Escola Universitària
Politècnica de Mataró

Centre adscrit a:

UNIVERSITAT POLITÈCNICA
DE CATALUNYA

Ingeniería Técnica en Informática de Gestión

Proyecto Final de Carrera

Aplicaciones de la Realidad Aumentada

Memoria

Guido A. Ciollaro Rodrigo-Magro
PONENT: Catalina Juan Nadal

PRIMAVERA - 2011

TecnoCampus
Mataró-Maresme

Resum

Aquest projecte consisteix en mostrar que és la Realitat Augmentada, quin és el seu funcionament i fer unes demostracions funcionals basades en els coneixements adquirits.
Al començament es realitzarà una explicació referent a la Realitat Augmentada, on i com va començar, les seves característiques principals i en què es basen els seus models. A continuació, s'oferirà una introducció de FLARToolKit, una llibreria que permet crear Realitat Augmentada per a Adobe Flash. D'aquesta indicarem el mode de funcionament intern en les seves diferents fases. Al capítol següent, el quart, s'explicarà la metodologia a seguir durant les demostracions del Capítol 5. En aquest, s'exposaran un seguit de casos d'estudi guiats per la metodologia prèviament explicada. Es fixarà l'objectiu de cadascun, així com la seva implementació i els problemes sorgits a l'hora de realitzar-los. Per acabar, s'avaluarà breument el producte en si. Al Capítol setè, ens endinsarem al futur de la Realitat Augmentada, descobrint com s'aplica avui dia i a què podria arribar en un futur proper. Finalment, es realitzarà una avaluació crítica del treball dut a terme i es resumiran les conclusions obtingudes.

Resumen

Este proyecto consiste en mostrar que es la Realidad Aumentada, como funciona y hacer unas demostraciones funcionales en base a los conocimientos adquiridos.
Al comienzo se realizará una explicación a cerca de la Realidad Aumentada, donde comenzó, sus principales características y en que se basan los diferentes modelos. Luego, se hablará sobre FLARToolKit, una librería que permite crear Realidad Aumentada para Adobe Flash, indicando la manera en que funciona internamente y sus diferentes fases. En el siguiente capítulo, se explicará la metodología que se seguirá en las demostraciones del Capítulo 5. En dicho capítulo, se expondrán diferentes casos de estudio siguiendo la metodología previa. Se explicará el objetivo de cada caso de estudio así como su implementación y los problemas encontrados a la hora de realizarlos, para luego hacer una breve evaluación del producto. En el Capítulo 7, se hace una mirada al futuro de la Realidad Aumentada, encontrando como es aplicada hoy en día y como podría ser en un futuro cercano. Finalmente se realiza una evaluación crítica del trabajo hecho y se resumirán las conclusiones obtenidas.

Abstract
This project consists on showing what Augmented Reality is, how it works and do a functional demonstration based on the acquired knowledge.
An explanation about Augmented Reality will be made, its history, main features and how the different kinds of Augmented Reality work. Later, we will talk about FLARToolkit, a library that allows creating Augmented Reality for Flash, indicating the way that it works internally and its different phases. In the next Chapter, the methodology that will be applied to the demonstrations of Chapter 5 will be explained. In this Chapter, the different case studies will be exposed following the previous methodology. The objectives of each study case as well as its implementations and the problems found when testing them will be explained, for later do a brief evaluation of the product. In Chapter 7, we will have a look at the future of Augmented Reality, finding how it is applied nowadays and how it could develop in the near future. Finally, a critical evaluation of the work done is made and the achieved conclusions will be summarized.

Índice

Índice de Figuras

Glosario de Términos

AR	Augmented Reality
ARToolKit	Augmented Reality Toolkit
FLARToolKit	Flash Augmented Reality ToolKit
Marker	Patrón físico reconocible para AR
Thresholding	Umbralizado
Threshold value	Valor de umbralizado o Valor de umbral
Homography transformation	Transformación homográfica
DAE	Formato de COLLADA
WiiMote	Mando de la consola Wii
WiiFlash Server	Aplicación externa usada para conectar el WiiMote
Mixed Reality	Realidad Mixta

Introducción

La Realidad Aumentada consiste en añadir información digital al mundo real, generando de esta manera una especie de "aumento" en la cantidad de información percibida por los seres humanos sobre el mundo que los rodea. Esta clase de aumento puede tener un profundo impacto en la sociedad y en la manera en la que estamos acostumbrados a vivir. De todas formas, para que este cambio pueda producirse, la tecnología actual debe evolucionar. Este proyecto tiene dos objetivos principales. El primero es investigar las tecnologías que nos permiten, hoy en día, desarrollar aplicaciones de Realidad Aumentada. El segundo, es aplicar estas tecnologías para desarrollar unos casos de estudio simples para demostrar las diferentes posibilidades que nos ofrece la Realidad Aumentada. Estas demostraciones no pretenden crear grandes y complejas aplicaciones de Realidad Aumentada, sino servir de base y ejemplo para que cualquiera que esté interesado en la Realidad Aumentada pueda utilizarlas y expandirlas para crear su propia aplicación.

1. Objetivos

1.1 Propósito

Presentar las tecnologías que hoy en día se usan para crear aplicaciones de Realidad Aumentada, así como realizar unos casos de estudio mostrando las diferentes posibilidades que ofrece esta tecnología

1.2 Finalidad

Permitir al usuario interesado en la Realidad Aumentada conocer las tecnologías usadas hoy en día y tener una base sólida para crear su propia aplicación de Realidad Aumentada.

1.3 Objeto

Proyecto Final de Carrera, consistente de tres documentos, memoria, estudio económico y apéndices, que muestra toda la información necesaria para comenzar a desarrollar aplicaciones de Realidad Aumentada. Incluye también cuatro casos de estudio (con su correspondiente metodología, diseño e implementación) que pueden ser utilizados por el usuario como base y expandirlos.

1.4 Abasto

Se especifican los diferentes modelos de Realidad Aumentada que existen. Luego, se presentan las diferentes tecnologías usadas y e entra a detallar en profundidad cómo funciona el modelo basado en markers usado por la librería FLARToolKit. A continuación, se mostrarán y explicarán los diferentes casos de estudio. Finalmente se dará un vistazo al futuro de la Realidad Aumentada y sus posibilidades.

En los apéndices, se hallará información importante que no ha podido ser incluida en la memoria del proyecto

2. Background

2.1 ¿Que es la Realidad Aumentada?

Realidad Aumentada (AR, del Inglés Augmented Reality) es una técnica derivada de la Realidad Aumentada. El objetivo de la AR es combinar, usando entradas generadas por ordenador tales como sonidos o gráficos, el mundo real con elementos virtuales con el fin de aumentar la percepción de la realidad de una persona.

El aumento normalmente es en tiempo real. Estamos acostumbrados a la AR de formas tales como los marcadores en un partido de football retransmitido por la televisión. Con la ayuda de tecnología avanzada en AR, la información acerca del mundo que rodea al usuario se vuelve interactiva.

2.2 ¿Cómo funciona la Realidad Aumentada?

El proceso de AR se podría dividir en dos fases principales: Reconocimiento & Tracking y la fase de Rendering.

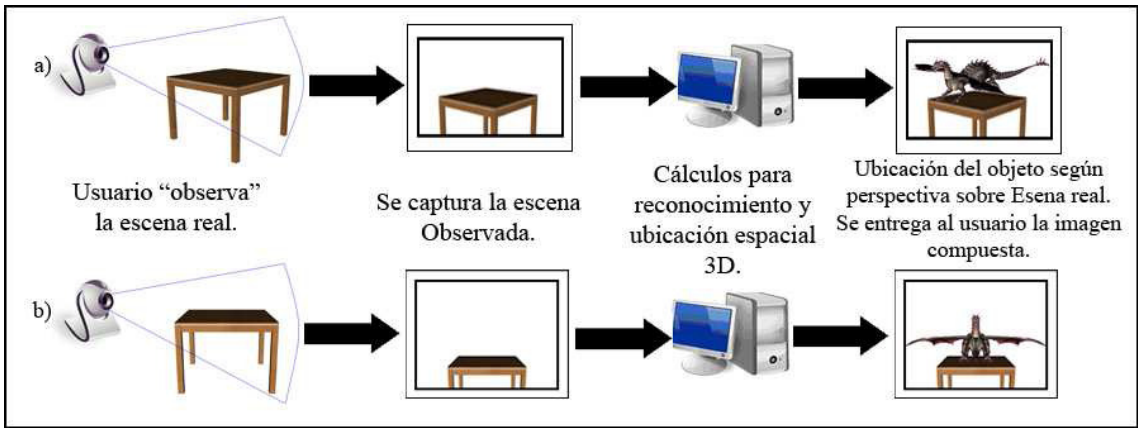

Figura 1 - Esquema de funcionamiento de un sistema de Realidad Aumentada típico. En b se ha girado el punto de vista que la cámara tenía en a, que a su vez hace que también el objeto virtual gire en relación al nuevo punto de vista.

Como se muestra en la Figura 1, la primera parte del proceso consiste en capturar la escena del mundo real. La escena es procesada y es cuando la fase de Reconocimiento & Tracking (o seguimiento) empieza.

Esta fase consiste en procesar la escena capturada por la cámara con el fin de solucionar uno de los principales problemas de la AR: el Viewpoint tracking (o Seguimiento del punto de vista). Esto es un problema clave porque la AR debe saber dónde y cómo posicionar los objetos virtuales para su correcta visualización por parte del usuario.

Hay tres modelos diferentes para conseguir solucionar el problema del viewpoint tracking:

- **Localización espacial usando GPS:** Este modelo es usualmente utilizado en Smartphone (tales como iPhone o BlackBerry) que incluyen cámaras y un sistema GPS capaz de detectar la posición del usuario. Usando estas coordenadas, el software calcula la posición de los objetos virtuales a añadir a la escena. Este modelos es mas impreciso comparado con los otros modelos, principalmente debido al margen de error en la triangulación de los satélites GPS, y por ello se tiende a utilizar en sistemas de AR donde la precisión no es crítica. Por el contrario, este sistema es más rápido que los otros debido a que solo necesita conseguir la posición del GPS y no ha de realizar el proceso de reconocimiento de imagen que tiene un coste computacional mucho más elevado.

Figura 2 – Wikkitude Drive: Software de Realidad Aumentada creado por la compañía Mobilizy para Smartphone que usa localización por GPS.

- **Reconocimiento espacial sin markers físicos (Markerless Tracking):** Esta es una técnica más compleja debido a que el software de AR debe ser capaz de

reconocer diferentes objetos que componen la escena del mundo real. Este proceso involucra muchas técnicas de visión artificial y su rendimiento varía dependiendo la cantidad de objetos a reconocer. Para mejorar el rendimiento del modelo de Markerless Tracking, solamente ciertos objetos son reconocidos tales como superficies, cara o manos humanos, formas específicas, etc.…

Figura 3 – Recognizr: El software de la compañía TAT es capaz de reconocer caras específicas y mostrar su perfil en diferentes redes sociales.

- **Localización espacial usando markers físicos (Marker Tracking):** Este modelo de AR está basado en la detección de lo que se conoce como "Markers". Un marker es un rectángulo (cuadrado) blanco y negro con un patrón asimétrico en el interior.

