Principes et mise en oeuvre d'une API de detection de markers physiques

E. Dubois
IRIT - Elipse

IV – l'AR-Toolkit

- Une référence :
  - HIT Lab, University of Washington, USA
    Mark Billinghurst, *grof@hitl.washington.edu*
  - https://artoolkit.org/

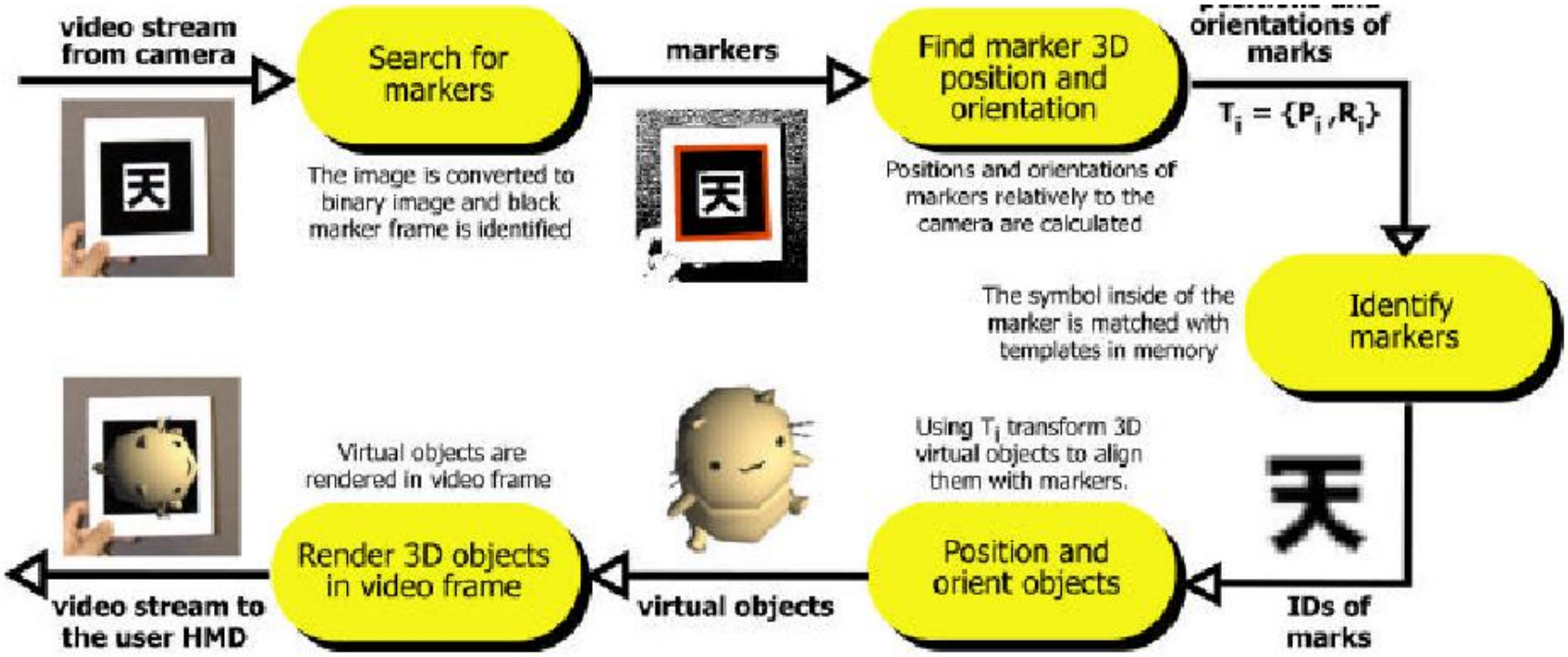
- Objectif
  - Superposer des informations numériques à un flux vidéo du monde physique

- Moyen
  - Librairie écrite en C (versionS Java existanteS)
  - Gratuit, open-source
  - Réservé à un usage non commercial
  - Exécutable de calibration de la caméra

Fonctionnement (1)

1. Acquisition :
   acquisition numérique d'un flux vidéo et seuillage
2. Détection :
   Recherche dans cette image de régions carrées.
   Pour chaque carré, identification du "pattern" situé à l'intérieur et mise en relation avec un des "patterns" prédéfinis
3. Localisation spatiale :
   Calcul de la position de la caméra réelle dans le repère associé au pattern (*matrice 3x4 de passage de la caméra au pattern*).
4. Traitement :
   Affichage du rendu graphique ou textuel, facilement "alignable" sur le monde réel.

