

# کارگاه آشنایی و ساخت برنامه های واقعیت افزوده

Augmented reality zero to hero

Reza Habibi



# Augmented Reality vs Virtual Reality



# Virtual Reality

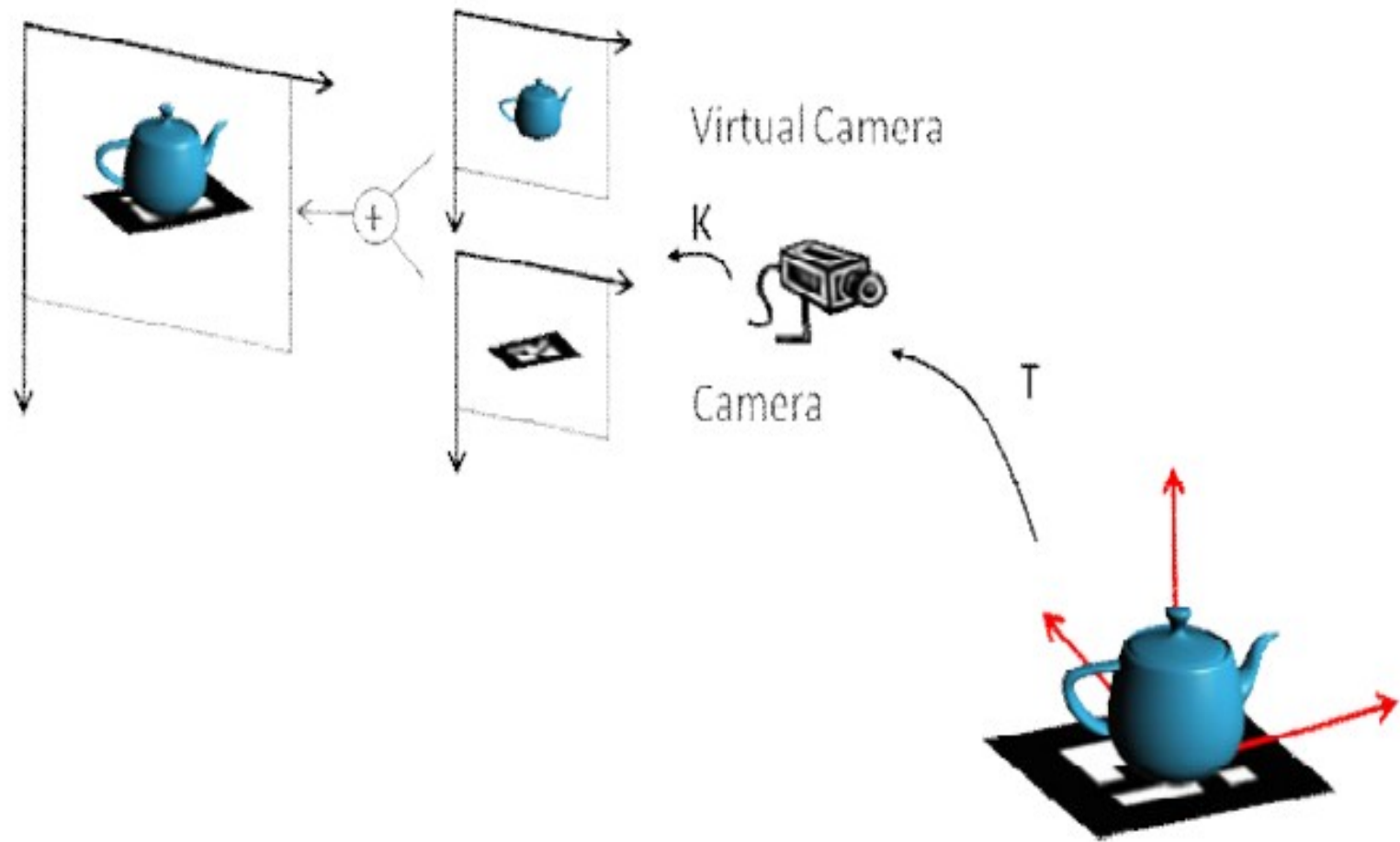




# How it works!

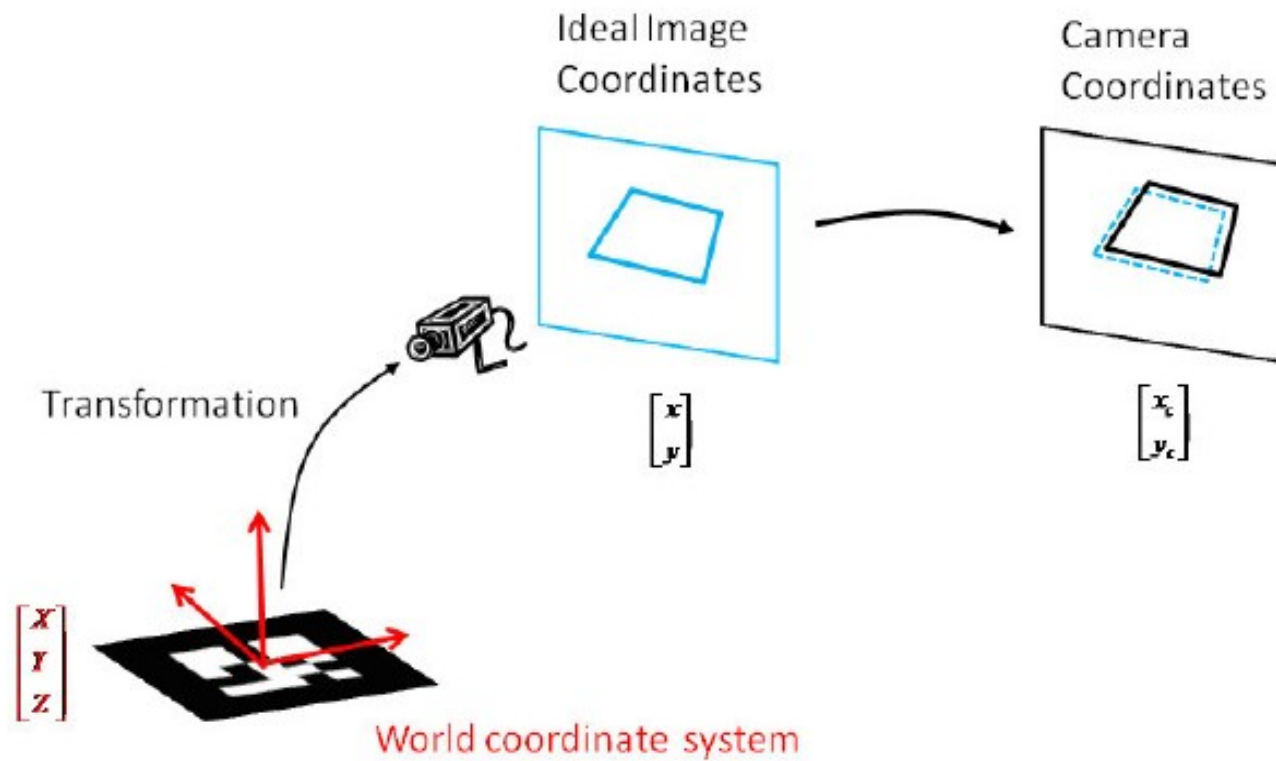


# How it works!

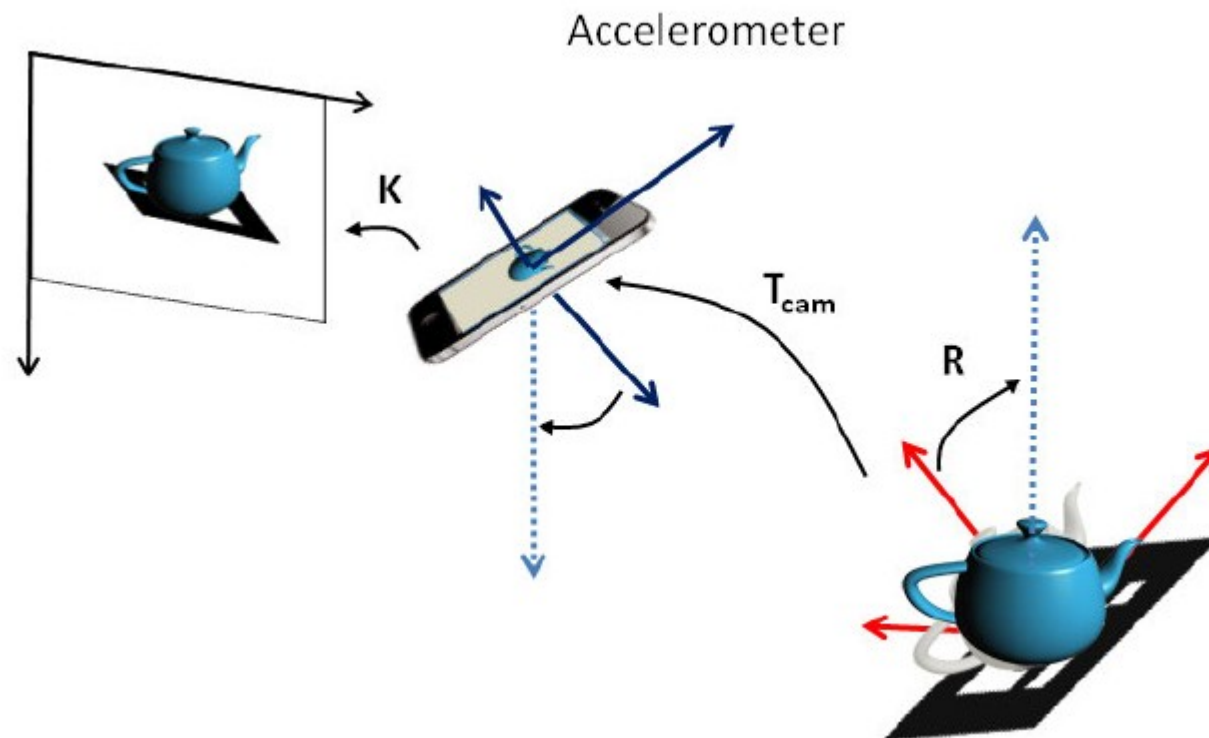




# How it works!

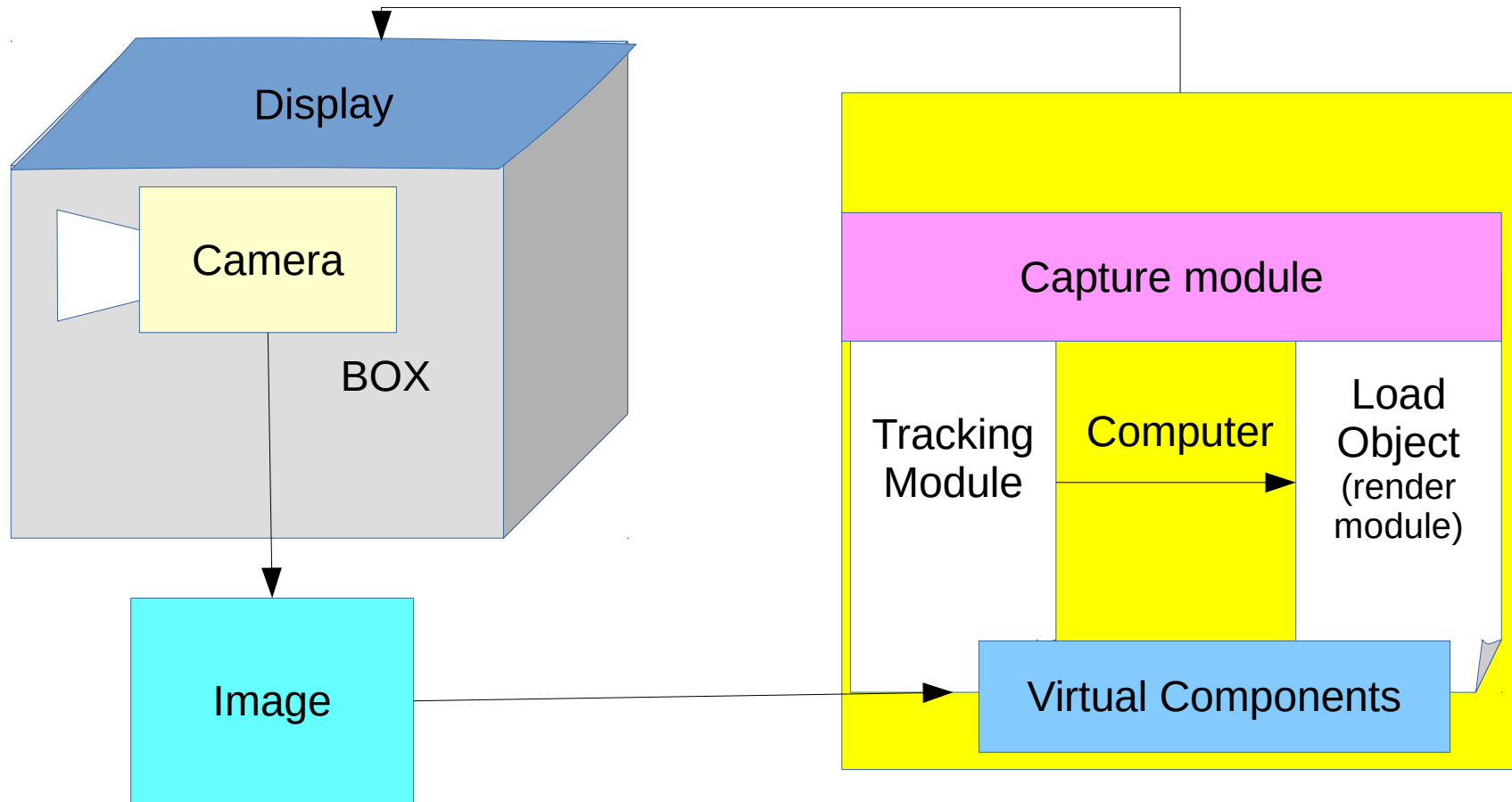


# How it works!





# How it works!



# Equipment

**1- Marker**

**2- 3D file**

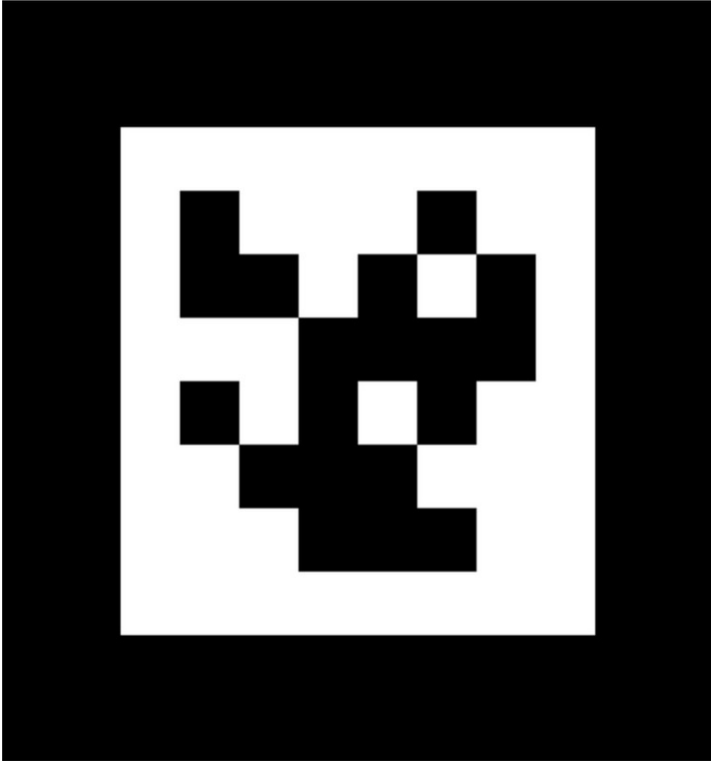
