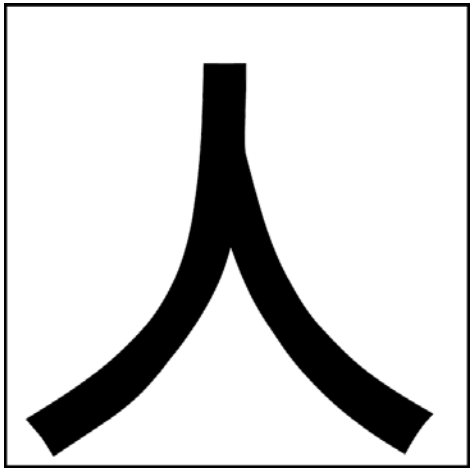
le cnam



Alexandre Topol

Propos de Ron Azuma

- To avoid limiting AR to a specific technology, AR systems should have the following characteristics:
  - Combines real and virtual elements
  - Interactive in real time
  - Registered in 3D

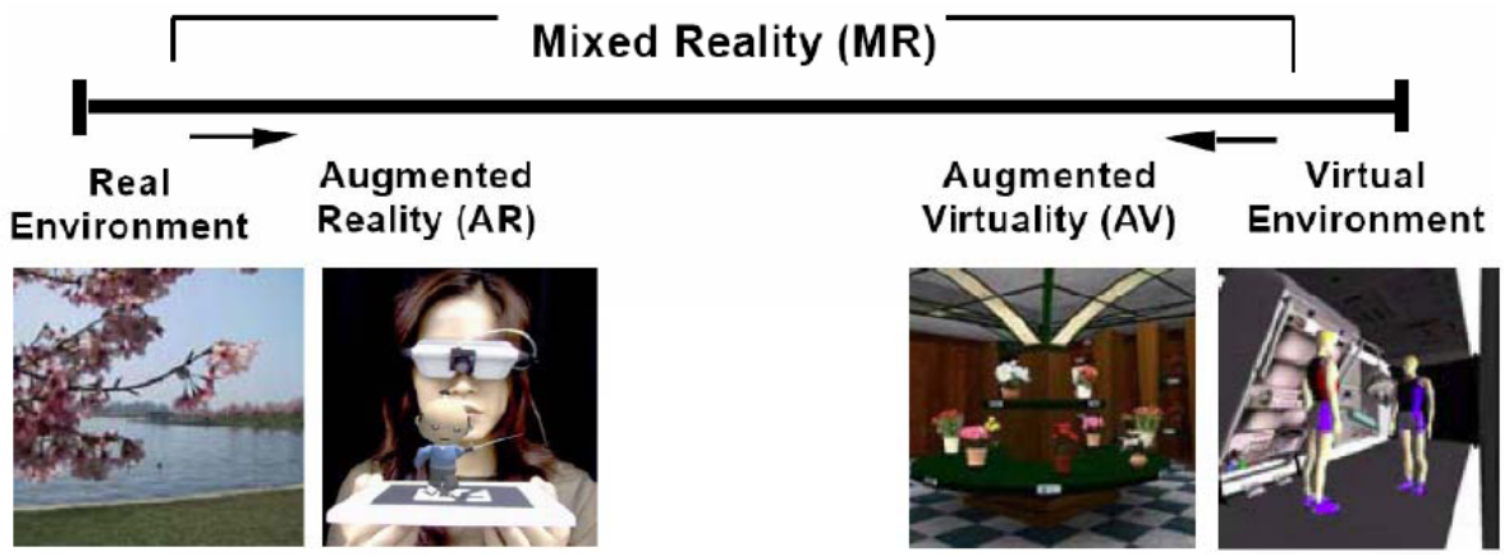
Définition Wikipedia

- Augmented reality (AR) is a term for a **live** direct or an indirect view of a **physical, real-world environment** whose elements are ***augmented*** by computer-generated sensory input, such as sound or graphics.

Le continuum réalité-virtualité

[Milgram et al., 1994]