Fonctionnement (2)



AR toolkit "patterns"

- Exemples de "pattern" détectables par l'AR toolkit.

Principe de mise en oeuvre

- 6 étapes
  1. Initialisation de l'application
  2. Capture d'une image vidéo
  3. Identification des marqueurs (situés dans les patterns)
  4. Calcul de la matrice de passage pattern – caméra
  5. Affichage du rendu
  6. Fermeture du canal d'acquisition vidéo

Code C / C++ | Code JAR Toolkit | Code NyARToolkit

Multiples versions

- OS X, iOS,
- windows,
- Android,
- Linux
- Bindong Unity

… et en 2D !

TopCode Library

http://users.eecs.northwestern.edu/~mhorn/topcodes/

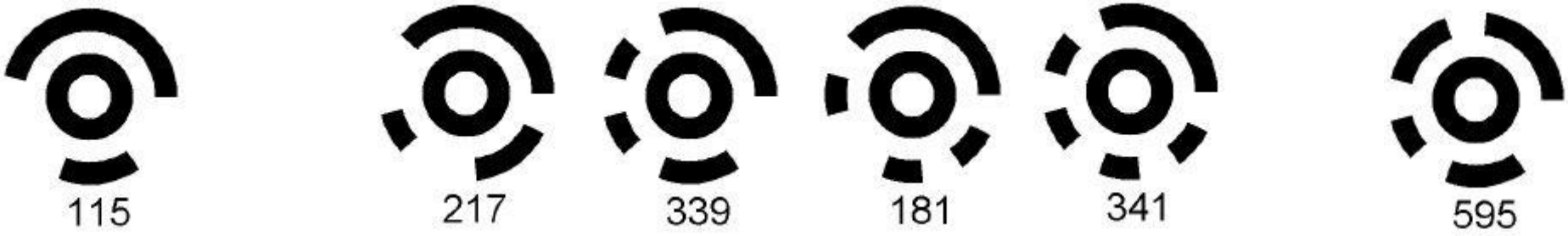
Introduction

- Encodage interne aux markers
  - ➔ pas besoin de faire apprendre les codes à l'API
  - ➔ codes utilisables prédéfinis (99 au total)
- Petite taille détectable
  - Minimum 25x25 pixels
- Codage
  - Noyau 100% Java

- Remarques complémentaire
  - Gratuit et OpenSource
  - Camera perpendiculaire à surface d'utilisation des Marker

TopCode Markers

- Combinaison de petits / moyens / longs arcs autour d'un cercle plein
  - 0, 1, 2, 3 ou 4 arcs de petite taille
  - 1, 0 ou 2 arcs de taille moyenne



- Les 99 markers

Le code : initialisation

- this.webcam.initialize();
  - Initialise la caméra (paramètre intrinsèques)
- this.webcam.openCamera(640, 480);
  - Connexion à la caméra
  - Spécifier une résolution supportée par la caméra
- animator.start();
  - Démarre un timer, qui envoie des *ActionEvents* à intervalle régulier (10 Hz)
  - *ActionEvents* captés dans *actionPerformed(…)*

Le code : détection et identification

- codes = scanner.scan (
  webcam.getFrameData(),
  webcam.getFrameWidth(),
  webcam.getFrameHeight());
  - Détecte les markers visibles dans la frame capturée
  - *codes* est de type *list<TopCode>*

- image = scanner.getImage();
  - Extrait l'image de la frame capturée pour affichage ultérieur éventuel

Le code : identification et traitement

- for (TopCode top : codes)
  - Parcourt *codes* (liste de *TopCode* détectés) et stocke l'item en cours dans *top*