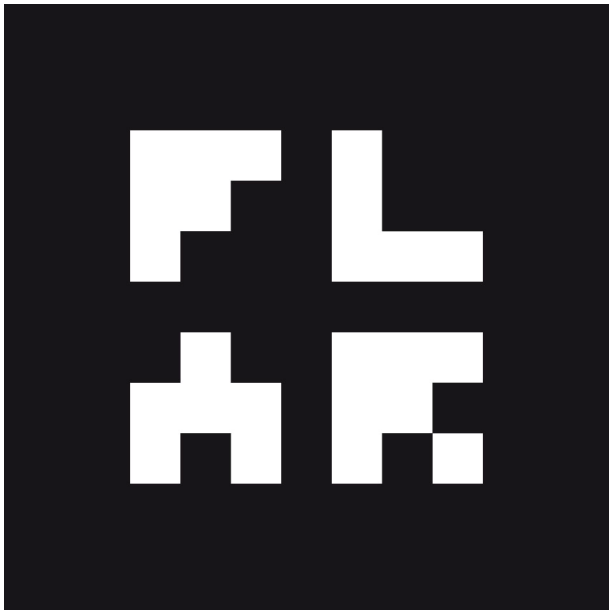

Figura 4– Ejemplo de un marker.

El reconocimiento del marker se hace por medio de librerías de seguimiento de visión (visión tracking libraries) para calcular la posición y orientación de los objetos 3D. Este modelo es muy rápido y preciso pero, como solo reconoce markers, falla a la hora de reconocer el entorno y eso genera que la sensación de AR se pierda dependiendo la situación.

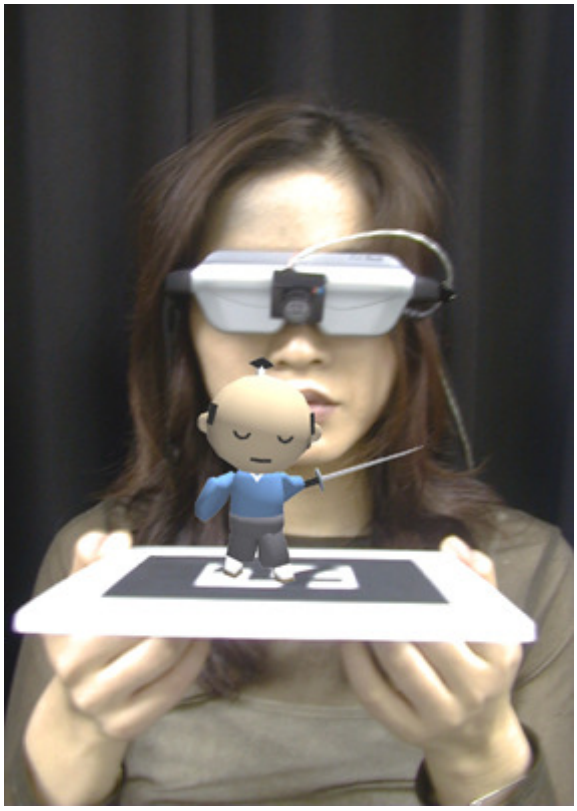

Figura 5 - Ejemplo de Realidad Aumentada usando markers.

Una vez acabada la fase de Reconocimiento & Tracking y el software de AR tiene toda la información acerca del entorno/marker (posición, orientación…), la fase de Rendering comienza. En esta fase, el software de AR utiliza la información de tracking de la escena del mundo real para añadir los objetos virtuales acorde a la información conseguida en la fase anterior. Todo este proceso se repite para cada frame.

En este proyecto, nos centraremos en el modelo de Marker Tracking y en una tecnología de tracking para AR específica; ARToolKit.

2.3 ¿Qué es ARToolKit?

ARToolKit (Augmented Reality Toolkit) es uno de las librerías de "vision tracking" más usadas para desarrollar aplicaciones de AR. Para ello, usa capacidades de "video tracking" para calcular, en tiempo real, la posición y orientación real de la cámara relativa al marker físico. Una vez que la posición real de la cámara se conoce, una cámara virtual puede ser posicionada en el mismo punto y se pueden dibujar modelos de ordenador 3D superpuestos al marker. Entonces ARToolKit resuelve dos de los

problemas claves de la Realidad Aumentada; El tracking del punto de vista y la interacción de objetos virtuales.

ARToolKit originalmente fue desarrollado por Hirokazu Kato del Nara Institute of Science and Technology en 1999 y fue publicado por la University of Washington HIT Lab. Actualmente es mantenido como un open source con licencias comerciales disponibles por parte de ARToolWorks.

Desde la publicación de ARToolKit, han aparecido muchas librerías que la han exportado a diferentes plataformas. Algunas de ellas son:

- **OSGART** – Una combinación de ARToolKit y OpenSceneGraph
- **ARTag** – Alternativa a ARToolKit que usa procesamiento de imágenes más complejos y procesamiento de símbolos para fiabilidad y resistencia a la luz. Solo licencias para uso no comercial.
- **Mixed Reality Toolkit (MRT)** – University College London
- **FLARToolKit** – Un port para Action Script 3 de ARToolKit para Flash 9+.
- **SLARToolkit** – Un port a Silverlight de NyARToolkit.
- **NyARToolkit** – Una librería de ARToolKit publicada para máquinas virtuales, particularmente Java, C# y Android.

En este proyecto se utilizara la librería FLARToolKit para estudiar los diferentes casos de estudio expuestos. FLARToolkit (Flash Augmented Reality ToolKit) fue adaptado a Action Script por Tomohiko Koyama, normalmente conocido por su sobrenombre “Saqoosha”. La primera versión del port fue publicada en Mayo 2008.

2.4 ¿Por qué FLARToolKit?

De todas las variantes de ARToolkit se decidió utilizar la versión para Adobe Flash, FLARToolkit, porque puede correr en cualquier navegador compatible con Flash Player. Hoy en día, casi todos los navegadores usan Flash y eso simplifica el proceso de descarga e inicialización de las aplicaciones. Usar FLARToolkit hace las aplicaciones de AR ligeras y fáciles de usar para el usuario. Esta es una ventaja frente a otras versiones como la de C++ en la cual el usuario debe descargar, instalar y finalmente

ejecutar la aplicación para hacerla correr, mientras que usando Flash estos pasos se hacen automáticamente.

Como una desventaja, el rendimiento de FLARToolkit, incluso que es bueno para aplicaciones simples, no es tan bueno como el rendimiento casi óptimo que puede conseguirse con aplicaciones hechas con C++ que corren a un nivel mucho más bajo.

FLARToolkit solamente es la librería que calcula la posición del marker en la escena, pero necesita utilizar algún tipo de motor 3D (render) para gestionar la escena tridimensional y presentar la imagen final de AR al usuario. Flash puede usar varios motores de render como: Aways3D, Sandy3D, PaperVision3D, Alternativa3D, etc.… Uno de los renders usados más común es PaperVision3D debido a que viene incluido con los ejemplos de FLARToolkit.

3. Funcionamiento interno de FLARToolKit

Internamente FLARToolkit ejecuta 7 fases para realizar el proceso de crear Realidad Aumentada: Capturar la imagen; Binarize input image; Labelling; Find squares; Matching with patterns; Calculate transform matrix; Render the 3D object. These steps are done throughout the whole execution of the application, for each frame of the captured video.

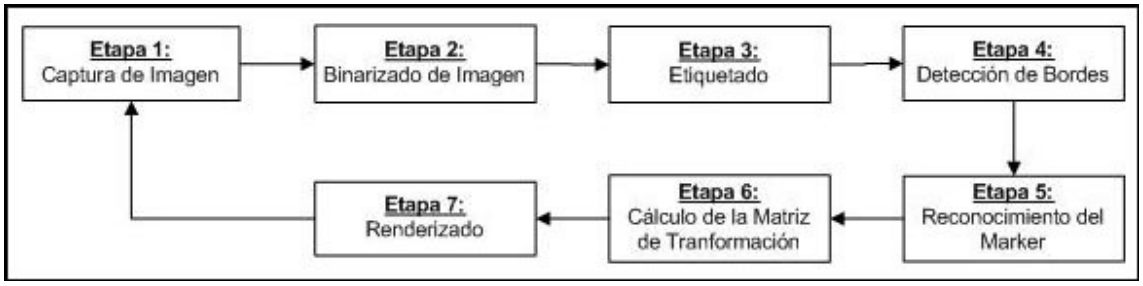

Figura 6 – Etapas en el funcionamiento interno de FLARToolkit.

3.1 Fase 1: Captura de imagen

Consiste en capturar la imagen raster o mapa de bits de la cámara para luego procesarla.

3.2 Fase 2: Binarizado de imagen

Esta fase consiste en transformar la imagen capturada a un formato más sencillo para optimizar el tiempo de detección del algoritmo. Para ello, la imagen es transformada a escala de grises, para finalmente realizar un thresholding (umbralizado) para obtener como resultado una imagen binarizada en blanco y negro (0 y 1). Esta imagen binarizada tiene un coste computacional mucho menor que una imagen en escala de grises o en color. Este proceso se muestra en la Figura 7.

Figura 7 – La imagen original se convierte primero a escala de grises y luego se umbraliza.

El thresholding es uno de los pasos más importantes dentro del proceso debido a que, dependiendo del valor de umbralizado (threshold value), el algoritmo puede detectar o no el marker y así mejorar o empeorar la experiencia del usuario.
FLARToolkit usa un threshold value fijo que está definido en el código fuente. Por lo tanto, la detección está afectada por las condiciones de luz del entorno, la calidad de la cámara o el material en el cual el marker está impreso (Figura 8). La situación ideal sería utilizar un threshold value variable que se ajuste automáticamente de acuerdo con los factores del entorno para incrementar el reconocimiento del marker en la escena.