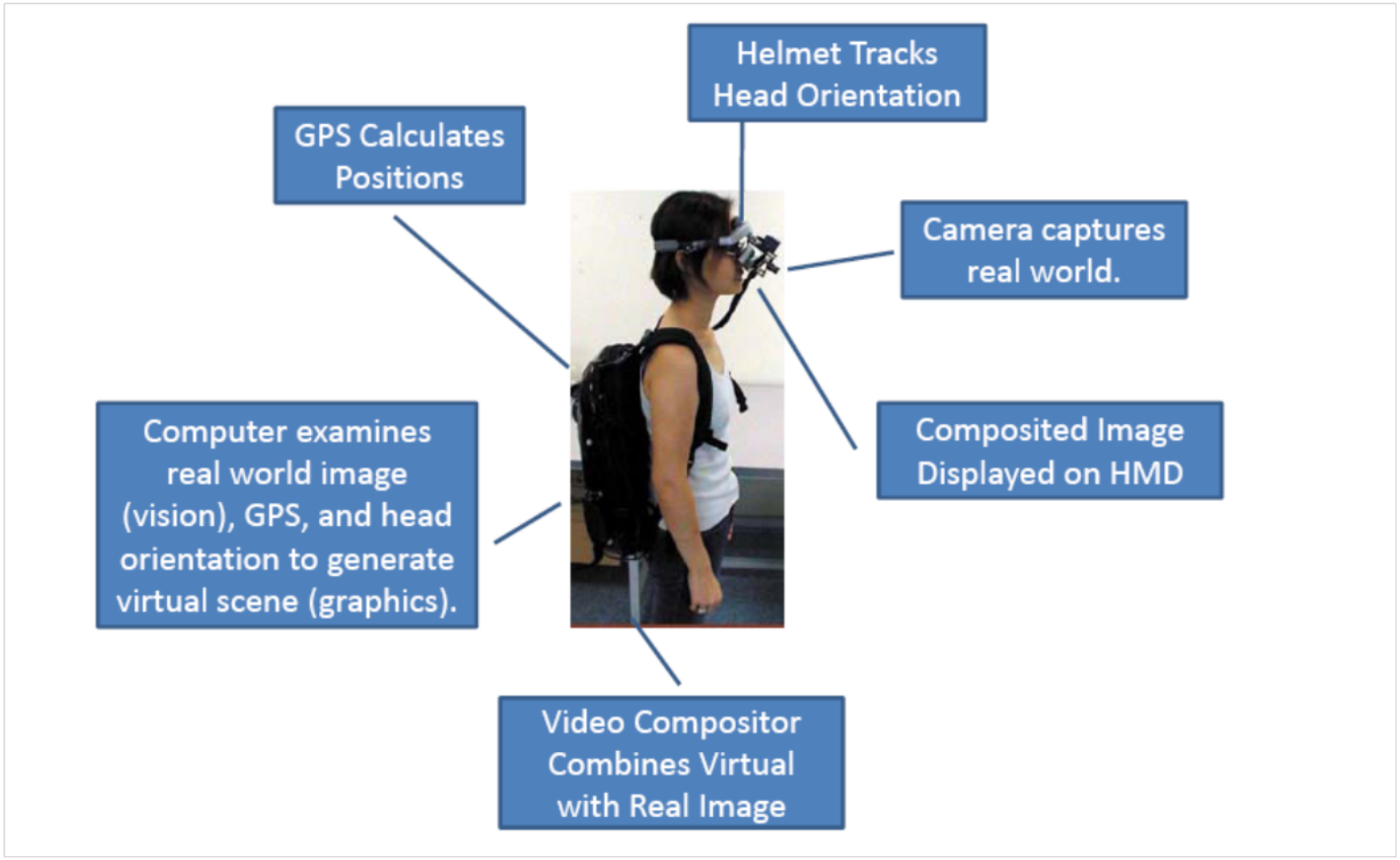
"Augmenting natural feedback to the operator with simulated cues" (Milgram et al., 1994)



Historique

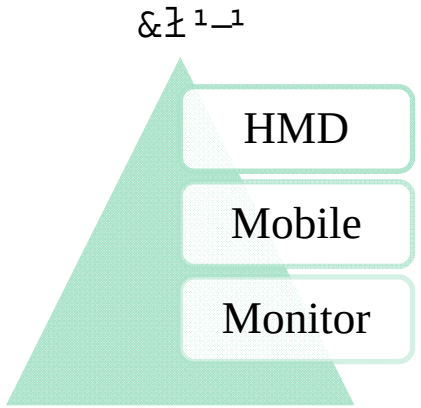
- 1966 - Ivan Sutherland invents HMD
- 1990 – Tom Caudell coins the phrase Augmented Reality
- 1992 – First functioning AR systems appear (military)
- 1999 - Hirokazu Kato creates ARToolKit at HITLab
- 2009 – ARToolKit ported to Flash by Saqoosha

Difficulté technique



Coût

- Les interfaces et dispositifs de la RA :
  - Head Mounted Displays (HMD)
  - Hand-Held Displays
  - Spatial Displays (environmentally integrated)
    - Monitors
    - Projectors



