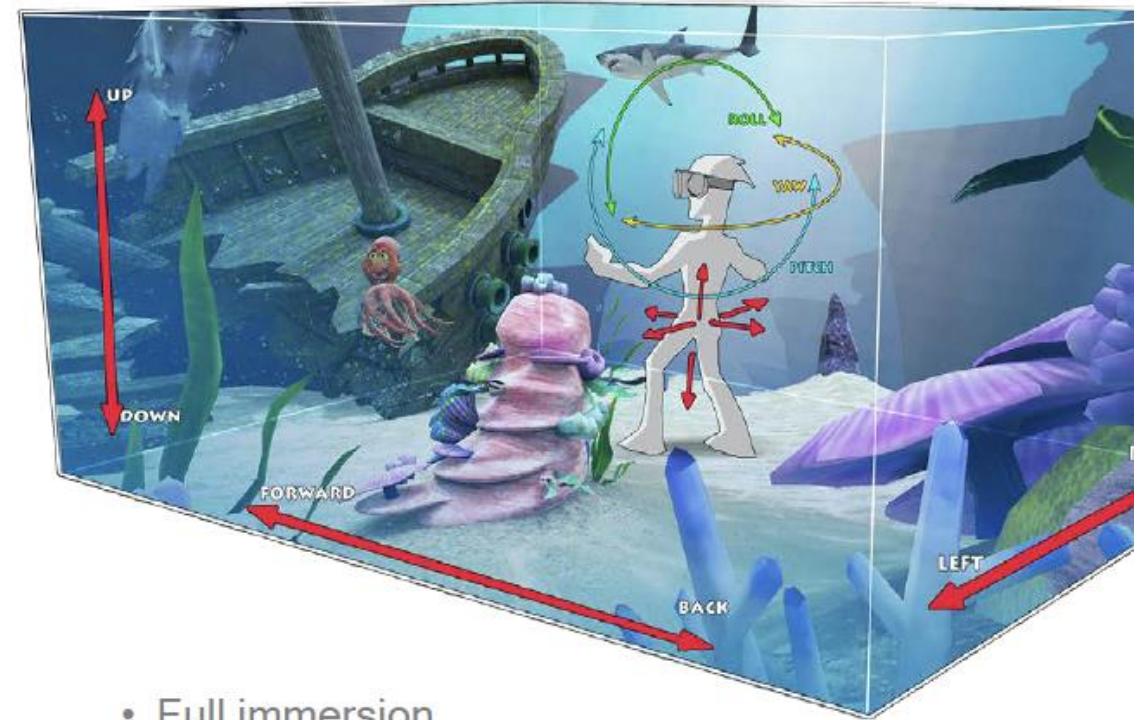
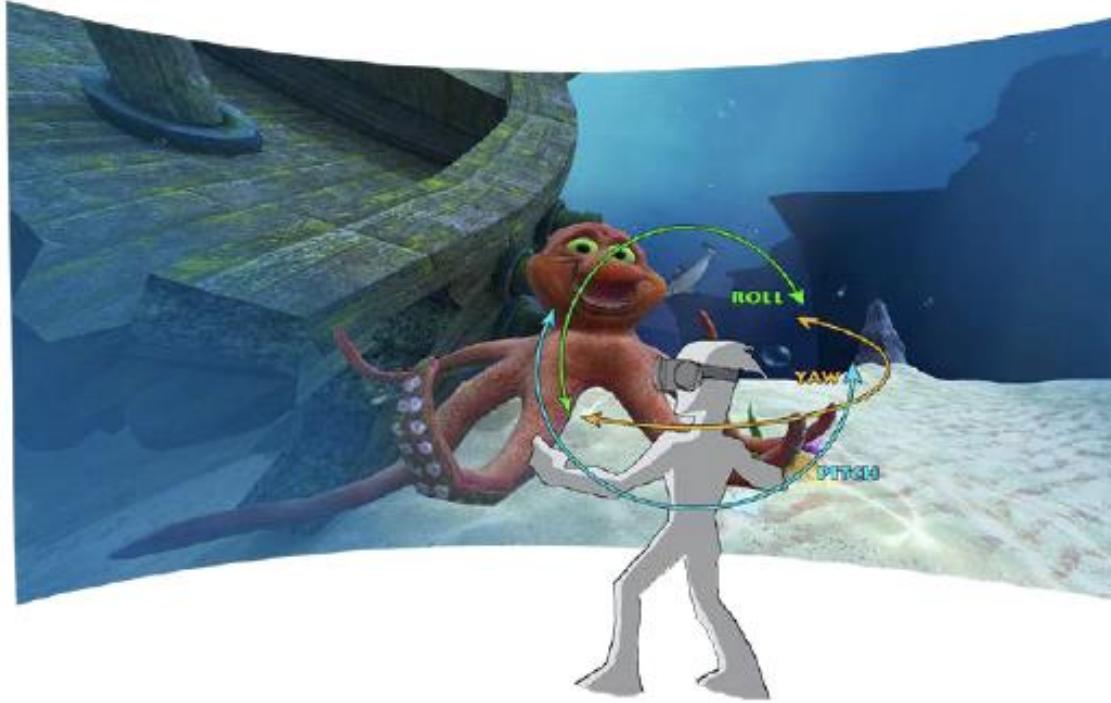
- Basic one
 - [Google CardBoard](#) and [VRAsE](#) use your smartphone
- State-of-the-art
 - [Carl Zeiss VR One Plus](#)
 - [HTC Vive](#)
 - [Microsoft HoloLens](#)
 - [OculusVR](#) bought by facebook
 - [Samsung GearVR](#)
 - [Sensics](#) invented the [Smart Goggles](#)
 - [Sony PlayStation VR](#)
 - [Sulon](#) Technologies