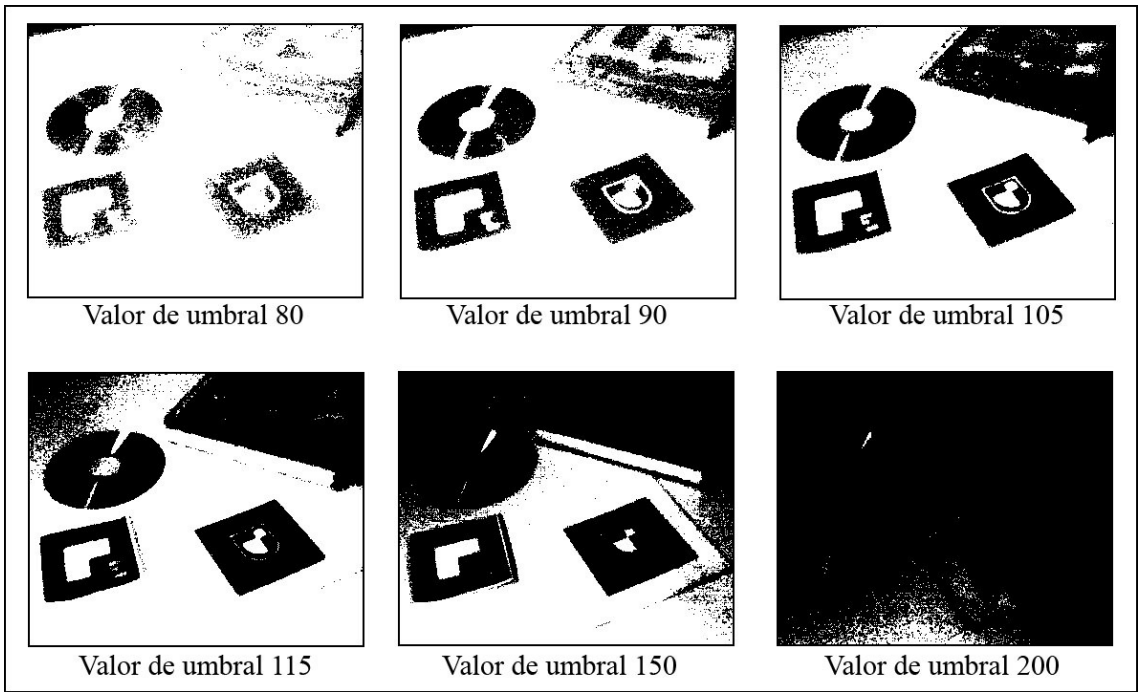

Figura 8 – Dependiendo del valor de umbral podemos pasar de una imagen blanca a una negra casi en su totalidad. El valor ideal en este ejemplo sería alrededor de los 105, donde cada objeto de la imagen puede ser separado del fondo claramente.

El framework para FLARToolKit llamado FLARManager implementa un threshold value variable y el programador puede incluso customizar el algoritmo para ajustar los valores según sus necesidades. Sin embargo, esto es algo que aún no está implementado nativamente en FLARToolKit.

3.3 Fase 3: Etiquetado

El siguiente paso en el proceso es el de detectar diferentes objetos dentro de la escena. Para ello, FLARToolKit detecta regiones de pixeles hasta notar un cambio brusco en ellos en cual caso agrupa los objetos por áreas o similitud entre áreas. Como se muestra en la Figura 9, los diferentes objetos detectados son etiquetados como candidatos a ser analizados para encontrar si coinciden con el marker o no.

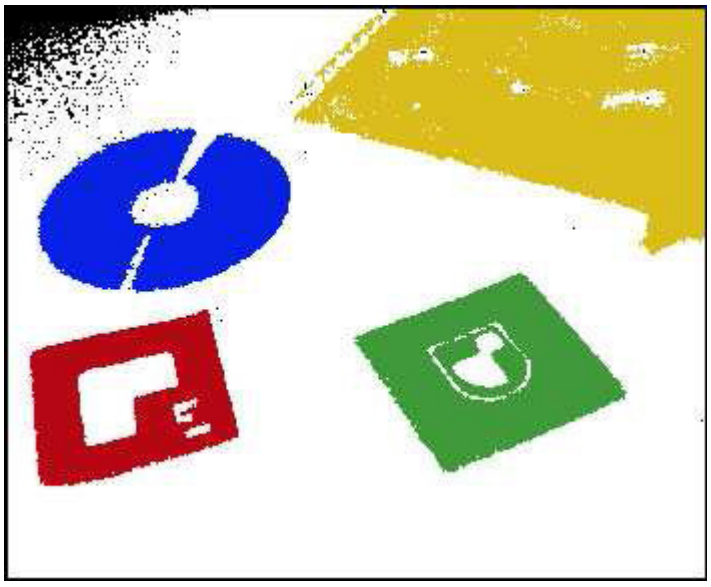

Figura 9 – Etiquetar es agrupar conjuntos de pixeles para detector similitudes entre ellos y así detector objetos.

3.4 Fase 4: Detección de bordes

Una vez los objetos en la escena han sido etiquetados, FLARToolkit procede a detectar los bordes de cada uno de ellos. De estos, FLARToolKit descartara los que no tengan una forma rectangular y calculara la posición de los vértices en la pantalla (plano XY) como se muestra en la Figura 10.

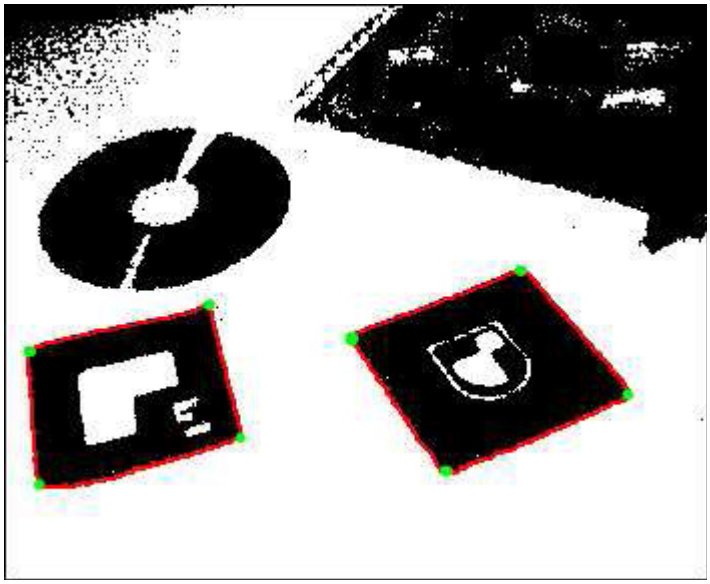

Figura 10 – Los bordes y vértices de los objetos rectangulares son detectados.

3.5 Fase 5: Reconocimiento del marker

El marker es fundamental para FLARToolKit debido a que es el patrón impreso en él, el cual permitirá establecer un punto de referencia para posicionar los objetos siguiendo su posición y orientación.

Para FLARToolKit, el marker debe ser rectangular (o cuadrado) y de el es extraído por defecto el 50% del área central. (Figura 11)

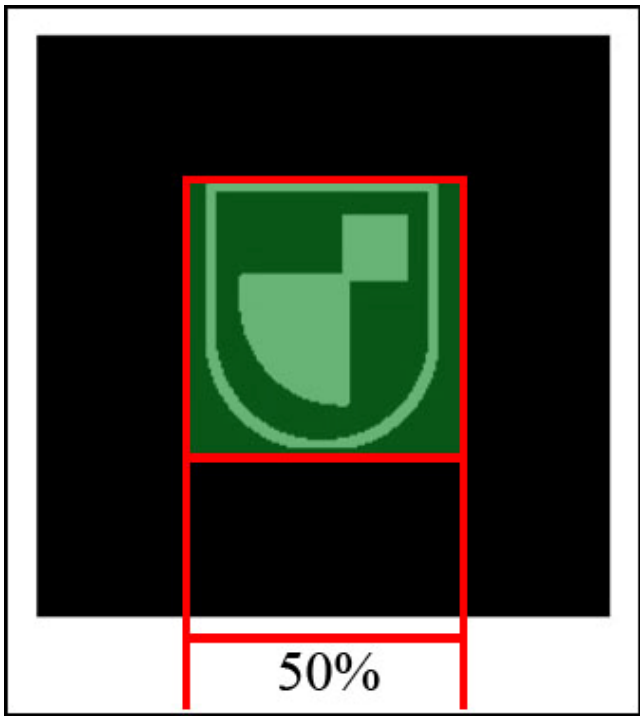

Figura 11 – Del marker, el área central es importante y debe contener una figura asimétrica de manera que sea fácil reconocer su orientación desde cualquier punto.

Para hacer el proceso de reconocimiento más rápido, FLARToolKit usa un fichero de extensión 'pat', el cual es simplemente un fichero de texto plano con el marker codificado en escala de grises y diferentes orientaciones.

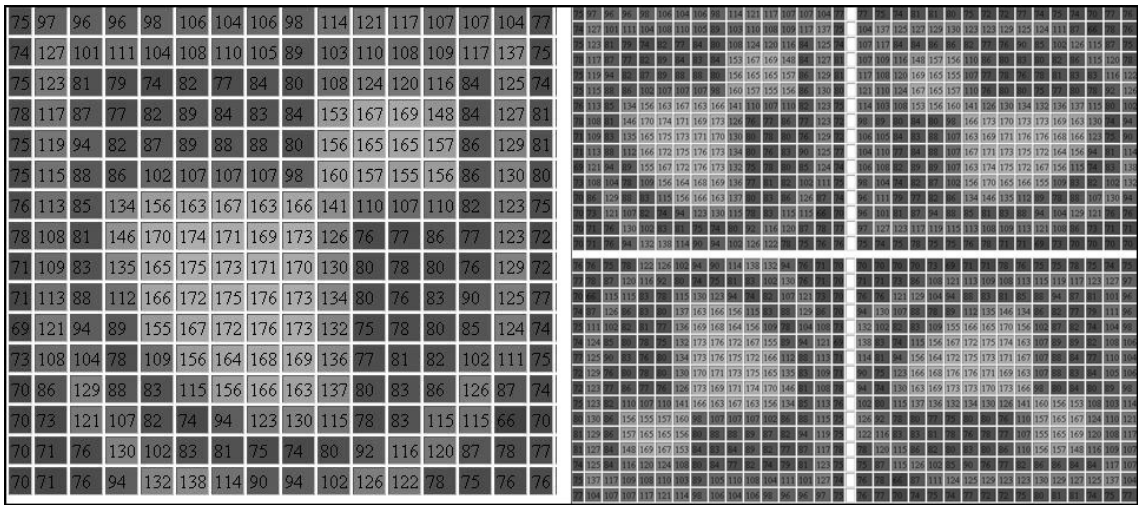

Figura 12 – Si se abre el fichero .pat, se verán los números que representan el marker en escala de grises.

Por defecto, FLARToolKit reconoce markers de 16x16 bits (como se muestra en la Figura 12), pero también acepta markers desde 4x4 hasta 64x64 bits. Con una resolución mayor, patrones más complejos pueden usarse pero el rendimiento tendrá un coste computacional mayor. Por otra parte, con una resolución menor, el reconocimiento es más rápido pero patrones simples deben usarse y, por este motivo, es más difícil saber exactamente en qué posición está orientado el marker.

En esta fase, el área central del marker se extrae utilizando una 'homography transformation' (o transformación homográfica). Debido a que el tamaño real del marker es conocido (definido en el programa y normalmente como un estándar de 8x8cm), puede hacerse una corrección a la imagen para determinar el símbolo exacto dentro del marker y así eliminar distorsiones causadas por la perspectiva. (Figura 13)

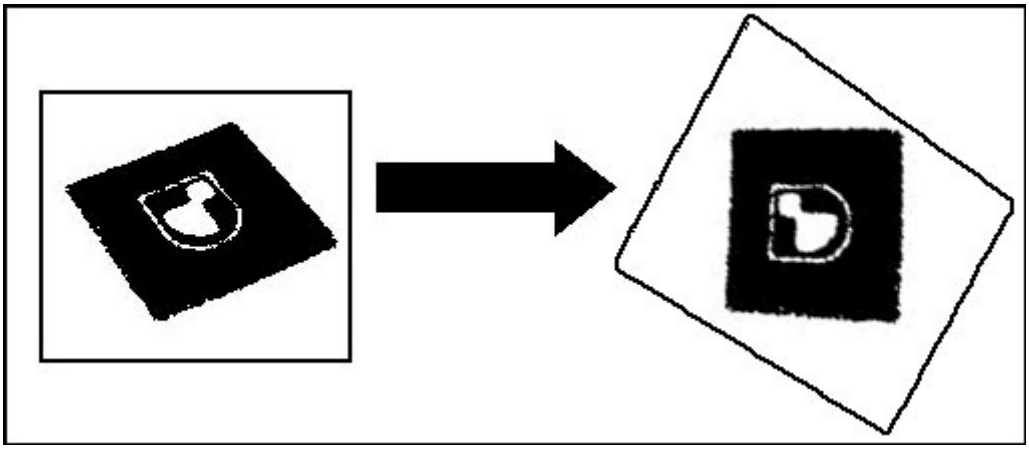

Figura 13 – Aplicando la transformación homográfica a los candidatos a markers, la imagen es deformada para extraer del centro el patrón sin deformaciones.