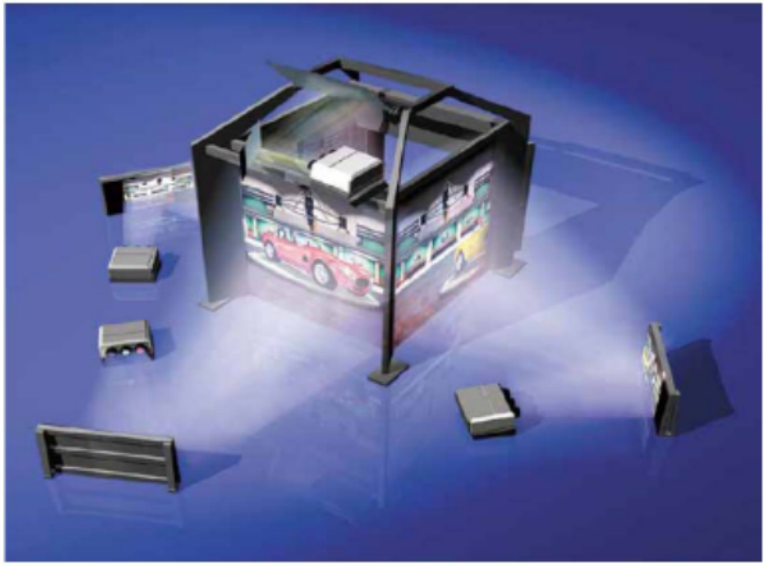
AR Displays - Hand-Held

AR Displays - Spatial

AR Displays – Stéréoscopie

le cnam

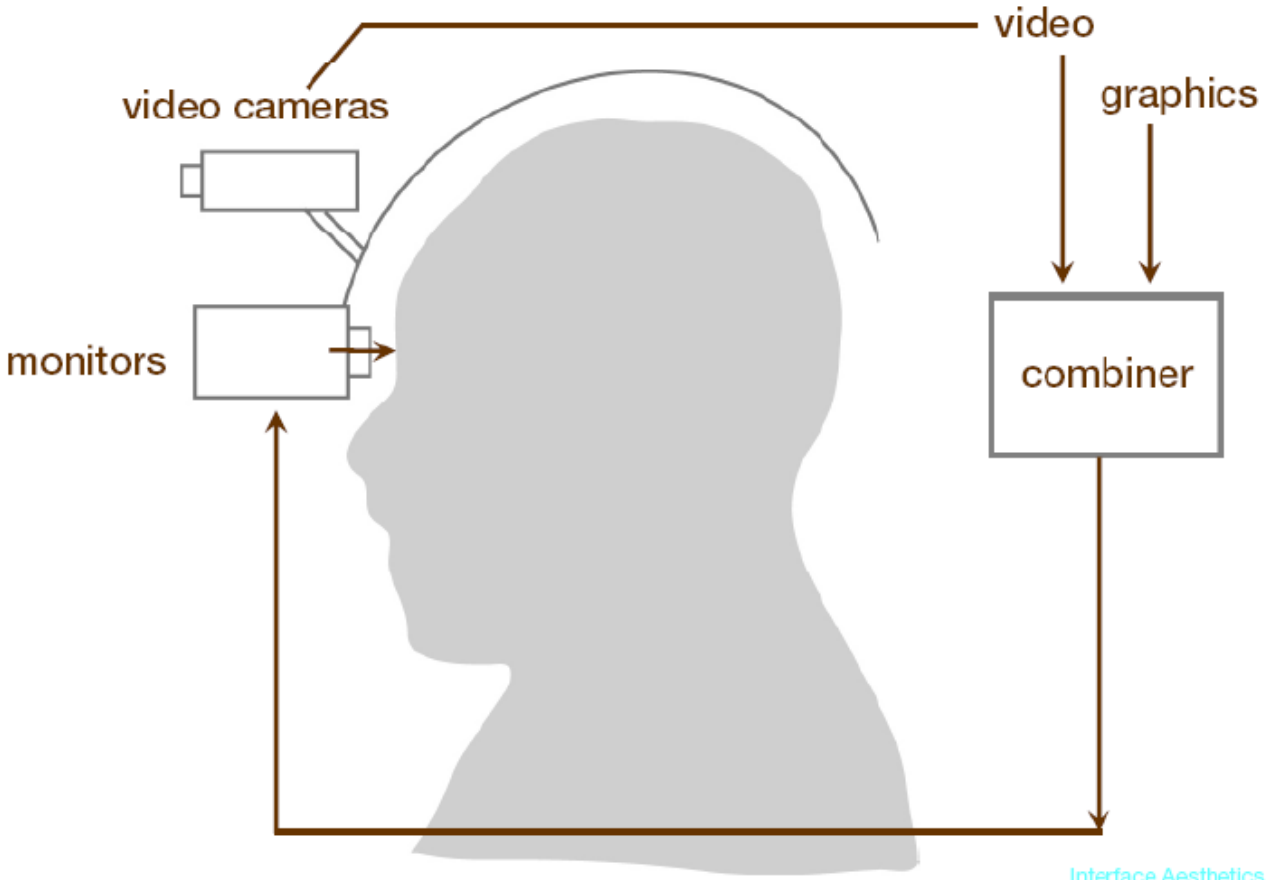