**3- Computer/Mobile**

**4- Camera**

**5- Good Programming Language and library**



# Marker – Hiro



**1- Single Marker Based**

**2- Multi Marker Based**

**3- Markerless**

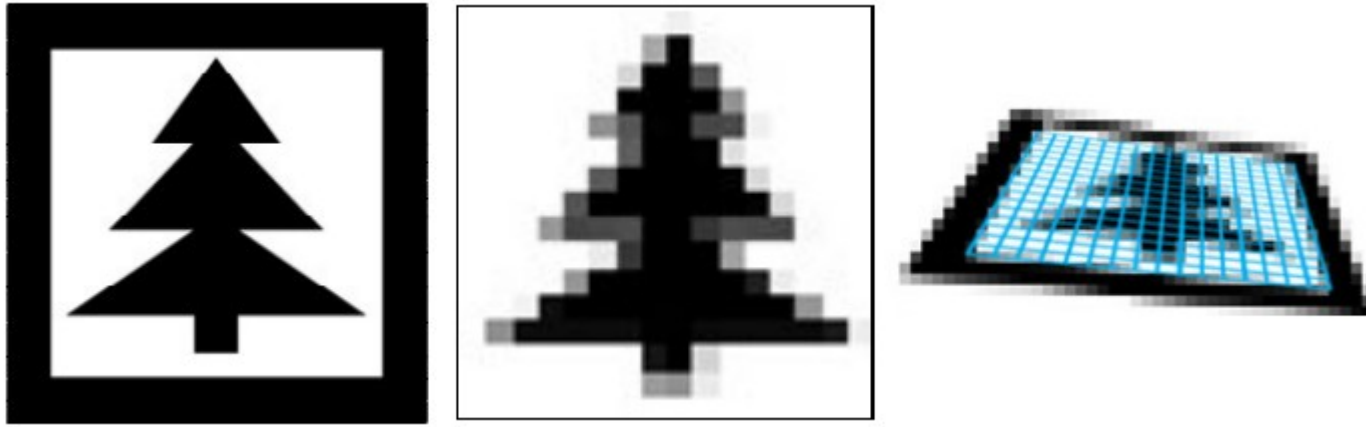
**4- Hybrid Marker**

# Marker – Hiro



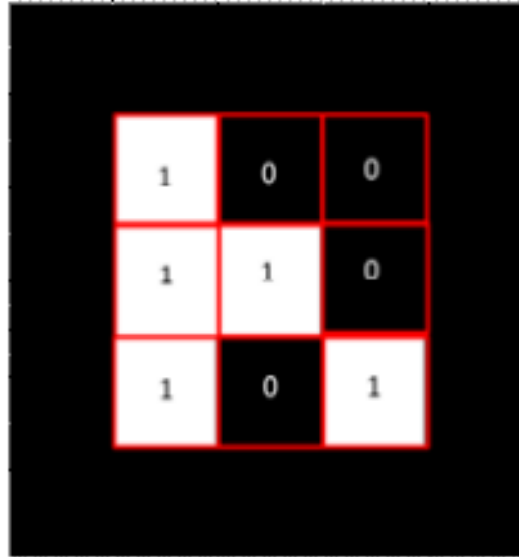
**Example of Template Markers.**

# Marker – Hiro



Deteced Marker

# Marker - Hiro



**Decode Bairy Data markers**

**100 110 101 = 309**

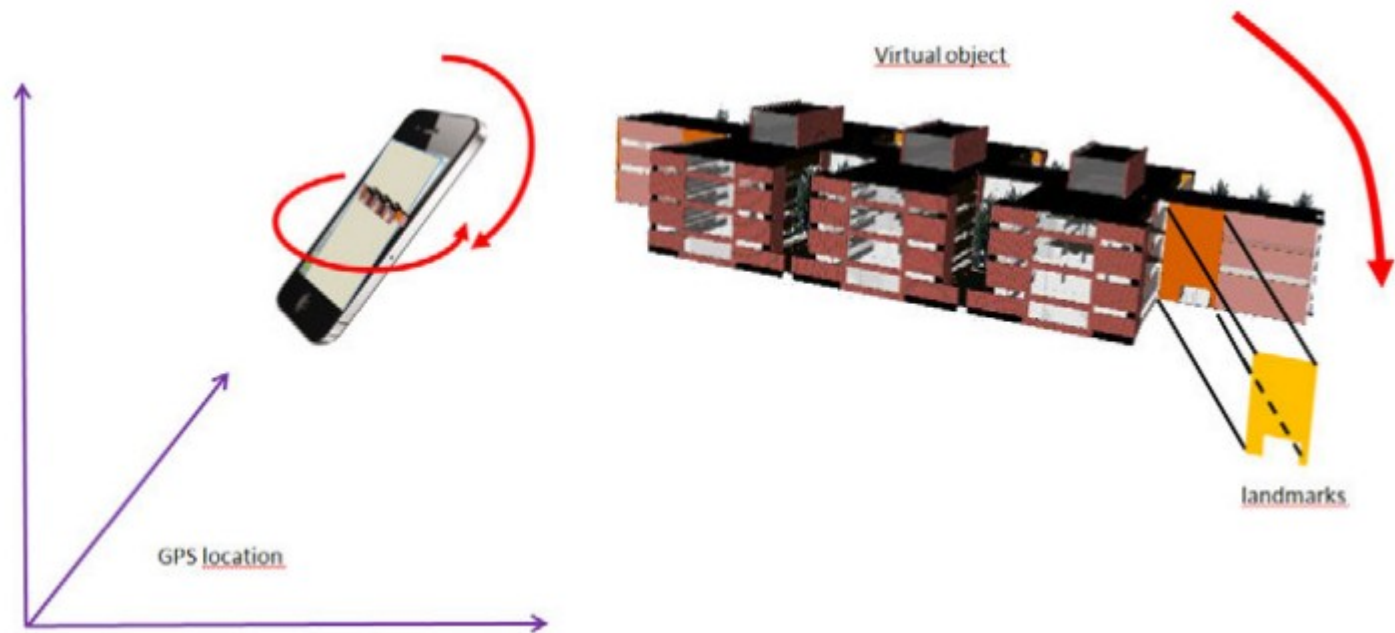
# Marker – Hiro



Image Marker