- Méthodes d'accès au …
  - top.getCode(); numéro du TopCode
  - top.getDiameter(); diamètre
  - top.getCenterX(); abscisse du centre
  - top.getCenterY(); ordonnée du centre
  - top.getOrientation(); orientation du TopCode en radian

En savoir plus …

- TopCode aussi en
  - C++
  - Version Android

- Nombreuses autres API pour détection et localisation de markers physiques …

Utilisation de l'AR-toolkit en C (1)

- Init()
  - Ouvre le canal vidéo et initialise les paramètres par défaut de l'AR toolkit
    (fichier de calibrage de la caméra, fichiers de description des patterns)
  - Détermine la taille de la fenêtre d'affichage.

Utilisation de l'AR-toolkit en C (2)

- À utiliser dans la "*mainLoop*"
  - arVideoGetImage() : capture une frame du flux vidéo
  - argDispImage() : affiche la frame vidéo capturée
  - arDetectMarker() : construit un tableau (en C) d'identifiant des marqueurs détectés
  - arGetTransMat() : calcule la matrice de passage entre le repère du pattern et la caméra (réelle)
  - draw() : affiche le rendu graphique, textuel, etc.

1. Code pour initialisation

```
main( )
{
// INITIALISATION
      if (arVideoOpen("") < 0 ) exit(0);                    //open the video path
      if (arVideoInqSize(&xsize, &ysize) <0) exit(0);                  // find the size of window
      if (arParamLoad(cparaname, 1, &wparam) < 0)    // set the initial camera parameter
                  {   exit(0);  }


// DEMARRAGE
      arParamChangeSize(&wparam, xsize, ysize, &cparam);
      arInitCparam ( &cparam);
      arParamDisp( &cparam );
      argInit( &cparam, 1.0, 0, 0, 0, 0 );                  // ouvre la fenetre graphique
      arVideoCapStart( );                                   //start video image capture
      bool m_localiserStarted = true;


      … // cf. next slide
```

2. Detecter les marqueurs

```
… // "MAIN LOOP"
while (m_localiserStarted)
{
        // CAPTURE D'UNE FRAME
        arVideoCapNext();
        if ( (dataPtr = (ARUint8 *) arVideoGetImage( )) == NULL) {  arUtilSleep(2);return;  }

        // DETECTION DES PATTERNS PRESENTS
        if( arDetectMarker(dataPtr, thresh, &marker_info, &marker_num) < 0 )  {  cleanup();  exit(0);  }

        // IDENTIFICATION DES PATTERNS PRE-CHARGE VISIBLE
        for( i = 0; i < objectnum; i++ )                                    // objectnum = nombre de pattern pré-
chargés
        {               k = -1;
                        for( j = 0; j < marker_num; j++ )
                        {                                                    // object[ ] = tableau de descripteurs des
                                if ( object[i].id == marker_info[j].id )    // pattern pré-chargés
                                {               if( k == -1 ) k = j;
                                                else {  if ( marker_info[k].cf < marker_info[j].cf ) k = j;  }    }
                        }
                        if( k == -1 ) {  object[i].visible = 0;  continue;  }

                        // CALCUL DE LA MATRICE DE PASSAGE
                        if  ( arGetTransMat(&marker_info[k], object[i].marker_coord, object[i].trans) < 0 )
                        {   object[i].visible = 0;  }
                        else {  object[i].visible = 1; }
        }
…
```

3. Dessiner sur un marqueur

```
…
// AFFICHAGE DE LA VIDEO
argDrawMode2D();
argDispImage( dataPtr, 0, 0);

// CALCUL ET AFFICHAGE DU GRAPHIQUE POUR CHAQUE PATTERN DETECTE
for( i = 0; i < objectnum; i++ )
{
    if( object[i].visible == 0 ) continue;
    argConvGlpara(trans, gl_para);
    argDrawMode3D();

    argDraw3dCamera(0, 0);
    glMatrixMode(GL_MODELVIEW);
    glScalef(1,-1,1);
    glMultMatrix(gl_para);

    // INSTRUCTIONS DE DESSIN
    …
}
// AFFICHAGE DU BACK BUFFER
argSwapBuffers( );

…
```