This Photo by Unknown Author is licensed under CC BY-SA-NC

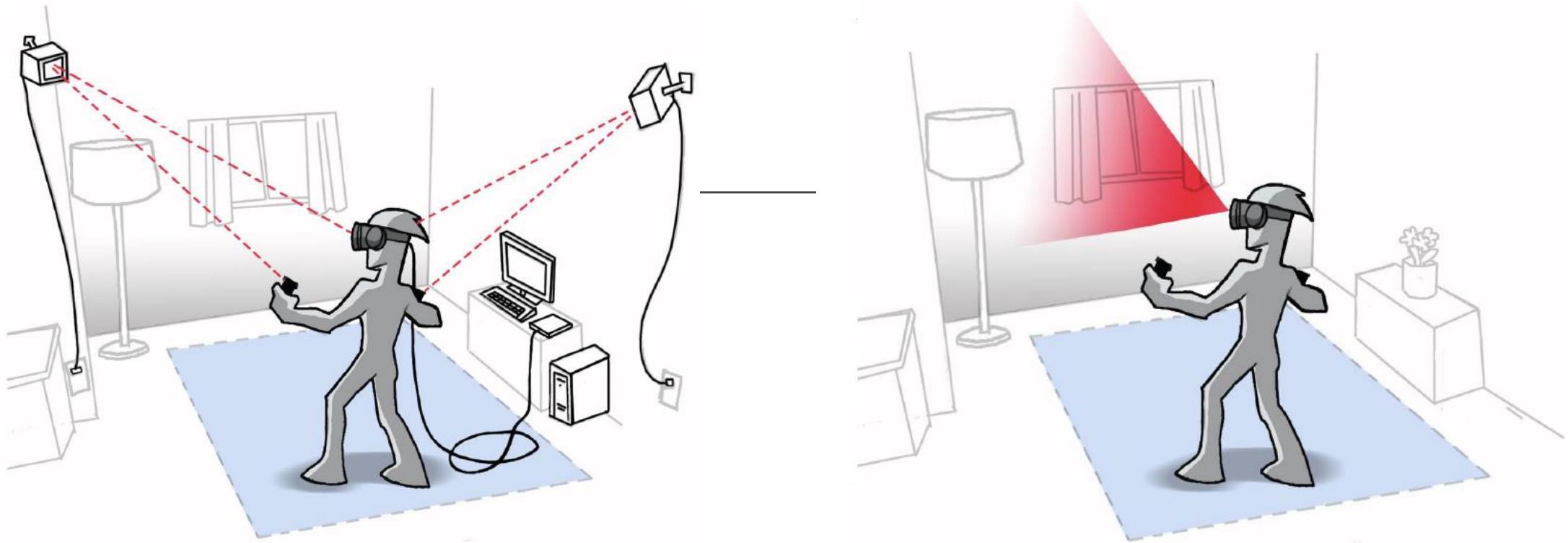


VR Metrics



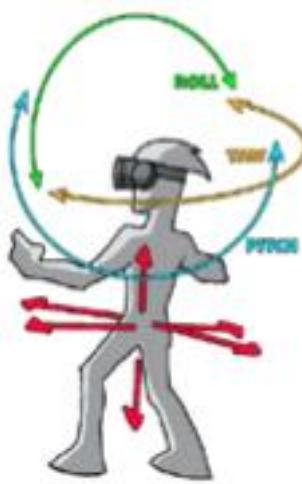
- Full immersion

Bring the user into the story



World-fixed vs. User Fixed (Revisited)