Luego, las imágenes extraídas de cada uno de los candidatos, son comparadas con los patrones definidos en el programa para su detección. La detección se hace por medio de comparar las imágenes con los patrones en diferentes posiciones y extraer el porcentaje de similitud. Basándose en esta puntuación, se decide finalmente cual de los patrones a detectar corresponde a una imagen de las candidatas. (Figura 14)

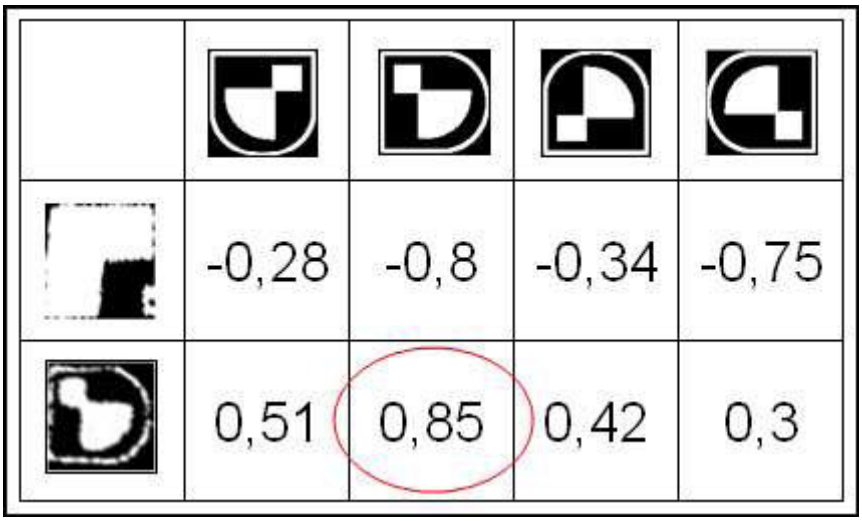

Figura 14 – Usando los patrones detectados, se estima la similitud con los patrones definidos en el programa.

3.6 Fase 6: Calcular la matriz de transformación

Basándose en la posición del vértice del patrón seleccionado en la imagen, la correspondiente matriz de transformación es calculada.

La matriz de transformación del marker es una matriz 4x4 que indica la posición tridimensional, rotación, etc. Esta matriz puede variar dependiendo del motor de render

usado; Por ello, FLARToolKit incluye funciones que permiten convertir de una matriz estándar a las versiones utilizadas por los diferentes motores de render.
Por ejemplo, en el motor de render Papervision3D, las columnas definen la escala, inclinación, rotación y posición the cada uno de los 3 ejes; Y la cuarta fila es usada únicamente como auxiliar para realizar multiplicaciones de matrices.

3.7 Fase 7: Render

Finalmente en esta fase, los objetos 3D son añadidos a la escena aplicando las correspondientes matrices de transformación y fusionados con la imagen original dando al usuario la escena final en Realidad Aumentada.
Una vez que estos pasos se han completado, el proceso vuelve al primer paso y avanza al siguiente frame de video.

4. Metodología

El objetivo de este proyecto es crear cuatro demostraciones de lo que se puede conseguir utilizando AR. Se ha optado por utilizar FLARToolKit para desarrollar dichas demostraciones. El entorno de desarrollo que se ha utilizado para crear estas demostraciones ha sido Adobe Flash Builder 4 (Versión 4.0.1).

Para cada caso de estudio, se utilizara un modelo de desarrollo en cascada, dado que los ejemplos que se crearan son suficientemente simples y están bien definidos.
Luego de haber definido el problema para cada caso de estudio, cada ejemplo se desarrollara siguiendo los pasos típicos de esta metodología:

1) Análisis: En esta fase, se definirán cuales son los requerimientos a cumplir.
2) Diseño: En esta fase, se explicara que diseño se seguirá para crear el ejemplo.
3) Implementación: En esta fase, se procederá a implementar las funcionalidades definidas en la fase de análisis, siguiendo el diseño establecido en la fase anterior.
4) Test: Se harán test para comprobar que el ejemplo cumple con los requisitos deseados y no hay errores. Si algún problema de relevancia fuese encontrado durante la fase de implementación, será explicado aquí así como la mejor solución encontrada para solucionarlo.

5. Casos de estudio

5.1 Caso 1: Realidad Aumentada simple: Mostrando un modelo 3D

Investigación y análisis

En este caso de estudio se mostrara la manera más sencilla de obtener AR utilizando FLARToolKit. Solamente se mostrara un solo modelo simple en formato DAE (formato de COLLADA). Se utilizara un único marker con un patrón único para mostrar el modelo 3D. Para hacer el ejemplo algo más interesante, también se añadirá una interacción simple con el objeto por la cual el usuario será capaz de moverlo en las cuatro direcciones utilizando el teclado.

La principal razón de crear este caso de estudio es mostrar cómo obtener la forma más básica de AR. Este ejemplo es muy similar a las aplicaciones creadas por grandes compañías en sus campañas de marketing y uno de los objetivos es mostrar el bajo nivel de dificultad que requiere crearlos.

Diseño

El diseño seguido para crear este caso de estudio se basa en el proporcionado por Saqoosha en el Starter Kit de FLARToolKit.

A continuación se implementa el caso de estudio.

Implementación

Lo primero que se hace en este ejemplo es importar todas las librerías necesarias así como cambiar los parámetros de la cámara web (como la longitud, ancho, framerate y el color de fondo). Debajo, comienza la clase Sample1. Esta clase extiende de la clase PV3DARApp (véase el Apéndice 4 para su implementación), que es una clase ya creada incluida en el Starter Kit de FLARToolKit. Dentro de la clase Sample1, se crea la variable ‘amorfo’ de tipo DAE que contendrá el modelo 3D. En el método constructor de la clase, se añade un EventListener para el evento INIT que se lanza cuando

FLARToolKit acaba de cargar. El método 'init' es el que comienza FLARToolKit y tiene dos argumentos: La ruta de los parámetros de configuración de la cámara y la ruta del patrón a utilizar. Cuando el método 'init' termina, se lanzara el evento INIT que será capturado por el EventListener previo y ejecutara el método _onInit. En este método se crea el objeto DAE al cual se asignara el modelo 3D utilizando el método 'load'. Si el modelo lo requiere, se debe escalar y rotar el mismo para que se muestre correctamente en la pantalla. Finalmente se asigna el modelo al marker para que cada vez que este sea detectado, el modelo se mostrara.

Todos estos pasos eran necesarios para obtener la AR deseada para este caso de estudio. Se ha añadido una pequeña interacción con el modelo para mostrar cuán fácil es interaccionar con la AR.

Para usar el teclado con el fin de mover el modelo 3D, se ha añadido un EventListener al stage (a la escena) para detectar cuando se presiona una tecla del teclado. Esto ejecutara un método updateP que dependiendo que tecla ha sido presionada, el modelo 3D cambiara de posición en una dirección diferente.

```
package {

  //Import the necessary libraries
  import flash.events.Event;
  import flash.events.KeyboardEvent;
  import flash.ui.Keyboard;
  import org.papervision3d.objects.parsers.DAE;
  import com.transmote.utils.time.FramerateDisplay;

  //Webcam parameters
  [SWF(width="640", height="480", frameRate="30",
backgroundColor="#EEEEEE")]

  public class Sample1 extends PV3DARApp {

    private var amorfo:DAE;

    public function Sample1 () {
```

```
    addEventListener(Event.INIT, _onInit);
    init( 'Data/camera_para.dat', 'Data/patt001.pat');
  }

  private function _onInit(e:Event):void {
    mirror = true;
    amorfo = new DAE(true,"anima",true);
    amorfo.load('model/amorfo.DAE');
    amorfo.scale = 1;
    amorfo.rotationX = 90;
    _markerNode.addChild(amorfo);

    stage.addEventListener(KeyboardEvent.KEY_DOWN, updateP);
  }


  private function updateP(e:KeyboardEvent):void {
    switch( e.keyCode )
    {
      case Keyboard.SPACE:
        _update(e);
        break;

      case "W".charCodeAt():
      case Keyboard.UP:
        amorfo.moveRight(5);
        break;

      case "S".charCodeAt():
      case Keyboard.DOWN:
        amorfo.moveLeft(5);
        break;

      case "A".charCodeAt():
      case Keyboard.LEFT:
        amorfo.moveBackward(5);
        break;

      case "D".charCodeAt():
      case Keyboard.RIGHT:
        amorfo.moveForward(5);
```

```
                break;
            }
        }
    }
}
```

Problemas

Los problemas que se encontraron durante la realización de este ejemplo fueron:

- Problemas con las librerías. Algunas de las librerías no estaban actualizadas y eso generaba una excepción de tipo "ClassNotFound" al intentar utilizar una librería especifica.
- El segundo problema fue con la orientación del modelo 3D. Cuando se modela un objeto 3D con 3D Studio Max, se usan diferentes ejes de coordenadas y eso causaba que el modelo fuese mostrado con una orientación incorrecta. Para solucionar esto, se tiene que rotar el modelo 90º en el eje X.
- El tercer problema encontrado en este ejemplo fue que la imagen de la cámara estaba invertida. Esto provocaba que cuando el usuario movía el marker en una dirección, la imagen se movía en la dirección opuesta y eso es confuso para el usuario. Para solucionar esto, se tiene que modificar la propiedad 'mirror' de la clase PV3DARApp y cambiar su valor a 'true'.

5.2 Caso 2: Realidad Aumentada simple 2 – Mostrando video

Investigación y Análisis

En este caso de estudio se pretende explicar cómo mostrar objetos diferentes a modelos 3D en AR, específicamente un video. De nuevo, solamente se utilizara un marker y un único patrón que se utilizara para calcular la posición y orientación del video.

La razón por la cual se crea este caso de estudio es para mostrar que no solo se pueden utilizar modelos 3D para obtener AR, sino que también se pueden utilizar otros objetos como video o texto.

Una nueva característica utilizada en este caso de estudio (y que será utilizada en el resto) es la implementación de FLARManager.

Diseño

Este ejemplo esta creado siguiendo el diseño propuesto por Lee Brimelow en su blog "The Flash Blog". Utiliza FLARManager sobre FLARToolKit.

A continuación se describe la implementación del caso de estudio.