# Marker - Hiro



**Hybrid Tracking system Using GPS**

# Marker – Hiro

**-Three main approaches to increase the number of points used for pose detection are:**

to use more than four points per marker

to use more than one marker

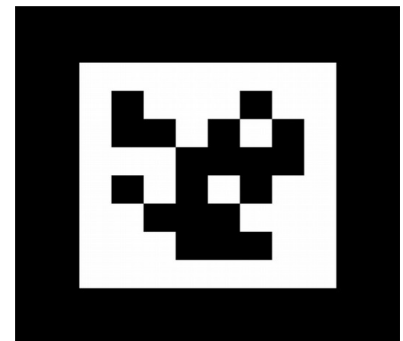
to use natural features in addition to the marker.

## **-Common Problem in Tracking**

This is an unwanted limitation in many applications and therefore, the second

and third options are commonly preferred. AR systems habitually use them to

increase robustness and usability.



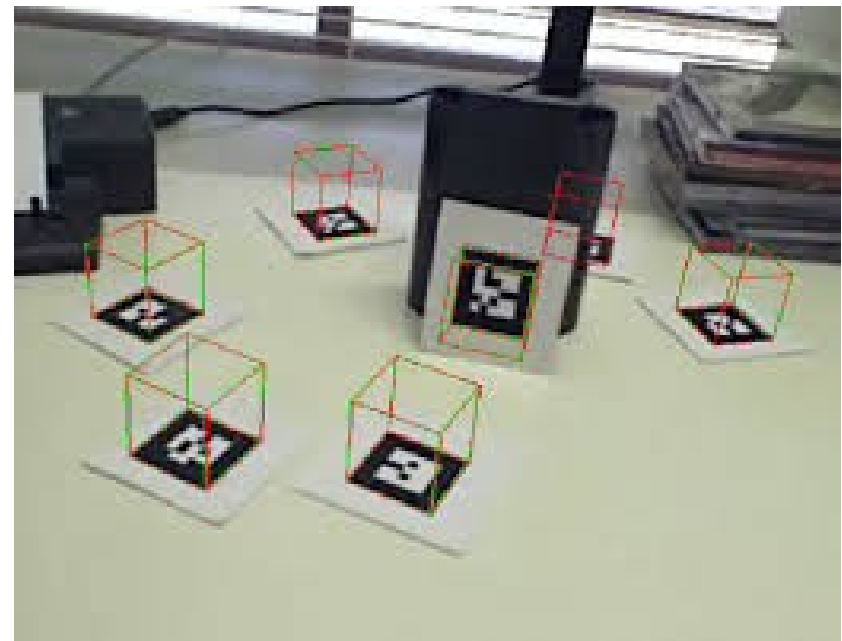
# Marker – Hiro

## -Multi Marker Systems

combine the information from all markers,

## - Predefined multi-marker

A multi-marker system can use a non-planar predefined marker field as well.



# Marker – Hiro

- **Automatic reconstruction of multi-marker setups**

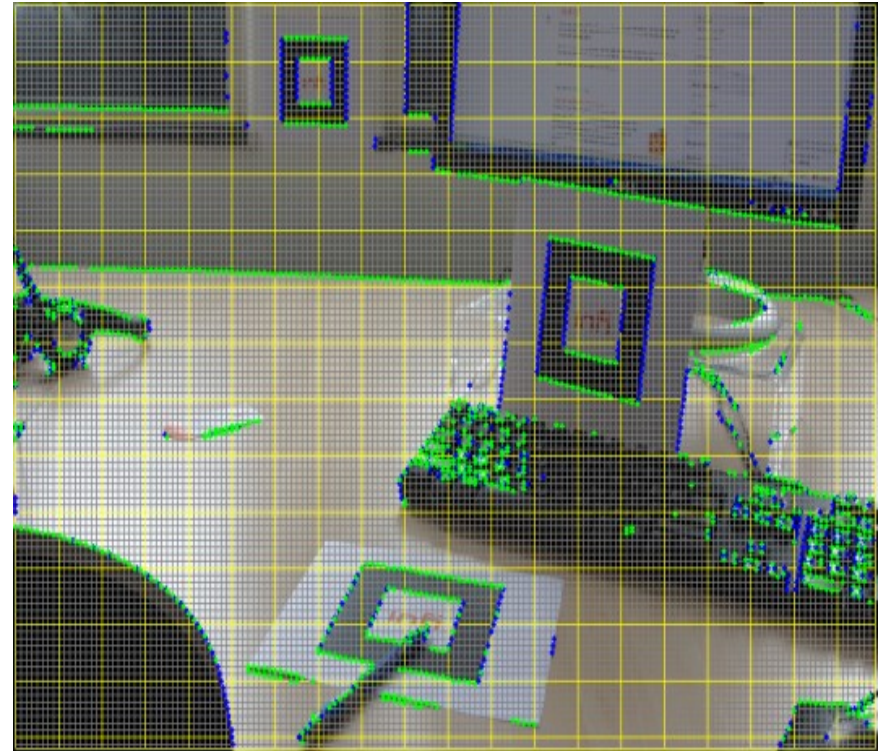
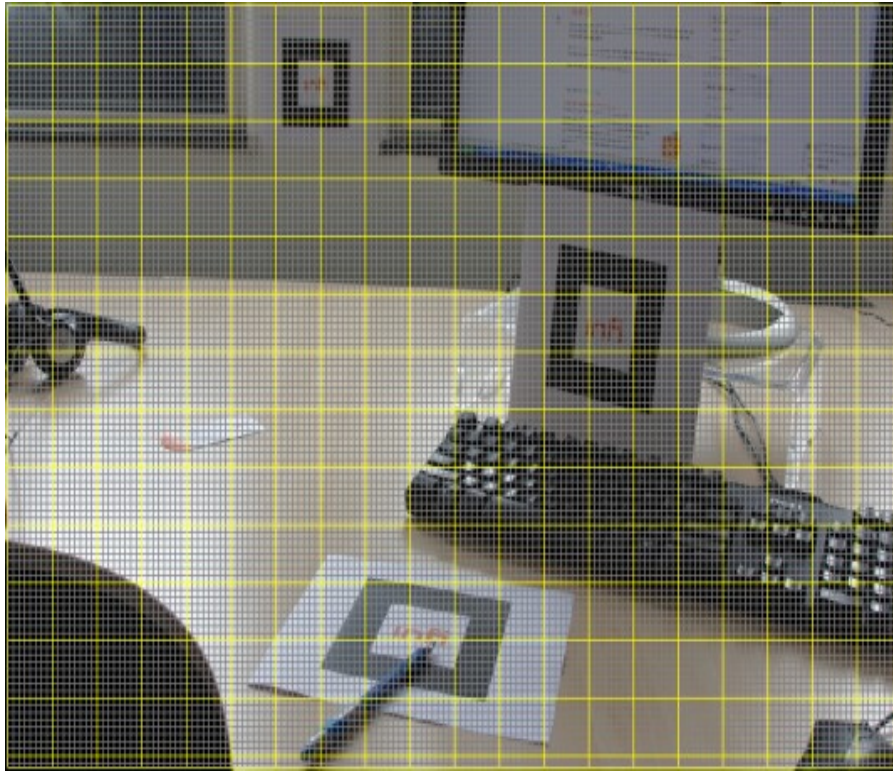
In automatic reconstruction of multi-marker setups, a system needs to determine.