4. Terminer la boucle d'événements

```
…
while (PeekMessage( &WmMsg, NULL, 0, 0,PM_REMOVE))
{
        int t;
        if ( (WmMsg.message == 257) && (WmMsg.wParam == 89) )
        {
                t = increaseThresh();  TRACE("THRESH++ = %d\n",t);
        }
        /* else … */

        DispatchMessage(&WmMsg);
}
}
```

- Documentation en ligne : http://artoolkit.sourceforge.net/apidoc/index.html

Utilisation de l'AR-toolkit en Java : la JAR-Toolkit (1)

- Différentes versions (n’est plus maintenue)
  - GL4Java
    - En baisse
  - JOGL
    - A voir (sensiblement similaire à la suivante)
  - Java 3D
    - Illustrée ici
    - Nécessite le Java3D – Direct X

Utilisation de l'AR-toolkit en Java (2)

- Déclaration

```
private JARToolKit3D  m_JARToolKit3D = null;
```

- Initialisation (dans le constructeur par exemple)
  - Création de l'instance unique (frameGrabber)

```
m_JARToolKit3D = JARToolKit3D.create();
```

  - Initialisation des paramètres de calibrage de la caméra

```
m_JARToolKit3D.initialize(
        JARToolKit3D.getAbsolutePath("data/camera_para.dat"));
```

(remplace les / par \\ …)

  - Insertion de la JARToolkit dans le graphe de scène

```
locale.addBranchGraph(
        m_JARToolKit3D.createViewingBranch(canvas));
```

Utilisation de l'AR-toolkit en Java (3)

- Initialisation (suite …)
  - Création du BackGround Java3D (et ajout)
    contentsTransGr.addChild(m_JARToolKit3D.createBackground());
  - Création du *behavior* gérant la détection des patterns de la jARToolkit (et ajout) m_arBehavior=JARToolKit3D.createRecognition();
    contentsTransGr.addChild(m_arBehavior);
    - Modifie la matrice d'un TG des que le pattern associé est détecté
  - Création d'un TransformGroup par pattern à localiser
    objTrans = m_JARToolKit3D.createPatternTransform(
    JARToolKit3D.getAbsolutePath("data/pattern/hiroPatt"), true);
  - Association de ce transformGroup au *behavior* de tracking (patternDriven-TransformGroup) m_arBehavior.registerObject(objTrans);
  - Ajout des objets Java3D attachés au patternDriven-TransformGroup

Utilisation de l'AR-toolkit en Java (4)

- Limite :
  - Behavior de détection de pattern built-in, donc encapsule
    - la détection
    - Localisation (calcul matrice de transfert)
    - Modification (mise à jour matrice de transformation du TG)
    - Redraw de la scene sur la base du graphe de scene
  - Comment intervenir au cours du traitement des patterns ?
    - Pour déclencher une action
    - Pour faire un usage différent des données de localisation

Utilisation de l'AR-toolkit en Java (5)

- Solution évidente …
  - Hériter de *arBehavior*
  - Surcharger la méthode *processStimuli* du behavior

- … MAIS impossible
  - jARToolkit.createRecognition()
    - créé une instance du *arBehavior*
    - Procède à l'initialisation de plusieurs parametres inaccessible en dehors de la méthode *createRecognition*()
    - Méthode *createRecognition*() est private … pas d'héritage possible de jARToolkit !

Utilisation de l'AR-toolkit en Java (6)

- Solution 2 : Création d'un *behavior* gérant la détection des patterns de la jARToolkit

  - Condition d'activation
    - private WakeupOnElapsedTime condition = new WakeupOnElapsedTime(20);

  - Constructeur
    - Utiliser un *patternDrivenTransformGroup* en paramètre

  - Réponse à l'évènement (*ProcessStimuli*)
    - ```
      // ARPattern. getTransMatrixJava3D(matrix) == true,
      // if the pattern was recognized, else false
      if ( (m_PDTG.getPattern()).getTransMatrixJava3D(matrix) )
      {
         // Mise à jour de la matrice de transformation du TransformGroup associé
         m_PDTG.update();