Implementación

De nuevo, lo primero que se hace es importar las librerías necesarias y definir los parámetros de la webcam. Una vez dentro de la clase videoTest, lo primero que se hace es definir la ruta donde se encuentran los FLARCameraParams. Esta ruta contiene todos los parámetros de la cámara que necesita FLARManager. Luego, se crean las variables necesarias que se utilizaran en este ejemplo. Dentro del constructor de la clase videoTest se inicializan los FLARParam (los parámetros del FLARManager) y los camParam (los parámetros de la cámara FLAR, que se encuentran en la ruta especificada anteriormente). Una vez que los parámetros se han cargado, se llama al método initFLAR. Este método crea un objeto nuevo FLARManager pasándole como argumentos la ruta del fichero de configuración de FLARManager. El flarConfig es un fichero XML que, entre otras cosas, contiene una lista de los patrones que se utilizaran en la aplicación. Esto dota a la aplicación de la posibilidad de usar más de un patrón permitiendo mostrar múltiples modelos en la misma aplicación (como se mostrara en el siguiente caso de estudio). Aparte del fichero de configuración, el método initFLAR añade unos EventsListeners para los eventos de añadir y remover el marker de la escena. Cuando estos eventos son lanzados, ejecutaran los métodos onAdded y onRemoved que realizan las acciones que han de suceder cuando un marker se añadido o removido de la escena respectivamente. Después de crear el objeto FLARManager con el fichero de configuración y añadir los EventListeners, se añade el objeto FLARManager a la aplicación utilizando la función addChild. Finalmente se añade el

event listener para el evento INIT que es lanzado cuando FLARManager termina de cargar. Esto ejecutara la función init3D.

La función init3D es donde, entre otras cosas, el motor de render 3D se inicializa. En ella se crea una scene3D, una cámara (esta cámara necesita ser del tipo FLARCamera3D), un Viewport3D y se crea el motor de render utilizando como argumentos las variables creadas recientemente. Porque se quiere mostrar un video, se crea un material (de tipo MovieMaterial) y con él se crea el plano donde la película se mostrara. Finalmente se crea un DisplayObject3D y le añadimos el plano. Se añade el DisplayObject a la escena y se termina añadiendo un EventListener para capturar el evento de ENTER_FRAME que ejecutara el método 'loop'. El método loop es el encargado de actualizar la posición del video dependiendo del marker. Si el marker está en escena, transformara el DisplayObject3D usando la trasformMatrix del marker para actualizar la nueva posición. Finalmente se tiene que renderizar la nueva escena usando el motor de render para mostrar la posición actualizada del video.

```
package
{
  import com.transmote.flar.FLARManager;
  import com.transmote.flar.camera.FLARCamera_PV3D;
  import com.transmote.flar.marker.FLARMarker;
  import com.transmote.flar.marker.FLARMarkerEvent;
  import com.transmote.flar.tracker.FLARToolkitManager;
  import com.transmote.flar.utils.geom.PVGeomUtils;
  import com.transmote.utils.time.FramerateDisplay;

  import flash.display.Sprite;
  import flash.events.Event;
  import flash.geom.Rectangle;

  import org.libspark.flartoolkit.core.param.FLARParam;
  import org.libspark.flartoolkit.support.pv3d.FLARCamera3D;
  import org.papervision3d.cameras.Camera3D;
  import org.papervision3d.materials.MovieMaterial;
  import org.papervision3d.objects.DisplayObject3D;
  import org.papervision3d.objects.primitives.Plane;
  import org.papervision3d.render.LazyRenderEngine;
```

```
  import org.papervision3d.scenes.Scene3D;
  import org.papervision3d.view.Viewport3D;

  [SWF(width="640", height="480", frameRate="60",
backgroundColor="#FFFFFF")]

  public class videoTest extends Sprite
  {
    [Embed(source="../resources/flarToolkit/FLARCameraParams.dat",
mimeType="application/octet-stream")]
    private var camParam :Class;
    private var parameter:FLARParam;
    private var fm:FLARManager;
    private var scene:Scene3D;
    private var view:Viewport3D;
    private var camera:FLARCamera3D;
    private var lre:LazyRenderEngine;
    private var p:Plane;
    private var con:DisplayObject3D;
    private var marker:FLARMarker;
    private var v:Vid;

    public function videoTest()
    {
      parameter = new FLARParam();
      parameter.loadARParam(new camParam());

      initFLAR();
      v = new Vid();
      v.vid.source = "kramer.m4v";
      v.vid.stop();
    }

    private function initFLAR():void
    {
      fm = new FLARManager("../resources/flar/flarConfig.xml",new
FLARToolkitManager(),this.stage);
      fm.addEventListener(FLARMarkerEvent.MARKER_ADDED, onAdded);
      fm.addEventListener(FLARMarkerEvent.MARKER_REMOVED, onRemoved);
      addChild(Sprite(fm.flarSource));
```

```
    fm.addEventListener(Event.INIT, init3D);
}

private function onAdded(e:FLARMarkerEvent):void
{
    marker = e.marker;
    p.visible = true;
    v.vid.play();
}

private function onRemoved(e:FLARMarkerEvent):void
{
    marker = null;
    p.visible = false;
    v.vid.stop();
}

private function init3D(e:Event):void
{
    scene = new Scene3D();
    camera = new FLARCamera3D(parameter);
    camera.z = -30;
    view = new Viewport3D(640, 480, true);
    lre = new LazyRenderEngine(scene, camera, view);

    var mat:MovieMaterial = new MovieMaterial( v, false, true);
    p = new Plane(mat, 320, 240, 2, 2);
    p.scaleY = -1;
    p.rotationZ = -90;
    p.visible = false;

    con = new DisplayObject3D();
    con.addChild(p);

    scene.addChild(con);
    addChild(view);
    addChild(new FramerateDisplay());

    addEventListener(Event.ENTER_FRAME, loop);
}
```

```
    private function loop(e:Event):void
    {
      if(marker != null)
      {
        con.transform =
PVGeomUtils.convertMatrixToPVMatrix(marker.transformMatrix);
      }
      lre.render();
    }
  }
}
```

Problemas

Los principales problemas en este caso de estudio fueron dos. El primero fue encontrar un contenedor capaz de contener video. Esto se soluciono usando un material de video y aplicándolo a un plano. El plano puede ser luego utilizado como un objeto normal al añadirlo a un DisplayObject3D. El segundo problema fue la cámara FLARManager usa una FLARCamera3D como cámara. Esta cámara debe ser inicializada con los FLARParams después de que FLARManager se cargue y no antes o durante. Finalmente, se encontraron problemas menores con versiones de librerías. Este es un problema general encontrado muchas veces en distintos proyectos cuando se usa una librería o tecnología nueva. La mejor manera de solucionar este problema (o al menos reducirlo) es usando un controlador de versiones o SVN.

5.3 Caso 3: Múltiples markers y modelos usando FlarManager

Investigación y Análisis

En este caso de estudio se pretende explicar cómo utilizar FLARManager para ser capaz de obtener múltiples markers y modelos en una misma aplicación. Este será el diseño básico que se seguirá para futuros ejemplos. La razón por la cual se decidió utilizar FLARManager es que permite la utilización de múltiples markers de una manera sencilla y provee mejores posibilidades de interactuar con los markers, modelos o patrones. Para ello, utiliza un fichero de configuración para gestionar los diferentes patrones a utilizar en la aplicación, así como también permite definir valores como el

tamaño de los markers o el valor de umbral (threshold value) que se aplica. (Véase un ejemplo en el Apéndice 6).

Diseño

Este caso de estudio sigue el diseño estándar de aplicaciones basadas en FLARManager como se muestra en la documentación de FLARManager.

A continuación se encuentra la implementación comentada de este caso de estudio.