the 3D coordinates of markers based on observations (2D images).

- Kalman Filtering

-

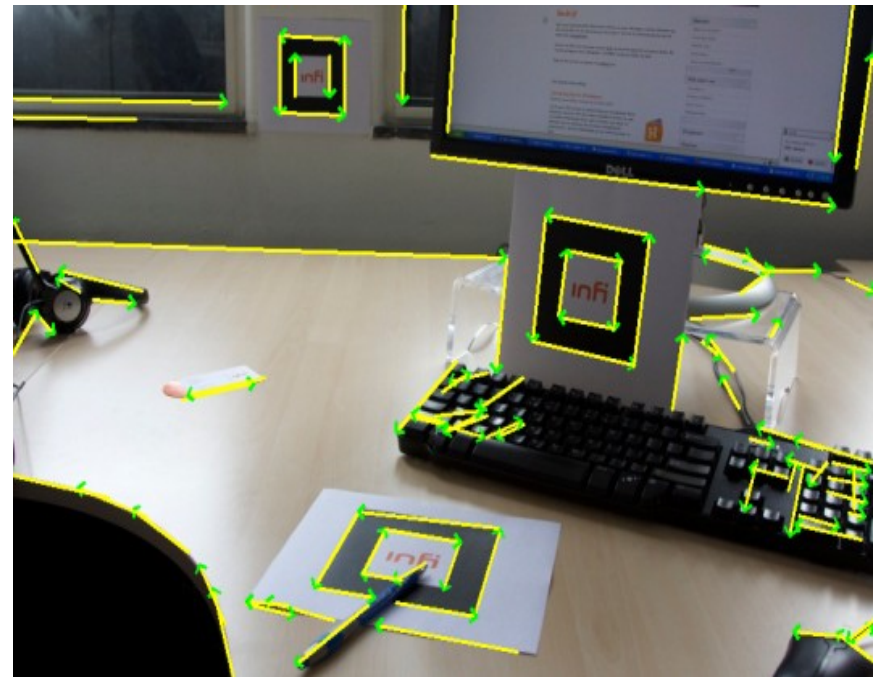
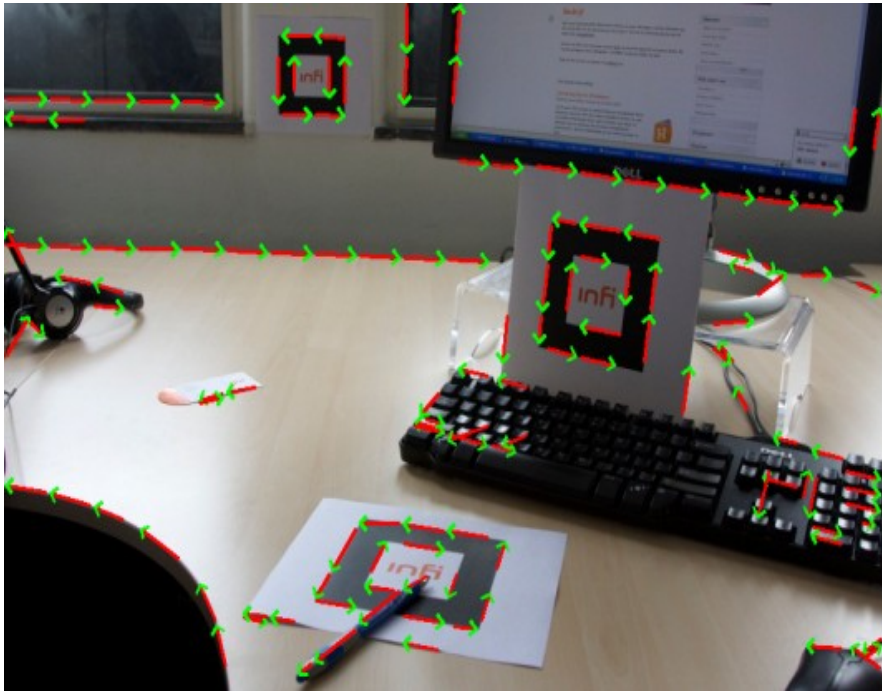
# Marker Detection for Augmented Reality