An illustration



Motion

Motion Detection

Sensor sampling
Sensor Fusion



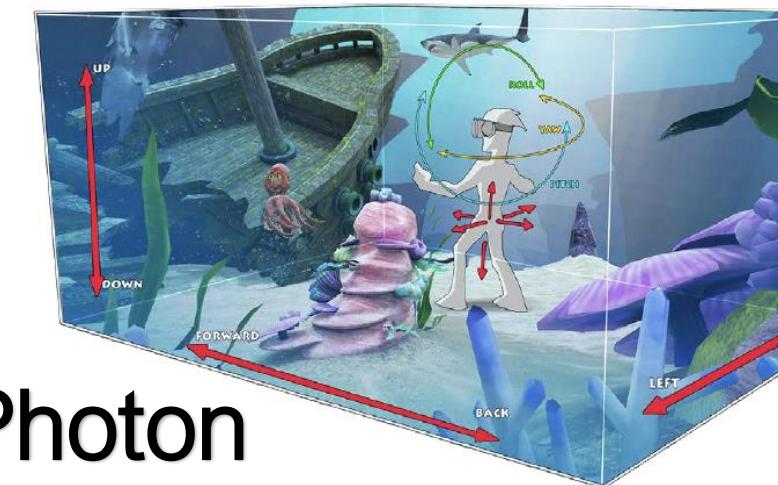
Visual Processing

View Generation
Render



Display

Image creation
Quality enhancement



Photon

Applications

Children Playing



Kids chasing virtual characters in more interactive and immersive games

Young Adults Exploring



A young man exploring Rome and seeing the Colosseum as originally built

Families Communicating



Families virtually brought together with life-like communication

Professionals Working



Architects collaborating on a shared design to improve efficiency

Fitness Enthusiasts Thriving



Group running with a virtual trainer to motivate them

Industry and Enterprise

Industrial and manufacturing

- Guided training and remote support
- Improved safety
- Real-time factory diagnostics

Healthcare

- More efficient patient care
- Diagnosis and treatment assistance
- Surgical training and visualization

Education

- Immersive, self-guided, interactive visual learning
- Any subject, from history and physics to vocational

Military

- Instructional training
- In-the-field assistance



Engineering

- 3D visualization and CAD
- Colleague collaboration and communication

Retail

- Try before you buy: clothes, furniture, car, real estate shopping, etc.
- Navigation to products and personalized coupons

Marketing and advertising

- Personalized ads based on context
- Consumer data - what they like, what they look at, etc.

Emergency response

- Police, fire, security response
- Potential improvements in safety, response time, and saving lives

AR Business Today



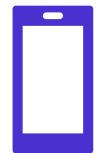
Marketing

Web-based, mobile



Gaming

Mobile, Physical input



Mobile AR

Geo-located information and service
Driving demand for high end phones



Upcoming areas

Manufacturing,
Medical, Military

Thank you

what is VR?