Implementación

```
package {
  /* Tweener Class [http://code.google.com/p/tweener/] */
  import alternativa.engine3d.primitives.Plane;

  import caurina.transitions.Tweener;

  import com.transmote.flar.FLARManager;
  import com.transmote.flar.marker.FLARMarker;
  import com.transmote.flar.marker.FLARMarkerEvent;
  import com.transmote.flar.utils.geom.FLARPVGeomUtils;

  import flash.display.Graphics;
  import flash.display.Shape;
  import flash.display.Sprite;
  import flash.events.Event;
  import flash.events.MouseEvent;
  import flash.filters.GlowFilter;
  import flash.geom.Point;
  import flash.utils.Dictionary;

  import org.libspark.flartoolkit.support.pv3d.FLARCamera3D;
  import org.papervision3d.lights.PointLight3D;
  import org.papervision3d.materials.shadematerials.FlatShadeMaterial;
  import org.papervision3d.materials.utils.MaterialsList;
  import org.papervision3d.objects.DisplayObject3D;
  import org.papervision3d.objects.parsers.DAE;
```

```
  import org.papervision3d.objects.primitives.Cube;
  import org.papervision3d.objects.special.Graphics3D;
  import org.papervision3d.render.LazyRenderEngine;
  import org.papervision3d.scenes.Scene3D;
  import org.papervision3d.view.Viewport3D;


  /* Change output settings */
  [SWF(width="640", height="480", frameRate="25",
backgroundColor="#000000")]

  public class Multiple_Markers extends Sprite {
    /* FLARManager pointer */
    private var fm:FLARManager;
    /* Papervision Scene3D pointer */
    private var scene3D:Scene3D;
    /* Papervision Viewport3D pointer */
    private var viewport3D:Viewport3D;
    /* FLARToolkit FLARCamera3D pointer */
    private var camera3D:FLARCamera3D;
    /* Papervision render engine pointer */
    private var lre:LazyRenderEngine;
    /* Papervision PointLight3D pointer */
    private var pointLight3D:PointLight3D;

    /* Initialise glow filter to add a white border around selected
objects in our scene */
    private var glow:GlowFilter = new GlowFilter(0xFFFFFF, 1, 7, 7, 30,
1, false, false);

    /* Vector storing references to all markers on screen, grouped by
pattern id */
    private var markersByPatternId:Vector.<Vector.<FLARMarker>>;

    /* Dictionary storing references to marker containers, indexed by
relevant marker object */
    private var containersByMarker:Dictionary;

    /* Dictionary storing references to the corner nodes for each
marker, indexed by relevant marker object */
    private var nodesByMarker:Dictionary;
```

```
/* Constructor method */
public function Multiple_Markers() {
  /* Run augmented reality initialisation */
  this.initFLAR();
}

/* Augmented reality initialisation */
private function initFLAR():void {

  /* Initialise FLARManager */
  this.fm = new FLARManager("flarConfig.xml");

  /* Temporary declaration of how many patterns are being used */
  var numPatterns:int = 12;

  /* Initialise markerByPatternId vector object */
  this.markersByPatternId = new
  Vector.<Vector.<FLARMarker>>(numPatterns, true);

  /* Loop through each pattern */
  while (numPatterns--) {

    /* Add empty Vector to each pattern */
    this.markersByPatternId[numPatterns] = new
    Vector.<FLARMarker>();
  }

  /* Initialise empty containersByMarker dictionary object */
  this.containersByMarker = new Dictionary(true); //weakKeys = true

  /* Initialise empty nodesByMarker dictionary object */
  this.nodesByMarker = new Dictionary(true);

  /* Event listener for when a new marker is recognised */
  fm.addEventListener(FLARMarkerEvent.MARKER_ADDED, this.onAdded);
  /* Event listener for when a marker is removed */
  fm.addEventListener(FLARMarkerEvent.MARKER_REMOVED,
  this.onRemoved);
  /* Event listener for when the FLARManager object has loaded */
  fm.addEventListener(Event.INIT, this.onFlarManagerLoad);
```

```
      /* Display webcam */
      this.addChild(Sprite(fm.flarSource));
    }

    /* Run if FLARManager object has loaded */
    private function onFlarManagerLoad(e:Event):void {
      /* Remove event listener so this method doesn't run again */
      this.fm.removeEventListener(Event.INIT, this.onFlarManagerLoad);
      /* Run Papervision initialisation method */
      this.initPaperVision();
    }

    /* Run when a new marker is recognised */
    private function onAdded(e:FLARMarkerEvent):void {
      /* Run method to add a new marker */
      this.addMarker(e.marker);
    }
    /* Run when a marker is removed */
    private function onRemoved(e:FLARMarkerEvent):void {
      /* Run method to remove a marker */
      this.removeMarker(e.marker);
    }

    /* Add a new marker to the system */
    private function addMarker(marker:FLARMarker):void {
      /* Store reference to list of existing markers with same pattern
id */
      var markerList:Vector.<FLARMarker> =
this.markersByPatternId[marker.patternId];
      /* Add new marker to the list */
      markerList.push(marker);

      /* Initialise the marker container object */
      var container:DisplayObject3D = new DisplayObject3D();
      if (marker.patternId == 0){
        // load the model.
        // (this model has to be scaled and rotated to fit the marker;
every model is different.)
        var model:DAE = new DAE(true, "model", true);
        model.load("scout.dae");
```

```
            model.rotationX = 90;
            model.rotationZ = 90;
            model.scale = 0.5;
            model.name = "c";

            // Add the model to the container.
            container.addChild(model);
            container.visible = true;
            // Add the container to the scene3D.
            scene3D.addChild(container);

        }else{
            /* Prepare material to be used by the Papervision cube based on
pattern id */
        var flatShaderMat:FlatShadeMaterial = new
FlatShadeMaterial(pointLight3D, getColorByPatternId(marker.patternId),
getColorByPatternId(marker.patternId, true));
        /* Add material to all sides of the cube */
        var cubeMaterials:MaterialsList = new MaterialsList({all:
flatShaderMat});

        /* Initialise the cube with material and set dimensions of all
sides to 40 */
        var cube:Cube = new Cube(cubeMaterials, 40, 40, 40);

        /* Shift cube upwards so it sits on top of paper instead of being
cut half-way */
        cube.z = 0.5 * 40;

        /* Scale cube to 0 so it's invisible */
        cube.scale = 0;

        /* Add animation which scales cube to full size */
        Tweener.addTween(cube, {scale: 1, time:0.5,
transition:"easeInOutExpo"});

        /* Set cube to be individually affected by filters */
        cube.useOwnContainer = true;
        /* Add cellshaded border using glow filter */
        cube.filters = [this.glow];
```

```
      cube.name = "c";
      /* Add finished cube object to marker container */
      container.addChild(cube);
      /* Add marker container to the Papervision scene */
      this.scene3D.addChild(container);

      }

      /* Add marker container to containersByMarker Dictionary object
*/
      this.containersByMarker[marker] = container;
    }

    /* Remove a marker from the system */
    private function removeMarker(marker:FLARMarker):void {
      /* Store reference to list of existing markers with same pattern
id */
      var markerList:Vector.<FLARMarker> =
this.markersByPatternId[marker.patternId];
      /* Find index value of marker to be removed */
      var markerIndex:uint = markerList.indexOf(marker);
      /* If marker exists in markerList */
      if (markerIndex != -1) {
        /* Remove marker from markersByPatternId */
        markerList.splice(markerIndex, 1);
      }

      /* Store reference to marker container from containersByMarker
Dictionary object */
      var container:DisplayObject3D = this.containersByMarker[marker];
      /* If a container exists */
      if (container) {
        /* Remove container from the Papervision scene */
        this.scene3D.removeChild(container);
      }
      /* Remove container reference from containersByMarker Dictionary
object */
      delete this.containersByMarker[marker];

      /* Clear any corner nodes for this marker from the display */
      this.nodesByMarker[marker].graphics.clear();
```

```
      /* Remove reference to corner nodes for this marker from
nodesByMarker Dictionary object */
      delete this.nodesByMarker[marker];
    }

    /* Papervision initialisation method */
    private function initPaperVision():void {
      /* Initialise a new Papervision scene */
      this.scene3D = new Scene3D();
      /* Initialise a new FLARCamera3D object to enable full AR
goodness */
      this.camera3D = new FLARCamera3D(this.fm.cameraParams);

      /* Define a new Papervision viewport object */
      this.viewport3D = new Viewport3D(640, 480, true);
      /* Add viewport to the main scene */
      this.addChild(this.viewport3D);

      /* Define a new Papervision point light */
      this.pointLight3D = new PointLight3D(true, false);
      /* Set light position */
      this.pointLight3D.x = 1000;
      this.pointLight3D.y = 1000;
      this.pointLight3D.z = -1000;
      /* Add light to the Papervision scene */
      this.scene3D.addChild(pointLight3D);

      /* Initialise the Papervision render engine */
      this.lre = new LazyRenderEngine(this.scene3D, this.camera3D,
this.viewport3D);

      /* Create event listner to run a method on each frame */
      this.addEventListener(Event.ENTER_FRAME, this.onEnterFrame);
    }

    /* Method to run on each frame */
    private function onEnterFrame(e:Event):void {
      /* Loop through corner nodes */
      for (var marker:Object in nodesByMarker) {
        /* Clear any corner nodes for this marker from the display */
        nodesByMarker[marker].graphics.clear();
```

```
      }

      /* Run method to update markers */
      this.updateMarkers();

      /* Render the Papervision scene */
      this.lre.render();
    }

    /* Update markers method */
    private function updateMarkers():void {
      /* Store reference to amount of patterns being tracked */
      var i:int = this.markersByPatternId.length;
      /* Store reference to list of existing markers */
      var markerList:Vector.<FLARMarker>;
      /* Empty marker variable */
      var marker:FLARMarker;
      /* Empty container variable */
      var container:DisplayObject3D;
      /* Empty integer */
      var j:int;

      /* Loop through all tracked patterns */
      while (i--) {
        /* Store reference to all markers with this pattern id */
        markerList = this.markersByPatternId[i];
        /* Amount of markers with this pattern */
        j = markerList.length;
        /* Loop through markers with this pattern */
        while (j--) {
          /* Store reference to current marker */
          marker = markerList[j];

          /* Initialise a new Shape object */
          var nodes:Shape = new Shape();
          /* Define line style for corner nodes */
          nodes.graphics.lineStyle(3,
getColorByPatternId(marker.patternId));
          /* Store reference to coordinates for each corner of the
marker */
          var corners:Vector.<Point> = marker.corners;
```

```
            /* Empty coordinate variable */
            var vertex:Point;
            /* Loop through rest of corner coordinates */
            for (var c:uint=0; c<corners.length; c++) {
              /* Store reference to current corner coordinates */
              vertex = corners[c];
              /* Draw a 2D circle at these coordinates */
              nodes.graphics.drawCircle(vertex.x, vertex.y, 5);
            }
            /* Add reference to corner nodes to nodesByMarker Dictionary
object */
            this.nodesByMarker[marker] = nodes;
            /* Add corner nodes to the main scene */
            this.addChild(nodes);

            /* Find reference to marker container in containersByMarker
Dictionary object */
            container = this.containersByMarker[marker];
            /* Transform container to new position in 3d space */
            container.transform =
FLARPVGeomUtils.convertFLARMatrixToPVMatrix(marker.transformMatrix);
          }
        }
      }

      /* Get colour values dependent on pattern id */
      private function getColorByPatternId(patternId:int, shaded:Boolean
= false):Number {
        switch (patternId) {
          case 0:
            if (!shaded)
              return 0xFF1919;
            return 0x730000;
          case 1:
            if (!shaded)
              return 0xFF19E8;
            return 0x730067;
          case 2:
            if (!shaded)
              return 0x9E19FF;
            return 0x420073;
```

```
        case 3:
          if (!shaded)
            return 0x192EFF;
          return 0x000A73;
        case 4:
          if (!shaded)
            return 0x1996FF;
          return 0x003E73;
        case 5:
          if (!shaded)
            return 0x19FDFF;
          return 0x007273;
        default:
          if (!shaded)
            return 0xCCCCCC;
          return 0x666666;
      }
    }
  }
}
```

Problemas

El principal problema cuando se trabaja con múltiples markers es como identificarlos. FLARManager utiliza un patternId para identificar entre los diferentes patrones. Este patternId se define en el fichero de configuración. Otro problema es el rendimiento. Si envés de utilizar cubos 3D simples, se utilizan modelos 3D complejos y se muestras varios o todos a la vez, el rendimiento de la aplicación caerá considerablemente.

5.4 Caso 4: Interacción utilizando un controlador externo (WiiMote)

Investigación y Análisis

En este caso de estudio se explicara como interactuar con los objetos de la escena usando un controlador externo como puede ser el mando de la Wii, o WiiMote. El objetivo es mostrar las posibilidades que provee la AR en una forma básica que luego puede ser expandida para desarrollos futuros.

La razón por la cual se eligió realizar este caso de estudio fue para mostrar el siguiente paso lógico en el desarrollo de la AR, la interacción. Con la habilidad de poder interaccionar con la AR, el usuario puede sacar mucho más provecho de las aplicaciones en el futuro, envés de simplemente ver imágenes aumentadas en la pantalla.

Diseño

Como en el caso de estudio anterior, se seguirá el diseño estándar de aplicaciones basadas en FLARManager como se muestra en la documentación del mismo.

A continuación se encuentra la implementación del mismo.

Implementación

La implementación del caso de estudio se basa en el modelo presentado en el caso 3 y se expande para conseguir interactuar, por medio del controlador de la Wii (WiiMote), con los objetos de la aplicación. La implementación completa puede encontrarse en el Apéndice 7.

Lo primero que se hace es añadir las librerías necesarias que no se encontraban previamente en el proyecto:

```
import org.wiiflash.Wiimote;
import org.wiiflash.events.ButtonEvent;
import org.wiiflash.events.WiimoteEvent;
import flash.events.*;
```

Los imports de 'wiiflash' son las librerías necesarias para conectar la aplicación con el Wiimote. Se utiliza una aplicación externa que debe estar en ejecución llamada WiiFlash Server.

Una vez dentro de la clase principal, se añaden dos variables:

```
private var myWiimote:Wiimote = new Wiimote();

private var selectedObj:int = -1;
```

La primera es el objeto que se usara para interactuar con el WiiMote y el WiiFlash Server. La segunda variable se utiliza para guardar el patternId del objeto seleccionado cuando se hace click sobre él.

Dentro del método initFLAR se añadirá en el comienzo la siguiente línea:

```
myWiimote.connect ();
```

Esta línea conectara el WiiMote con el WiiFlash Server (este último debe estar ejecutándose con anterioridad).