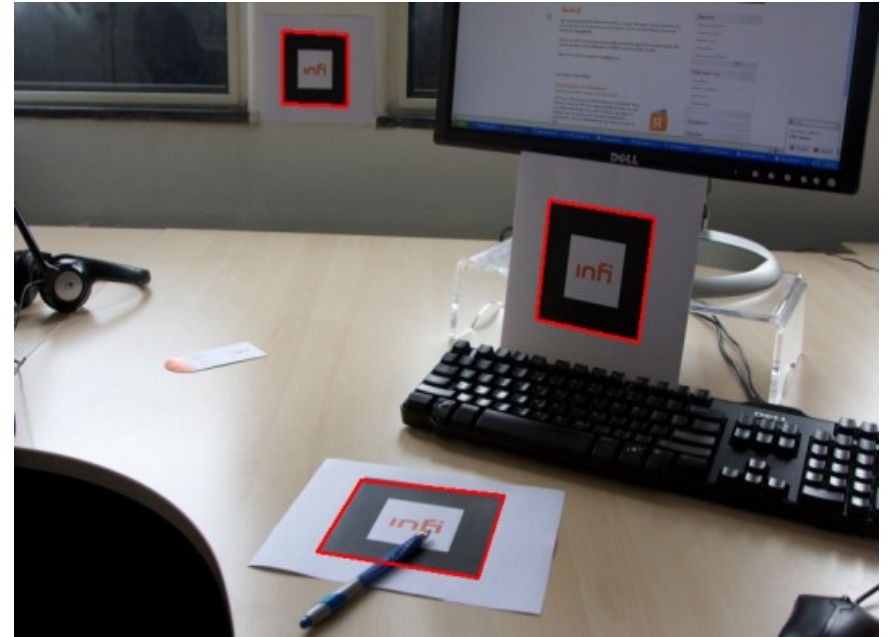
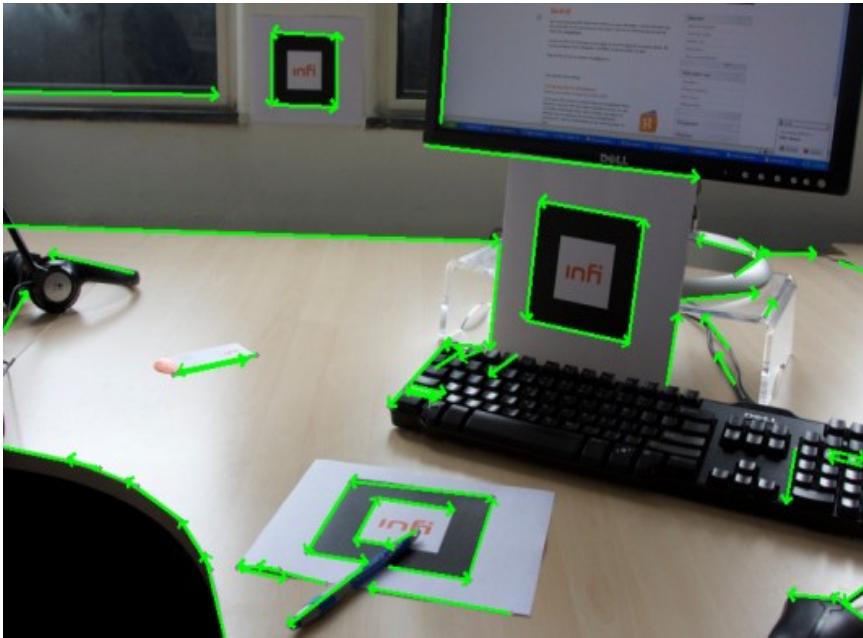


# Marker Detection for Augmented Reality

$$\mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A}$$



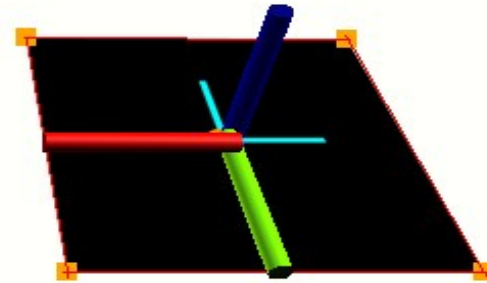
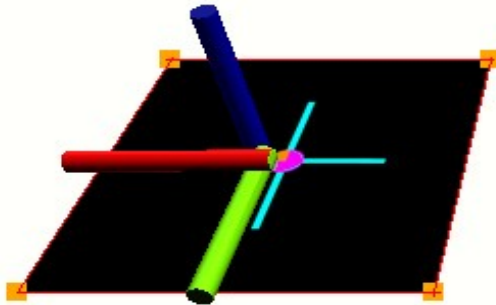
# Marker Detection for Augmented Reality