- emulating the real world
- making an electronic world seem real

interactive

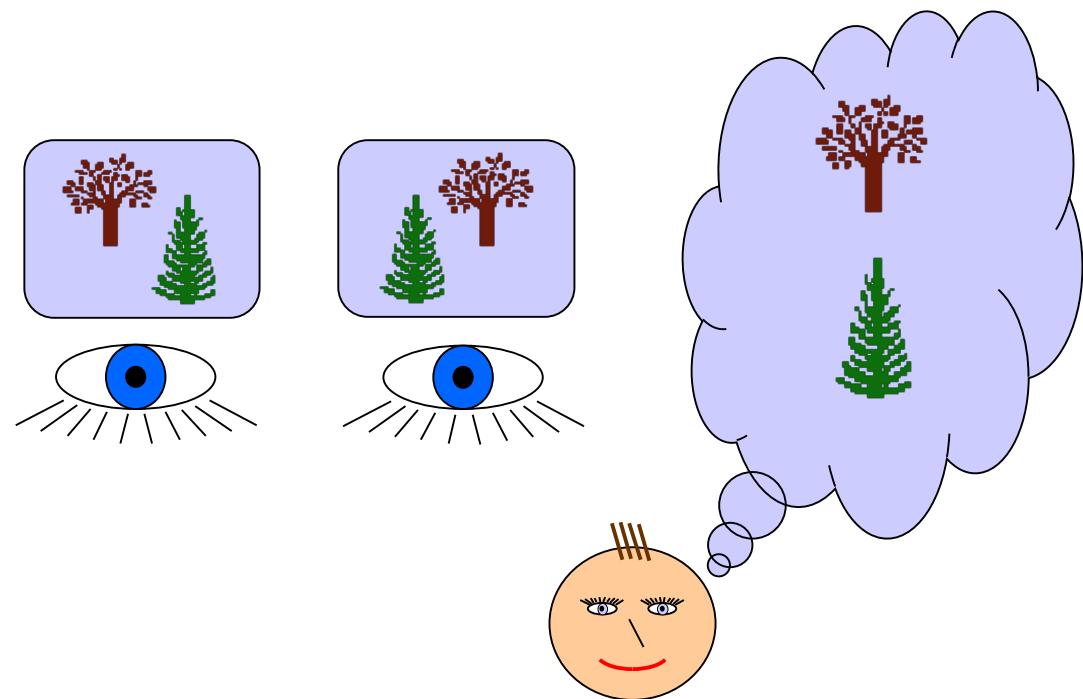
- not static 3D images
- not movies
- moving within the world
- manipulating objects in the world

types of VR

- desktop VR
 - ordinary screen, mouse or keyboard control
 - polemous mouse, steering wheel, joystick
- immersive VR
 - helmet/goggles, data glove
 - body suit, trampoline!
- physical world/VR mixes

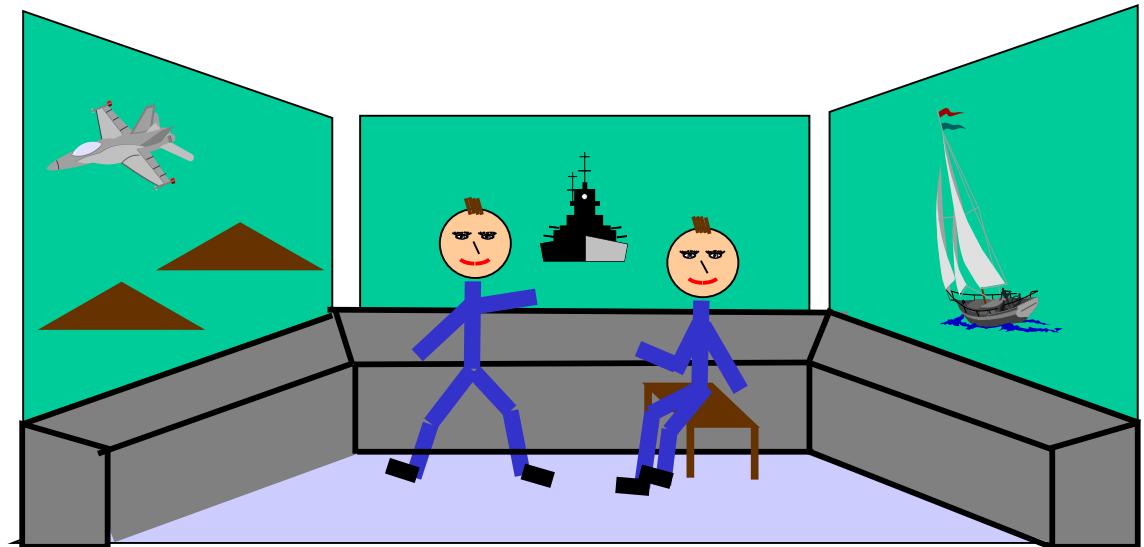
VR headsets

- small TV screen for each eye
- slightly different angles
- 3D effect



inside VR

- scenes projected on walls
- realistic environment
- hydraulic rams!
- real controls
- other people



engagement

feeling part of the virtual world

engage the senses

- sight:
 - visual realism, 3D effects: shadows, etc.,
- sound:
 - surround sound, sub-seat woofers etc.
- touch:
 - haptic and force feedback,
- and more ... wind, ... burning rubber!

engage the body

- realistic devices
- movement in the environment
- interaction and control of objects,
- rapid feedback ...
more important than photo-realism

Current problems

- ⌚ Cybersickness / simulator sickness
 - ⌚ Low-fidelity
 - ⌚ Expensive
 - ⌚ Lack of integration between application packages
-
- 😊 High-fidelity system
 - 😊 Cost-saving
 - 😊 Collaborative
 - 😊 High-level contact between participants in distributed VR