En el final del método initFLAR se añadirá un EventListener para capturar cuando el usuario hace click sobre un objeto utilizando el mouse o el WiiMote:

```
stage.addEventListener(MouseEvent.CLICK, selectObj);
```

Se añade al proyecto el método selectObj:

```
private function selectObj(e:MouseEvent):void {

  myWiimote.mouseControl = true;

  var numPatterns:int = markersByPatternId.length;

  while (numPatterns--){

    var markerList:Vector.<FLARMarker> =
    this.markersByPatternId[numPatterns];

    if (markerList.length > 0){
      var marker:FLARMarker = markerList[0];

      var corners:Vector.<Point> = marker.corners;
```

```
      var p:Point = new Point(e.stageX, e.stageY);

      if (insidePolygon(corners, p)){

        if (selectedObj != -1){
          var auxMarkerList:Vector.<FLARMarker> =
          this.markersByPatternId[selectedObj];
          var auxContainer:DisplayObject3D =
          this.containersByMarker[auxMarkerList[0]];
          var auxChild:DisplayObject3D =
          auxContainer.removeChildByName("c");
          if (auxChild != null){
            auxChild.scale = 1;
            auxContainer.addChild(auxChild);
          }
        }
        selectedObj = marker.patternId;
        var container:DisplayObject3D =
        this.containersByMarker[marker];
        var child:DisplayObject3D = container.removeChildByName("c");
        if (child != null){
          child.scale = 5;
          container.addChild(child);
        }
      }
    }
  }
}
```

La función de este método es identificar sobre que objeto ha hecho click el usuario. Para ello, utiliza las coordenadas donde se ha hecho click y utilizando el método llamado insidePolygon comprueba si se ha hecho click sobre un marker. El método insidePolygon lo que hace es comprobar si el punto que se le pasa como parámetros, están dentro del área de alguno de los markers y en tal caso, devolvería un true. Si el usuario ha hecho click sobre un marker (u objeto), una acción se realiza en el objeto correspondiente.

Se añade el método insidePolygon a nuestro proyecto:

```
/* Private function to know if a Point is inside a polygon */
private function insidePolygon(pointList:Vector.<Point>,
p:Point):Boolean
{
  var counter:int = 0;
  var i:int;
  var xinters:Number;
  var p1:Point;
  var p2:Point;
  var n:int = pointList.length;

  p1 = pointList[0];
  for (i = 1; i <= n; i++)
  {
    p2 = pointList[i % n];
    if (p.y > Math.min(p1.y, p2.y))
    {
      if (p.y <= Math.max(p1.y, p2.y))
      {
        if (p.x <= Math.max(p1.x, p2.x))
        {
          if (p1.y != p2.y) {
            xinters = (p.y - p1.y) * (p2.x - p1.x) / (p2.y - p1.y) +
            p1.x;
            if (p1.x == p2.x || p.x <= xinters)
              counter++;
          }
        }
      }
    }
    p1 = p2;
  }
  if (counter % 2 == 0)
  {
    return(false);
  }
  else
  {
    return(true);
  }
}
```

Este método comprobara si el punto dado esta dentro de un polígono y retornara true en caso afirmativo o false en caso contrario.

Problemas

Se encontraron varios problemas a la hora de realizar este caso de estudio:

- Para conectar el WiiMote con el ordenador usando WiiFlash Server se tuvo que utilizar un Bluetooth USB.
- Como saber si el usuario ha hecho click sobre un objeto o no.
- Se encontró un problema con la función mouseControl del objeto wiimote. No funcionaba correctamente. Esto se debía a que para usar dicha función, es necesario utilizar un dispositivo de Infrarrojo.

6. Evaluación del Producto

El producto presentado en este proyecto, consiste en cuatro casos de estudio. Estos casos no son aplicaciones grandes y complejas que resuelven requerimientos específicos. Cada uno de ellos es un ejemplo pequeño y suficientemente simple que muestra las diferentes posibilidades de la AR. Cada uno de estos casos tiene el objetivo de ser presentado y explicado en una manera tal que, cualquiera interesado en la AR, pueda utilizarlos como punto de partida para crear una aplicación de AR más grande y compleja partiendo de una base sólida.

Porque han sido desarrollados utilizando FLARToolKit basado en Flash, pueden ser ejecutados en casi cualquier navegador bajo cualquier Sistema Operativo. Además, pueden ser modificados sencillamente utilizando cualquier editor de texto o Action Script SDK tal como Flash Builder.

7. El futuro de la Realidad Aumentada

La Realidad Aumentada se ha ido introduciendo gradualmente en el mercado, pero aún tiene un largo camino por delante. Todo depende en cuanto provecho podamos sacarle y que la tecnología continúe evolucionando a su favor.

Una de las grandes preguntas que se pueden hacer cuando se piensa en el futuro de la Realidad Aumentada podría ser: ¿Tendrá el mismo futuro que la Realidad Virtual Inmersiva la cual está casi olvidada? Esta cuestión nace porque la AR es vista como una evolución de la Realidad Virtual.

Una de las principales razones por la cual la Realidad Virtual Inmersiva fallo, fue su alto coste tanto en hardware como en software, puesto que no era una tecnología lo suficientemente portable para llevar a cualquier parte. Tan solo grandes organizaciones, como la militar, podría costearse esta tecnología donde el riesgo sin un entorno virtual donde practicar era demasiado elevado.
Es debido a la superación de estas dificultades por las cuales la AR se ha expandido tanto y es seguida por tanta gente.

La Realidad Aumentada ya se ha implementado con relativo éxito en Internet y Smartphone. Esto dota a la AR con un gran mercado permitiendo a casi cualquier usuario, hacer uso de ella en cualquier lugar con la única necesidad de una cámara.

Uno de los campos más interesantes a día de hoy donde la Realidad Aumentada ha sido implementada con éxito, es el campo de marketing. Grandes compañías como Adidas, Coca-Cola o Axe han probado de utilizar la AR en sus campañas de marketing presentándola como una tecnología revolucionaría y con resultados impresionantes.
Por ejemplo, Adidas añadió un pequeño marker a uno de sus zapatillas la cual, si se ponía delante de una cámara usando su aplicación, mostraba un original vecindario que salía directamente de nuestras zapatillas.

Figura 15 - http://www.adidas.com/originals/us/neighborhood

En otra campaña, esta vez con respecto a la Copa Mundial de Football, la compañía Coca-Cola añadió unos pequeños markers a sus botellas. Dependiendo del país, cuando se mostraba el marker a una cámara, una animación que mostraba una pequeña hada vestida con los colores de la selección aparecía dando soporte a su equipo.

Por último, y el caso más reciente, la marca de desodorantes Axe (Lynx en Inglaterra) utilizo la AR de una forma muy novedosa. Siguiendo la trama de sus anuncios publicitarios donde caían ángeles del cielo, instalaron un gran marker en el suelo de la estación de trenes de Londres.

Figura 16 – Demostración AR de AXE en la estación de Londres

Cuando la gente se paraba sobre él, utilizaban una pantalla gigante para hacer aparecer, por medio de AR, ángeles que caían a su lado e interactuaban con ellos. Esta demostración tuvo un gran impacto en la gente.

La Realidad Aumentada Markerless (sin la utilización de markers físicos) ofrece un futuro muy interesante y más cuando se combina con gafas comerciales de AR donde objetos virtuales pueden ser superpuestos a la visión humana.

BMW está trabajando, por ejemplo, en una aplicación de AR que permitirá a sus mecánicos o gente sin experiencia, a solucionar problemas simples en sus vehículos mostrando al operador de forma visual, interactiva y paso por paso como resolver cierto tipo de averías. (Figure 16)

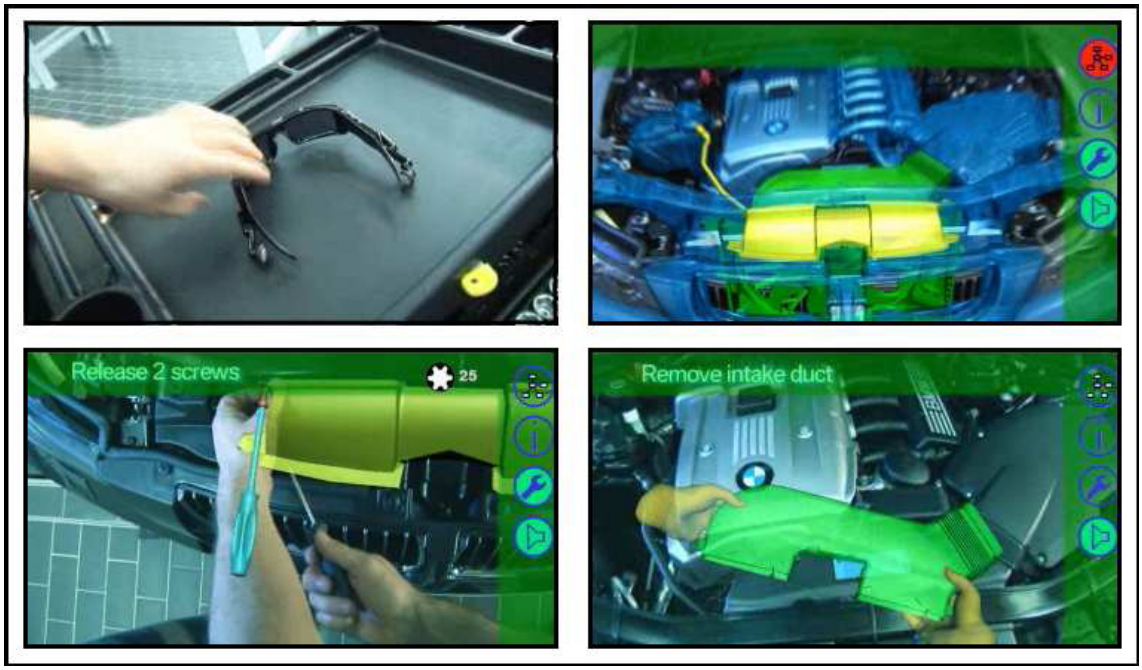

Figura 17 – Proyecto de Realidad Aumentada de BMW para asistencia mecánica

Un campo donde la AR puede influenciar mucho es la Educación. La Realidad Aumentada intentaría construir métodos educativos donde el estudiante tomase el control de su propio aprendizaje interactuando con un entorno real y virtual. En situaciones de aprendizaje parcialmente virtuales como la AR, el estudiante podría manipular objetos virtuales y aprender tareas o habilidades. El beneficio con el aprendizaje mediante la AR es que no hay errores "reales". Por ejemplo, si un bombero aprende a combatir diferentes tipos de fuegos, o un cirujano aprende una laparoscopía en una situación de AR, no hay consecuencias reales si cometen un fallo durante el entrenamiento. Este tipo de entrenamiento provee oportunidades para un aprendizaje más real y promueve diferentes estilos de aprendizaje. Aplicaciones de Realidad Aumentada que pueden "aumentar" libros de texto, tienen la capacidad de involucrar al lector de maneras que antes no eran posibles. Un viaje a un museo o lugar histórico con un grupo de estudios usando aplicaciones de AR, podrían proveer a cada estudiante con una experiencia única de aprendizaje.