AR Displays – CAVE



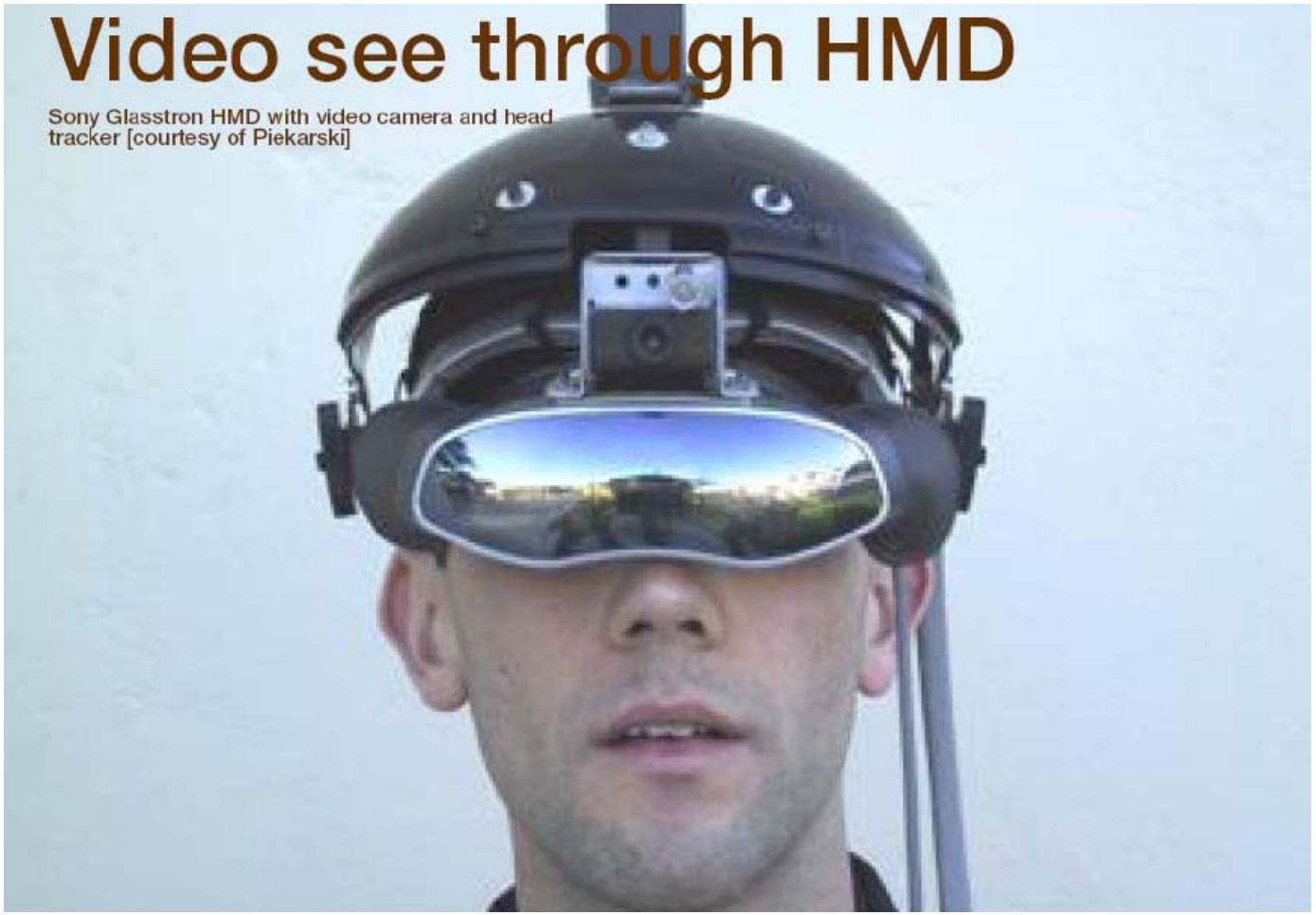
AR Displays - HMD

AR Displays – HMD Video

[adopted from Billinghurst & Ollila]