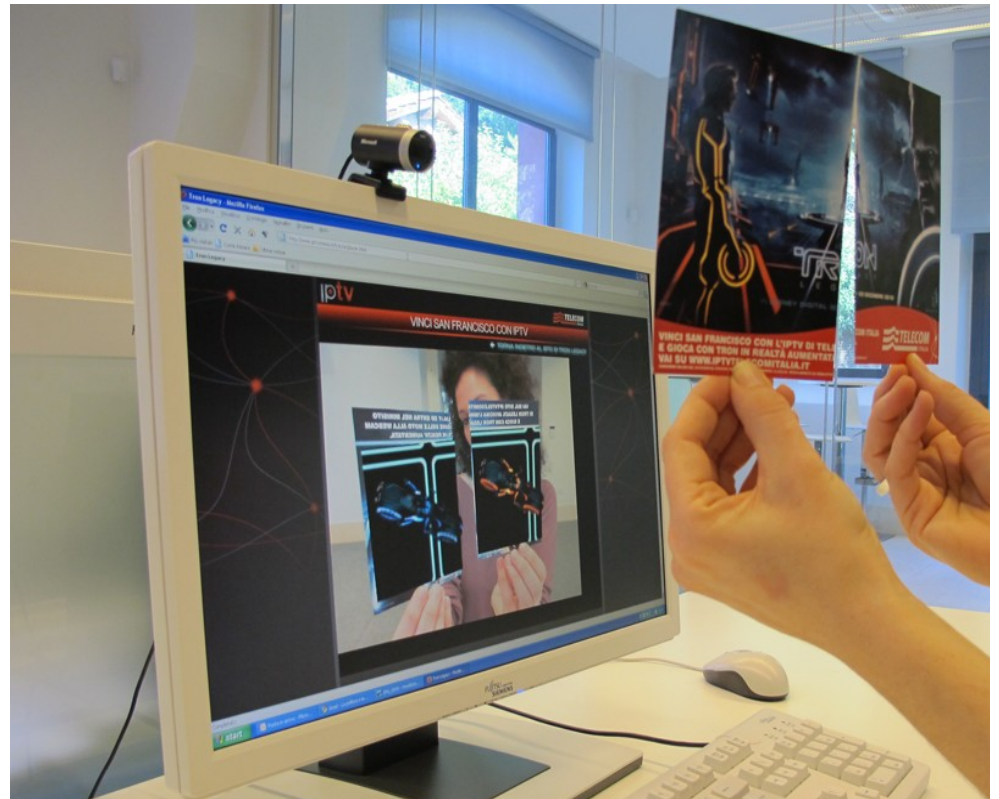


# Marker Detection for Augmented Reality

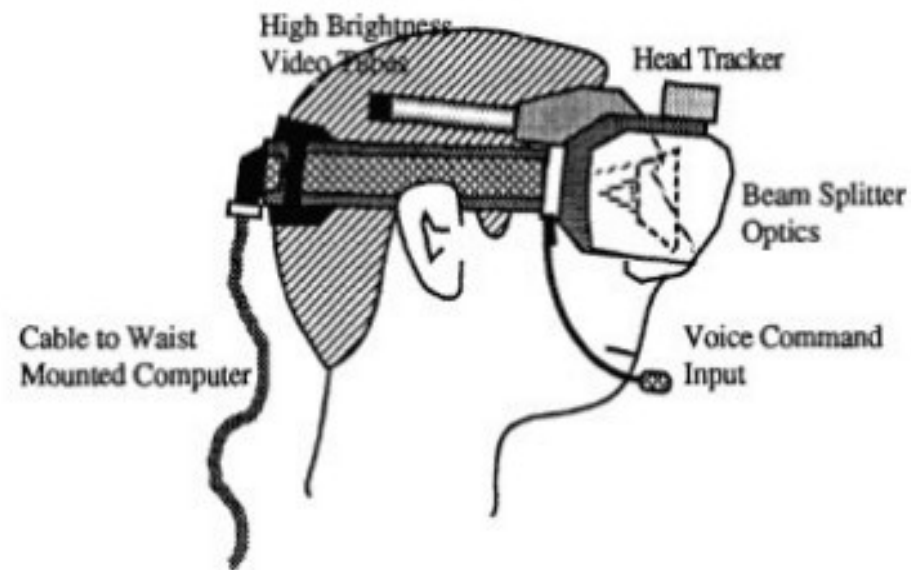
8 X :755 Y:372



# Camera



# HMD



# AR development library

## **ARToolKit:**

C-library to create augmented reality applications; was ported to many different languages and platforms like Android, Flash or Silverlight; very widely used in augmented reality related projects

## **JavaCV,ArUco,:**

OpenCV

## **BeyondAR,GeoAR,DroidAR,:**

augmented reality framework based on geo localisation for Android

# AR development toolkit

## **Vuforia Augmented Reality SDK:**

is a Software Development Kit for creating augmented reality applications for mobile devices

## **metaio SDK :**

offers free natural features tracking and 3D tracking that is available for Android, iOS and Microsoft Windows.

## **BeyondAR,GeoAR,DroidAR,:**

augmented reality framework based on geo localisation for Android

# Software 3D



3Ds Max



Blender

# 3D file

Type of 3D file:

- max
- obj
- dae/collada
- 3ds





# Game Engine



Unity



library

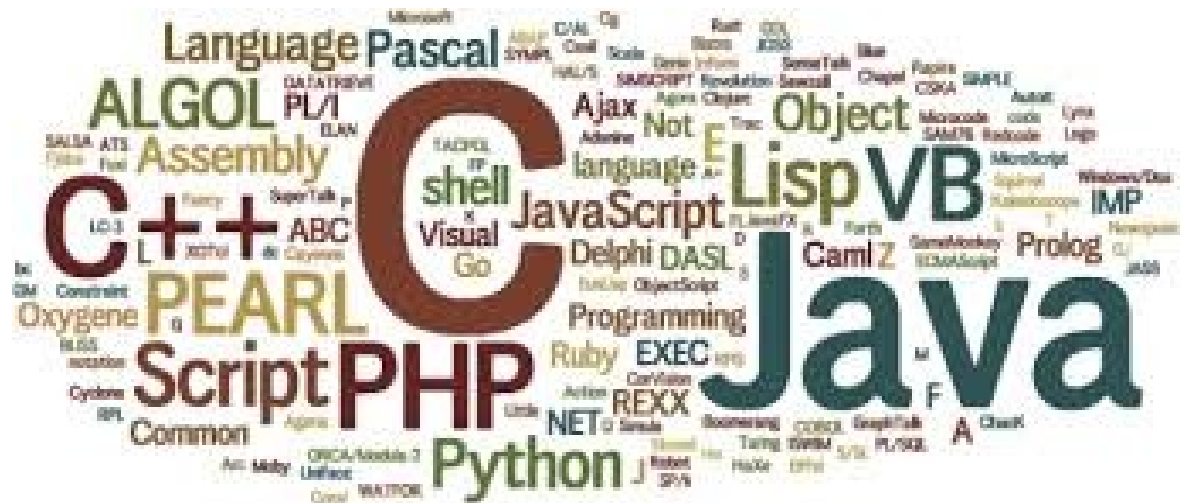
- ARToolKit
- OpenCv

ARTOOLKIT



# language

- C++
- C#
- Python
- Javascript
- AS



**STAND BACK**



**I'M GOING TO TRY  
SCIENCE**

Let's Create app!!

# Marker

- 1- 600 600px image Black background
- 2- 450 450 px white text
- 3- Run marker generator <http://goo.gl/eZL5>
- 4- save .pat or .patt file

# Javascript Basic

```
1 | <script src="scripts/main.js"></script>
```

```
1 | var myHeading = document.querySelector('h1');  
2 | myHeading.innerHTML = 'Hello world!';
```

JS

Hello world!



# Variables

Variables are containers that you can **store values** in.

```
1 | var myVariable;
```

JS

After declaring a variable, you can give it a value:

```
1 | myVariable = 'Bob';
```

You can do both these operations on the same line if you wish:

```
1 | var myVariable = 'Bob';
```



# JS

Note that variables have different [data types](#):

Variable	Explanation	Example
<b><u>String</u></b>	A string of text. To signify that the variable is a string, you should enclose it in quote marks.	<pre>var myVariable = 'Bob';</pre>
<b><u>Number</u></b>	A number. Numbers don't have quotes around them.	<pre>var myVariable = 10;</pre>
<b><u>Boolean</u></b>	A True/False value. The words true and false are special keywords in JS, and don't need quotes.	<pre>var myVariable = true;</pre>
<b><u>Array</u></b>	A structure that allows you to store multiple values in one single reference.	<pre>var myVariable = [1, 'Bob', 'Steve', 10]; Refer to each member of the array like this: myVariable[0], myVariable[1], etc.</pre>
<b><u>Object</u></b>	Basically, anything. Everything in JavaScript is an object, and can be stored in a variable. Keep this in mind as you learn.	<pre>var myVariable = document.querySelector('h1'); All of the above examples too.</pre>

# Conditionals

Conditionals are code structures that allow you to test whether an expression returns true or not, and then run different code depending on the result. The most common form of conditional is called **if ... else**. So for example:

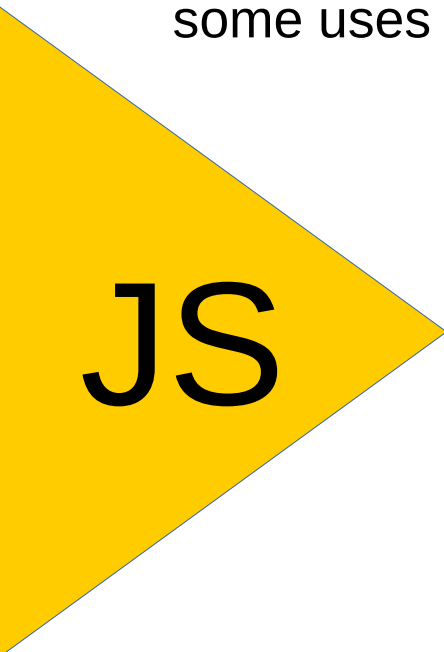
A large yellow triangle pointing to the right, containing the text 'JS' in black.