         // On récupère la matrice de transformation spéciale Java3D
         Matrix4d m = m_PDTG.getTransformation() ;
         …
      }
      ```
      ```

Configuration de la machine

- Classpath
  - Contient le java3D-Utils-src.jar
- Dll ARToolkit (JNI)
  - Copiées dans le répertoire d'exécution
- Path
  - Ajouter le jARToolkit\bin
- NOTE : c'est du java pour PC …

Utilisation de la Java ARToolkit : NyARToolkit

- Lien
  - http://nyatla.jp/nyartoolkit/wiki/index.php?FrontPage.en
- Avantages
  - Maintenue : évolutions constantes
  - Différentes versions : Java (JMF, Java3D, Jogl, QT), Android, C#, ActionScript, C++
  - Version objet
- Inconvénients
  - Tous les commentaires en Japonais
  - Pas de suivi simultané de multiples marqueurs
    - Patch requis

Utilisation

- Initialisation
  - Charger les pattern et stocker les ARCodeIndex générés

    ```
    ar_codes[0] = new NyARCode(16, 16);                  // taille du pattern
    ar_codes[0].loadARPattFromFile(CARCODE_FILE1);    // fichier de pattern
    ```
  - Création d'une instance de dispositif de capture et initialisation des paramètres de calibrage du dispositif

    ```
    ar_param = new J3dNyARParam();
    ar_param.loadARParamFromFile(PARAM_FILE);
    ```

    Instance créée à l'intérieur d'une classe en charge de la détection d'un pattern (cf. code exemple : Class NyARJava3D) ou de multiple marker (cf code exemple : Class MultipleJava3D_2_5)

- Création du graphe de scene incluant
  - Universe, Locale, View, ViewPlatform,
  - Background (grab du dispositif de capture)
  - Les TransformGroup pères de chaque sous graphe de scene a piloter par un pattern

- Création d'un manageur de Comportement (Java3D) pour gerer la detection / reconnaissance

  ```
  nya_behavior = new  NyARMultipleMarkerBehaviorHolder(ar_param, 30f, ar_codes, marker_width, 2);
  ```

- Configuration du manageur de behaviour
  - Affectation des TG à modifier

    ```
    nya_behavior.setTransformGroup(transformGroups[0], PATT_HIRO_ID);
    ```
  - Affectation du BAckground

    ```
    nya_behavior.setBackGround(background);
    ```
  - Ajout dans le graphe de scène

Pour aller plus loin …

- Rôle et fonctionnement du NyARMultipleMarkerBehaviorHolder
  - Créé un Raster (buffer video) associé à l'instance de capture :

    ```
    J3dNyARRaster_RGB(this._cparam, this._capture.getCaptureFormat())
    ```

  - Créé un détecteur de marqueurs associé à l'instance de capture, au ARCodeIndex, à la taille des markers, au raster :

    ```
    NyARDetectMarker(this._cparam, i_ar_code, i_marker_width, i_markers,
    this.nya_raster.getBufferType());
    ```

  - Créé un comportement (Java3D) de reconnaissance de marqueurs multiples : **class NyARMultipleBehavior**
    - **ProcessStimulus : Reveillé toutes les millièmes de secondes**
    - Constitue une « stack » de carré détecté dans le raster et pourchacun le tag avec un ARCodeIndex le plus probable

      ```
      found_squares = related_nya.detectMarkerLite(raster, 100); // 100 = seuil de la photo
      ```

    - Pour chaque élément de la stack, récuperation de la Matrice de transformation

      ```
      related_nya.getTransmationMatrix(i, src);
      ```

    - Mise à jour du tranform group associé au pattern identifié par le ARCodeIndex et génration d'un evenement

      ```
      listener.onUpdate(related_nya.getARCodeIndex(i), t3d);
      ```

Extensions de l'AR toolkit

- Version Java
  - http://www.c-lab.de/jartoolkit/ (pas maintenue)
  - NyARToolkit : http://nyatla.jp/nyartoolkit/wiki/index.php (pas encore pu la compiler …)
- Version Pocket-PC
  - http://www.ims.tuwien.ac.at/research/handheld_ar/_developer/artoolkit.php