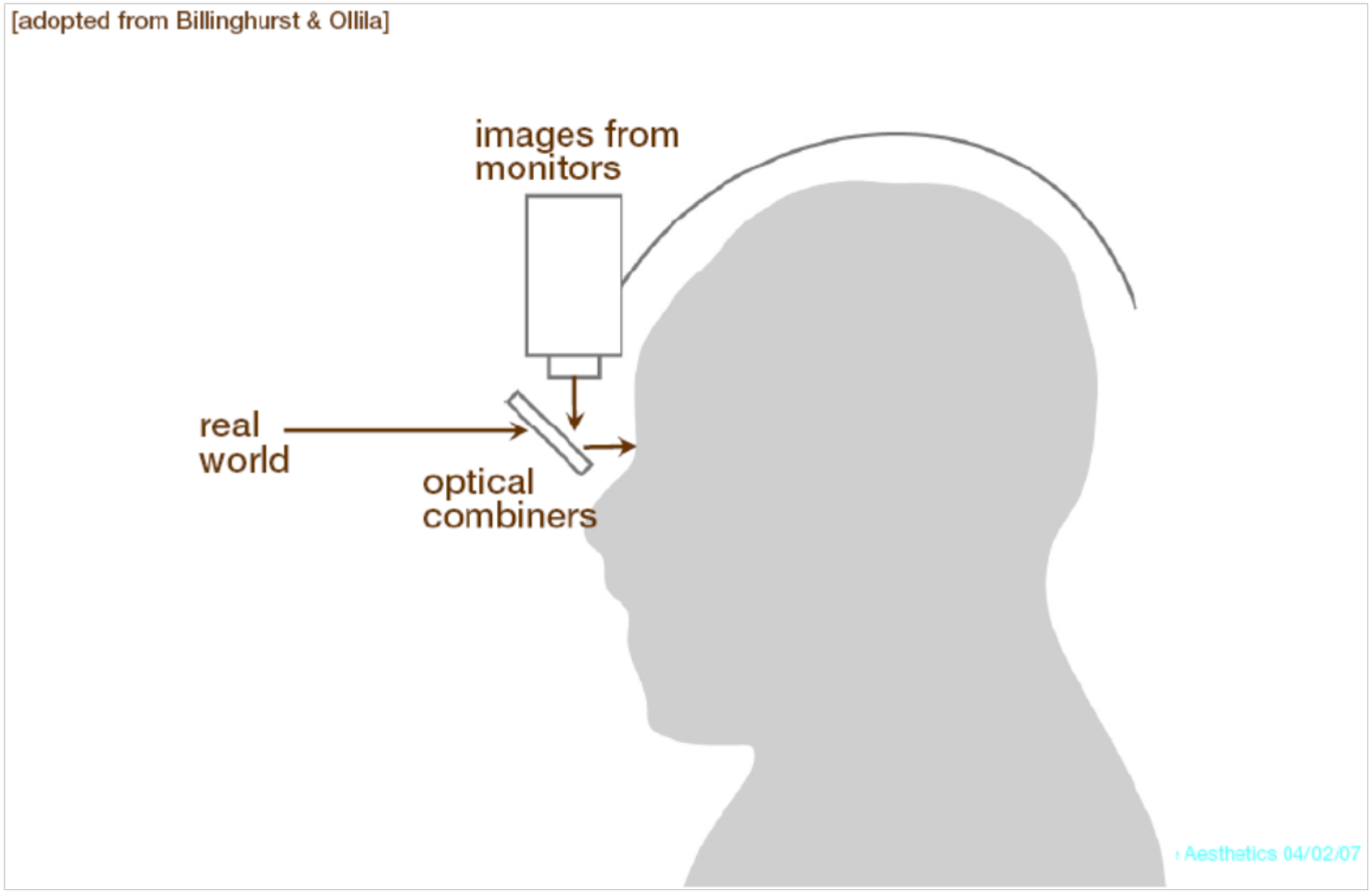


AR Displays – HMD Video



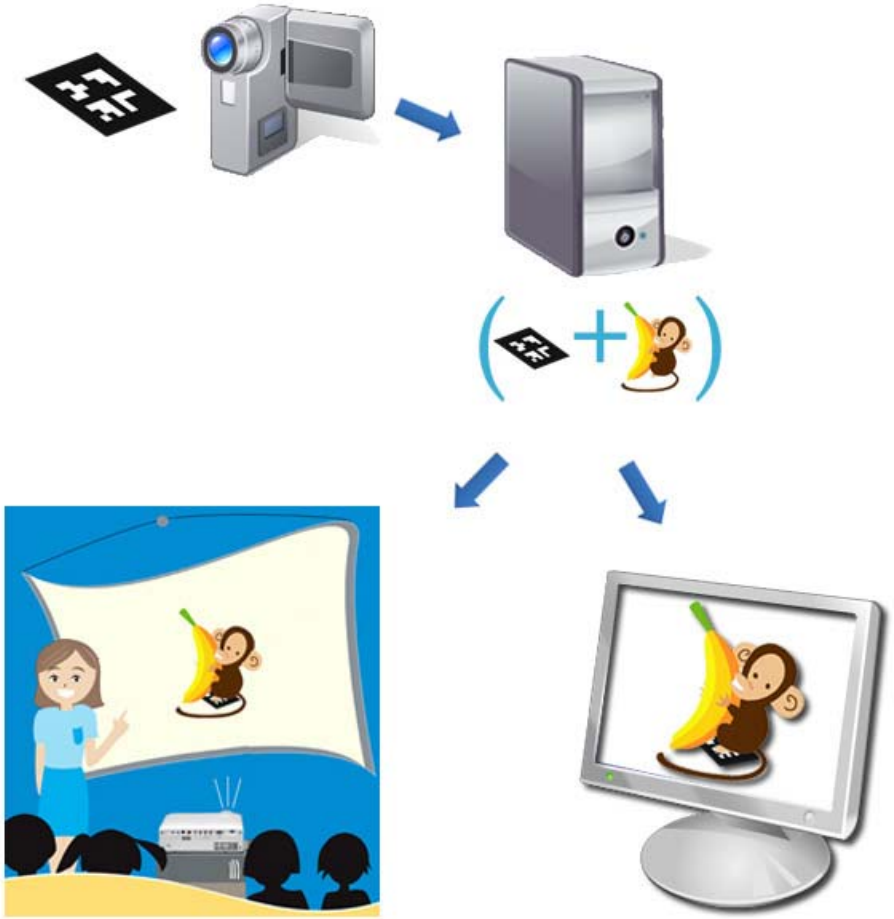
AR Displays – Optical HMD

le cnam



AR Displays – Optical HMD

AR System Design



Chaîne de production

le cnam

API QR Code

le cnam

- http://www.shiffman.net/p5/pqrcode/
- Télécharger l'API processing
- Sont inclus les sources et les exemples
- L'installation … comme d'hab …. On copie dans le répertoire library
- Les QR codes sont très largement utilisés au Japon
- Données plus importantes que les codes barres (~100 octets)