JS

```
1 var iceCream = 'chocolate';
2 if (iceCream === 'chocolate') {
3   alert('Yay, I love chocolate ice cream!');
4 } else {
5   alert('Awww, but chocolate is my favorite...');
6 }
```

# Functions

Functions are a way of packaging functionality that you want to reuse, so that whenever you want the functionality you can call the function with the function name rather than constantly rewriting the entire code. You have already seen some uses of functions above, for example:

A large yellow triangle pointing to the right, containing the text 'JS' in black.

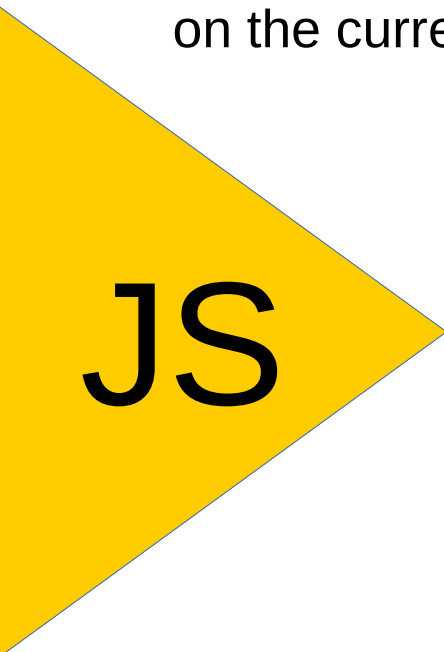
JS

```
1 function multiply(num1,num2) {  
2   var result = num1 * num2;  
3   return result;  
4 }
```

```
1 multiply(4,7);  
2 multiply(20,20);  
3 multiply(0.5,3);
```

# Events

to create real interactivity on a website, you need events — these are code structures that listen out for things happening to the browser, and then allow you to run code in response to those things. The most obvious example is the click event, which is fired by the browser when the mouse clicks on something. To demonstrate this, try entering the following into your console, then clicking on the current webpage:

A large yellow triangle pointing to the right, containing the text 'JS' in black.

# JS

```
1 | document.querySelector('html').onclick = function() {  
2 |     alert('Ouch! Stop poking me!');  
3 | }
```

# Awe.js

Awe.js is a JavaScript library that uses Three.js, your device's camera and some pretty smart techniques to create augmented reality in the browser.

**geo\_ar** – Allows you to place objects at set compass points.

**grift\_ar** – Compatible with an Oculus Rift.

**leap\_ar** – Integrates with the Leap Motion controller.

**marker\_ar** – Allows you to create an experience that is positioned on Augmented Reality markers. This is the one we'll be working with in this demo.



created by  
buildAR



# Awe.js

Everything starts within the load **event** on our window.

```
window.addEventListener('load', function() {  
  var menu_open = false;  
  
  // Our code continues here  
});
```



# Awe.js

Then, we start to use the awe.js library. Everything we do is defined within the **window.awe.init()** function. We start with some global settings for our AR scene.

```
window.awe.init({  
  device_type: awe.AUTO_DETECT_DEVICE_TYPE,  
  settings: {  
    container_id: 'container',  
    fps: 30,  
    default_camera_position: { x:0, y:0, z:0 },  
    default_lights: [{  
      id: 'point_light',  
      type: 'point',  
      color: 0xFFFFFF  
    }]  
  },  
}
```



# Awe.js

Once our settings are in place, we then define what to do when awe.js has initialised. Everything is wrapped within a `awe.util.require()` function which defines what browser capabilities it requires before loading additional JavaScript files we'll need

```
ready: function() {  
  awe.util.require([  
    {  
      capabilities: ['gum','webgl'],
```





# Awe.js

The files that are defined pull in specific functionality for awe.js –

lib/awe-standard-dependencies.js, lib/awe-standard.js and lib/awe-standard-window\_resized.js

files: [

['lib/awe-standard-dependencies.js', 'lib/awe-standard.js'],

'lib/awe-standard-window\_resized.js',

'lib/awe-standard-object\_clicked.js',

'lib/awe-jsartoolkit-dependencies.js',

'lib/awe.marker\_ar.js'

],



# Awe.js

set it to be invisible until we spot the marker.

```
awe.pois.add({id: 'marker', position: {x: 0, y: 0, z: 10000}, visible: false});
```



# Awe.js

Elements we add into our POIs are called “projections” within awe.js

```
awe.projections.add({  
  id: 'wormhole',  
  geometry: {shape: 'plane', height: 400, width: 400},  
  position: {x: 0, y: 0, z: 0},  
  rotation: {x: 90, z: 45},  
  material: {  
    type: 'phong',  
    color: 0x000000  
  }  
}, {poi_id: 'marker'});
```



# Awe.js

our marker detection event. We add this as an array passed to the function `awe.events.add()`.

```
awe.events.add([  
  // Our events here  
]);
```

We've only got one awe.js event, so there is just one single event here.

```
id: 'ar_tracking_marker',  
  device_types: {  
    pc: 1,  
    android: 1  
  },
```



# Awe.js

Then we have `register()` and `unregister()` functions to add and remove the event listener that is watching for the marker.

```
register: function(handler) {  
    window.addEventListener('ar_tracking_marker', handler, false);  
},  
unregister: function(handler) {  
    window.removeEventListener('ar_tracking_marker', handler, false);  
},
```



# Awe.js

We then define the event handler which will run once we spot a marker. We look out for the “64” marker and only run a response when we find it.

```
handler: function(event) {  
  if (event.detail) {  
    if (event.detail['64']) {  
      // Our response!  
    }  
  }  
}
```



# Awe.js

Within our response to finding a marker, we want to move our POI that we called 'marker' onto the spot with our physical paper marker and make it visible. We transform it to align to the physical marker using `event.detail['64'].transform`.

```
awe.pois.update({  
  data: {  
    visible: true,  
    position: {x: 0, y: 0, z: 0},  
    matrix: event.detail['64'].transform  
  },  
  where: {  
    id: 'marker'  
  }  
});
```



# Awe.js

We also set our 'wormhole' projection to be visible.

```
awe.projections.update({  
  data: {  
    visible: true  
  },  
  where: {  
    id: 'wormhole'  
  }  
});
```





# Awe.js

We finish by telling awe.js to update the scene.

```
awe.scene_needs_rendering = 1;
```



# Thanks !

Rezahabibi(رضاحیبی) Software Engineer | [Reza.hbi@gmail.Com](mailto:Reza.hbi@gmail.com)