Squelette d’application

le cnam

L’initialisation

- Déclarer l’espace de nom

```
import pqrcode.*;
```

- Créer et initialiser un objet decoder

```
Decoder decoder;
void setup() {
  decoder = new Decoder(this);
}
```

Squelette d’application

le cnam

L’utilisation

- Demander au décoder de décoder une image

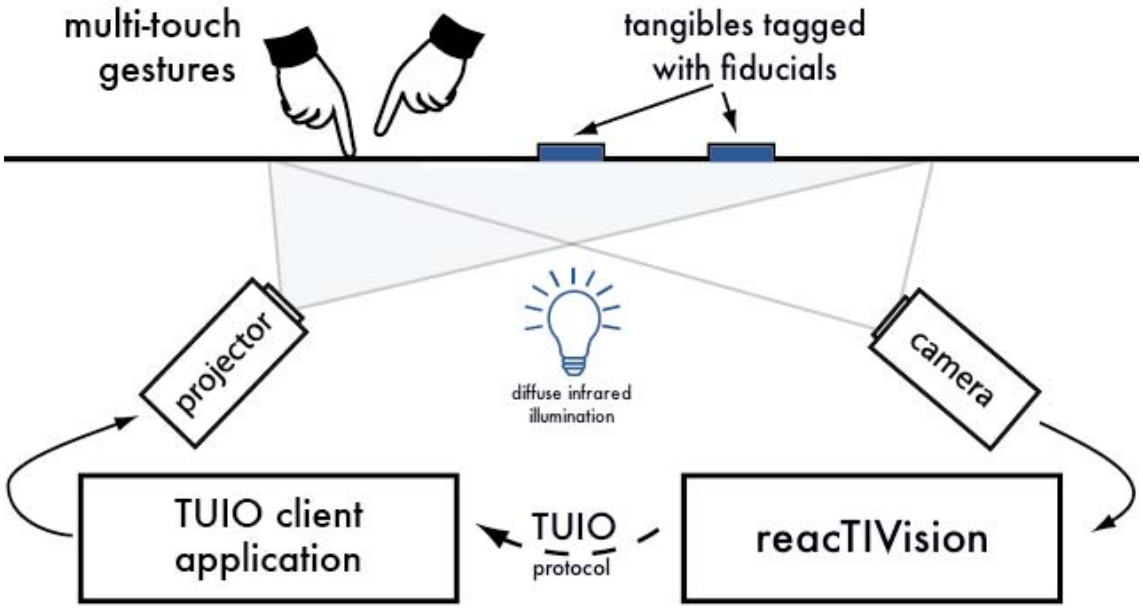
```
PImage img = loadImage("qrcode.png");
decoder.decodeImage(img);
```

- Récupérer le résultat dans la callback

```
void decoderEvent(Decoder decoder) {
  String statusMsg = decoder.getDecodedString();
  println(statusMsg);
}
```

Fiducial marker

- ReacTIVision



ReacTIVision vs QRCode

- QRCode n’est pas fait pour faire de la reconnaissance en temps réel
  - Ça fonctionne par snapshop : la chose m’intéresse je scanne son QRCode pour avoir plus d’informations
  - Possible de faire de la reconnaissance à la volée
  - Encore moins fait pour faire du tracking d’objets
- TUIO fait du tracking temps réel sur 3 degrés de liberté
  - Les deux translations dans un plan parallèle à la caméra
  - La rotation selon la normale à ce plan
  - Pas d’information de profondeur associé aux marqueurs 2D
- Possibilité dans le protocole TUIO 2.0 de faire du suivi en 3D

Installation ReacTIVision

le cnam

- D’abord le moteur de vision
- Puis le client TUIO pour que notre application communique avec le moteur de vision (pour nous le client processing)
- Les messages OSC encode un protocole TUIO
  - OSC (Open Sound Control) est un formalisme pour coder un contenu (comparable à XML ou JSON) qui émerge dans le monde multimédia (PureData, Max/msp, …)
  - TUIO est un protocole et une API pour les surfaces tangibles et multitouchs.

Utiliser TeacTIVision

le cnam

- Lancer le moteur de vision
- **Puis** lancer le client processing
- Le client doit effectuer les opérations suivantes :
  - Importer la librairie TUIO
  - Instancier un objet TUIO
  - Récupérer les objets TUIO vus
  - Implémenter les callbacks de la librairie TUIO qui permettent de détecter les changements d'états

Squelette d’application

L’initialisation

```
import TUIO.*;
TuioProcessing tuioClient;


void setup() {
  [...]
  tuioClient  = new TuioProcessing(this);
}
```

La PApplet courante est passée lors de la construction de l’objet TUIO. Cela suppose que les callbacks TUIO soient implémentées dans la classe courante.

Squelette d’application

le cnam

La boucle de rendu

```
void draw() {
  Vector tuioObjectList = tuioClient.getTuioObjects();
  for (int i=0;i<tuioObjectList.size();i++) {
     TuioObject tobj = (TuioObject)tuioObjectList.elementAt(i);
     translate(tobj.getScreenX(width),tobj.getScreenY(height));
     rotate(tobj.getAngle());
     println("ID = " + tobj.getSymbolID());
}
```

Squelette d'application

Les callbacks

```
// called when an object is added to the scene
void addTuioObject(TuioObject tobj) {
  println("add object "+tobj.getSymbolID()+" ("+tobj.getSessionID()+")
"+tobj.getX()+" "+tobj.getY()+" "+tobj.getAngle());
}

// called when an object is removed from the scene
void removeTuioObject(TuioObject tobj) {
  println("remove object "+tobj.getSymbolID()+" ("+tobj.getSessionID()+")");
}

// called when an object is moved
void updateTuioObject (TuioObject tobj) {
  println("update object "+tobj.getSymbolID()+" ("+tobj.getSessionID()+")
"+tobj.getX()+" "+tobj.getY()+" "+tobj.getAngle()
          +" "+tobj.getMotionSpeed()+" "+tobj.getRotationSpeed()+"
"+tobj.getMotionAccel()+" "+tobj.getRotationAccel());
}
```

Squelette d’application

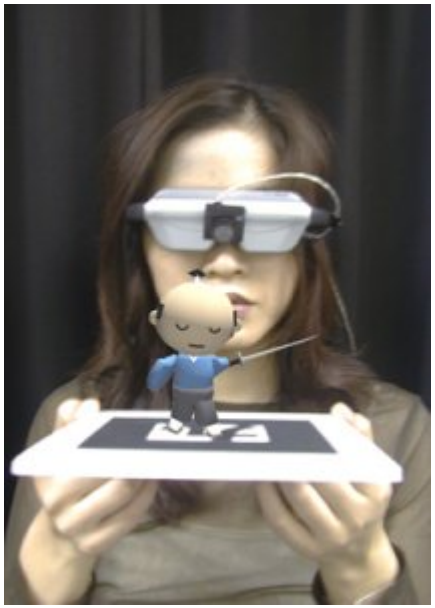

```
// called when a cursor is added to the scene
void addTuioCursor(TuioCursor tcur) {
  println("add cursor "+tcur.getCursorID()+" ("+tcur.getSessionID()+ ") "
+tcur.getX()+" "+tcur.getY());
}

// called when a cursor is moved
void updateTuioCursor (TuioCursor tcur) {
  println("update cursor "+tcur.getCursorID()+" ("+tcur.getSessionID()+ ") "
+tcur.getX()+" "+tcur.getY()+" "+tcur.getMotionSpeed()+" "+tcur.getMotionAccel());
}

// called when a cursor is removed from the scene
void removeTuioCursor(TuioCursor tcur) {
  println("remove cursor "+tcur.getCursorID()+" ("+tcur.getSessionID()+")");
}

// called after each message bundle
// representing the end of an image frame
void refresh(TuioTime bundleTime) {
  redraw();
}
```

ARToolKit

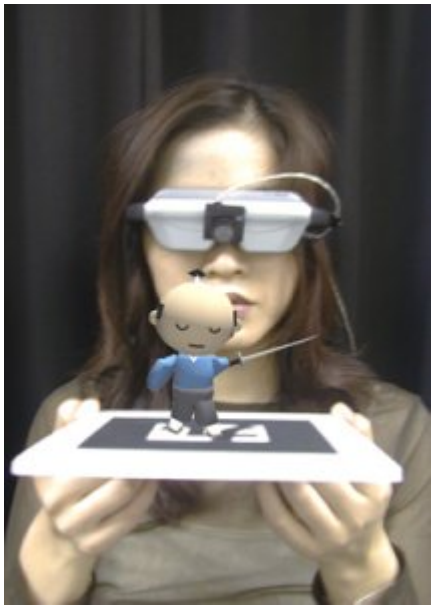
- Une bibliothèque logicielle permettant de construire facilement des applications de RA

  (Sur-impression d'images virtuelles sur dans un monde réel)

- ARToolkit se base sur des algorithmes de vision pour calculer et l'orientation de la (vraie) caméra par rapport à des marqueurs physiques



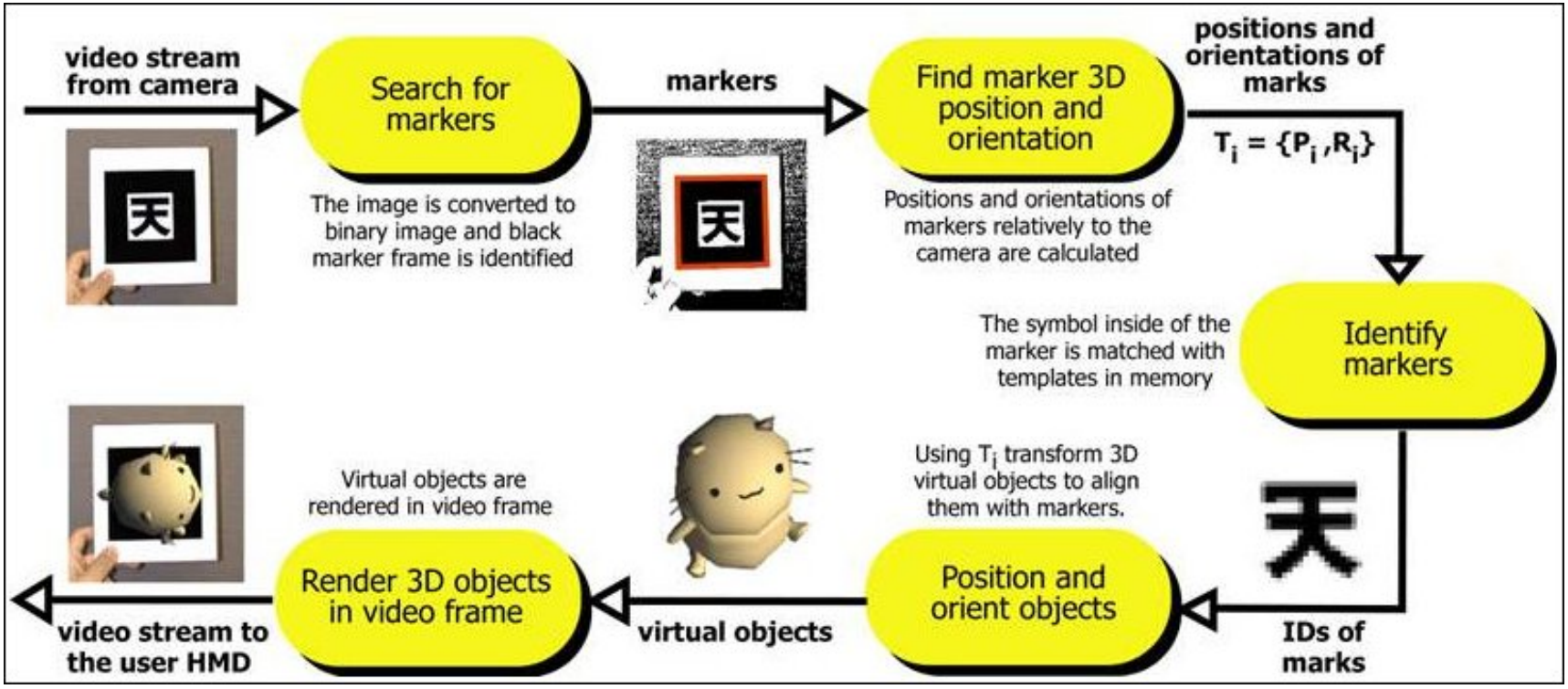
ARToolKit



- Avantage de la librairies :
  - Tracking de la position/orientation d'une camera
  - Tracking de codes dans un carré noir
  - Possibilité d'utiliser n'importe quel code
  - Calibration de caméra automatique
  - Rapidité pour permettre le temps réel
  - Sur plateformes Linux, MacOS and Windows
  - Code source distribué
  - **Et** un portage sous processing :
    - simpleARToolkit lui-même utilisant le binding java jARToolKit
    - http://code.google.com/p/simple-artoolkit-processing/

Fonctionnement d'ARToolkit

le cnam



Utiliser SimpleARToolkit

- Installer la librairie (comme d'habitude)
- Déclarer les espaces de noms nécessaires

  (attention dans les exemples il manque *video*)

```
import processing.opengl.*;
import processing.video.*;
import pARToolKit.*;
```

- Déclarer une référence SimpleARToolkit

```
SimpleARToolKit ar01;
```

- Instancier l'objet et déclarer une callback à exécuter

```
ar01 = new SimpleARToolKit(this,"patt.hiro");
ar01.register("showBox");
```

Utiliser SimpleARToolkit

- Si nécessaire afficher l'image acquise par la caméra

```
ar01.showImage();
```

- Rechercher les matchs

```
if (ar01.findMatch(100))
```

- Si match trouvé, on exécute la callback associée durant l'initialisation

```
ar01.showObject();
```

Utiliser SimpleARToolkit

- Exemple de callback

```
void showBox(SimpleARToolKit t) {
  pushMatrix();
  noFill();
  stroke(255,200,0);
  box(50);
  popMatrix();
}
```