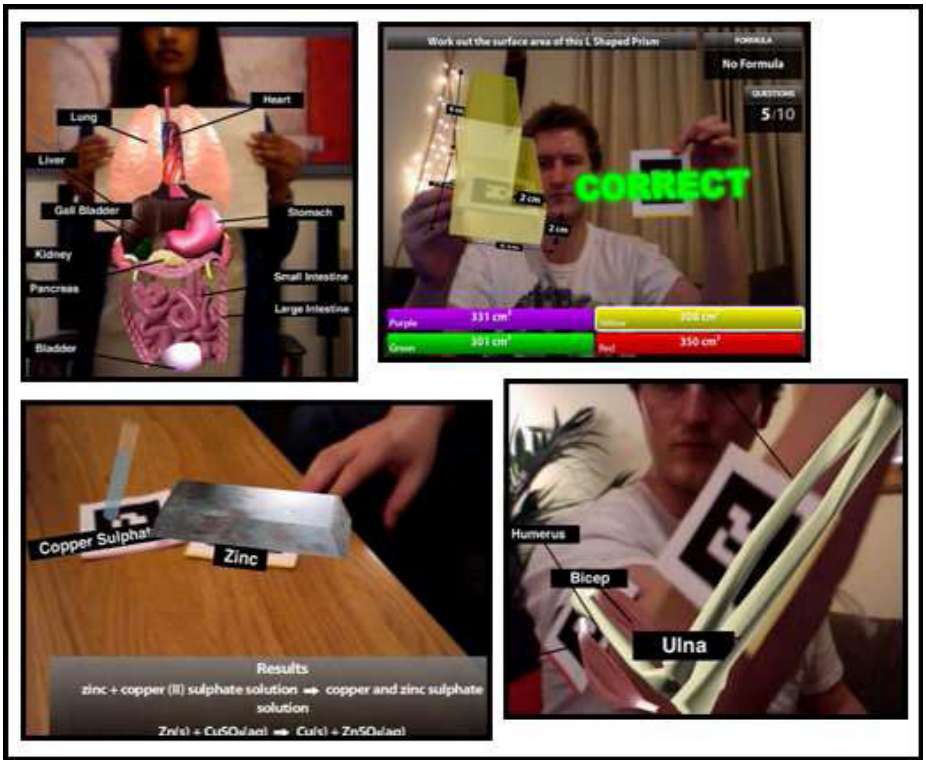

Figura 18 – El proyecto independiente LearnAR, pretende crear pequeños aplicativos de AR con propósitos educativos. En el futuro, esto podría ser más realista y markerless.

Otro campo en donde la AR ha tenido y tendrá un gran impacto es en el campo del entretenimiento. Ya existen varios juegos interactivos que utilizan la AR (Figura 18) e incluso si bien están limitados a usar markers, tienen un buen future. Especialmente en el campo de los juegos móviles, la AR a tenido un gran impacto al sacar provecho de las posibilidades que ofrecen los dispositivos móviles tal como la portabilidad, el tener una cámara integrada, localización GPS, etc.

Figura 19 – El juego de PSP "Invizimals" usa un marker y la cámara de la PSP para generar Realidad Aumentada.

Como se puede ver, los campos en los cuales la Realidad Aumentada puede ser aplicada son varios. En el futuro, cuando los avances tecnológicos necesarios se alcancen, tales como unas gafas de AR que permitan superponer información virtual a lo que vemos, la AR podría formar parte de nuestra vida cotidiana como lo es hoy la televisión o hablar por el móvil.

8. Revisión crítica

En el comienzo, tenía la idea de desarrollar una aplicación grande y compleja hecha con Realidad Aumentada. Estaba pensando en hacer un libro de Realidad Aumentada donde el texto estuviese enriquecido con imágenes y animaciones virtuales que mostrasen lo que estaba pasando. Otra idea era la de crear un juego para móviles en Realidad Aumentada donde alguien pudiese descargárselo por internet y jugar con su Smartphone en cualquier parte. La razón por la cual cambie de parecer fue que el propósito de este proyecto era el de mostrar que era posible hacer con la AR y mostrar como hacerlo, envés de crear una sola aplicación compleja que nadie sin conocimientos en la materia pudiese comprender o utilizar. En cambio, decidí crear pequeños ejemplos simples empezando por la forma más básica de AR que es mostrar un modelo 3D, evolucionando a cosas más complejas tales como utilizar múltiples markers en una misma aplicación e interactuar con ellos por medio de un controlador externo.

Este es uno de los puntos que, cuando tengo que hacer una revisión crítica de mi trabajo, debo aclarar en mayor profundidad. Mirando atrás, si ahora alguien quisiese crear un Libro de Realidad Aumentada, solo tendría que usar uno de los dos primeros ejemplos (o una combinación de ellos), crear unos markers con sus propios patrones y añadirlos al libro. Con un poco de creatividad e imaginación, alguien podría conseguir algo tan bueno como “The MagicBook: a transitional AR interface”. The MagicBook es una interfaz de Mixed Reality (Realidad Mixta) que usa un libro real para transportar de manera similar a los usuarios entre la Realidad y la Virtualidad. Un sistema de seguimiento basado en la visión es usado para superponer modelos virtuales en las páginas reales de un libro, creando una escena de Realidad Aumentada. Cuando los usuarios ven una escena de AR están interesados en ella y pueden introducirse en ella y experimentarla. La interfaz también soporta colaboraciones escalables, permitiendo a múltiples usuarios experimentar la el mismo entorno virtual ya sea desde una perspectiva egocéntrica o exocéntrica.

Si alguien quisiera realizar una aplicación de AR más compleja, podría usar FLARManager con múltiples markers e interacción entre ellos para crear una aplicación de AR interactiva.

De cualquier manera, los casos de estudio contienen las bases para desarrollar esto y mucho más, u ese era uno de los principales objetivos de este proyecto.

Una de las cosas que creo que mejoraría este proyecto sería incluir un quinto caso de estudio mostrando la implementación de una aplicación que aplicase leyes físicas a los modelos. Creo que esta es una gran característica de la AR y dota al usuario de una experiencia incluso mejor al proporcionar al modelo con la capacidad innata de reaccionar contra otros modelos o fuerzas tales como la gravedad.

9. Conclusiones

La Realidad Aumentada usa sistemas de visión artificial y otras tecnologías para solucionar un importante problema: El seguimiento del punto de vista o Viewpoint tracking. Estas técnicas provienen principalmente de los progresos hechos en las investigaciones llevadas a cabo sobre la Realidad Virtual, la cual es la base para la Realidad Aumentada.

Existen muchas herramientas que permiten el desarrollo de aplicaciones en Realidad Aumentada. La mayoría de ellas aún siguen en fase de desarrollo y aún tienen mucho camino por delante. Mientras que las herramientas comerciales ofrecen una amplia oferta de posibilidades, las herramientas gratuitas les siguen los pasos y un claro ejemplo es la librería de ARToolKit que es muy poderosa en términos de Realidad Aumentada usando Markers.

La Realidad Aumentada intenta aumentar la percepción de una persona combinando objetos virtuales con el mundo real todo en la misma escena. Básicamente, la Realidad Aumentada usa el modelos de visión artificial clásico, añadiendo al modelo un feedback continuo acerca de la posición y punto de vista del usuario para superponer información virtual a la imagen capturada.

Podemos identificar tres modelos diferentes de Realidad Aumentada. El modelo Markerless es el que presenta mayor futuro porque se integra fácilmente con el entorno y no tiene restricciones en sus posibilidades. Desafortunadamente, es el que menos desarrollado se encuentra de los tres y la principal razón es que la tecnología necesaria para utilizarlo aún está en fase de desarrollo. El modelo usado en Smartphone con GPS, debido a la portabilidad de los dispositivos, representa un gran sector de mercado y ha empezado a expandirse. Aunque los GPS son sistemas inexactos y tienen ciertas limitaciones, este modelo presenta un alto grado de utilidad.

Finalmente, el modelo que utiliza markers físicos es uno de los más desarrollados hoy en día, pero por el contrario es el más limitado. Aunque este modelo puede ser implementado usando hardware y software barato y sencillo, lo cual ha ayudado a su expansión masiva, esta clase de Realidad Aumentada requiere de patrones previamente definidos para reconocerlos y esta es su mayor limitación. Pese a ello, esta clase de Realidad Aumentada se ha expandido mucho en los últimos años en campos tales como marketing o videojuegos, teniendo un gran impacto en ellos. Este es el motivo por el cual se escogió ARToolKit para ser analizado.

De las diversas variantes de ARToolKit, FLARToolKit fue seleccionado porque se ha hecho muy popular debido a la facilidad de implementación en los navegadores web ya que utiliza la tecnología Flash. Utilizando esta herramienta, un conjunto de casos de estudio se desarrollo con el objetivo de mostrar cómo crear aplicaciones de Realidad Aumentada empezando con modelos básicos, hasta usar herramientas de control como FLARManager para gestionar múltiples markers en la misma aplicación e interactuar con ellos.

Para terminar, se realizo un rápido repaso del futuro de la Realidad Aumentada. Aunque las limitaciones en la tecnología de hoy en día no permitan grandes progresos en diversos campos de interés, a medida que la tecnología avance la Realidad Aumentada probablemente cambiara y comenzara a formar parte de nuestra vida normal. En el futuro, nuestra percepción de la realidad y del mundo estará probablemente aumentada virtualmente.

Bibliografía

Billinghurst y Kato, 2002 "*Collaborative Augmented Reality*".

Billinghurst, Imamoto, Kato, Poupyrev y Tachibana, 2000 "*Virtual Object Manipulation on a Table-Top AR Environment*".

Billinghurst, Kato y Poupyrev, 2001 *"The MagicBook: A Transitional AR Interface*"

http://blog.aumentality.com/

http://blog.papervision3d.org/

http://en.wikipedia.org/wiki/ARToolKit

http://en.wikipedia.org/wiki/Augmented_reality

http://en.wikipedia.org/wiki/Homography

http://en.wikipedia.org/wiki/Thresholding_(image_processing)

http://es.scribd.com/doc/27363030/Flartoolkit-intoduction

http://flash.tarotaro.org/blog/2009/07/12/mgo2/

http://foro.aumentality.com/

http://gotoandlearn.com/play.php?id=114

http://wiiflash.bytearray.org/

http://wik.ed.uiuc.edu/index.php/Augmented_Reality_in_Education#Educational_Applications

http://words.transmote.com/wp/flarmanager/

http://www.adobe.com/products/flash-builder.html

http://www.artoolworks.com/

http://www.autodesk.com/3dsmax

http://www.bmw.com/com/en/owners/service/augmented_reality_workshop_1.html

http://www.cs.unc.edu/Research/stc/FAQs/OpenCV/OpenCVReferenceManual.pdf

http://www.hitl.washington.edu/artoolkit/

http://www.invizimals.com/

http://www.jiglibflash.com/blog/

http://www.libspark.org/wiki/saqoosha/FLARToolKit/en

http://www.mobilizy.com/

http://www.tat.se/

http://www.wikitude.org/en/

Kato, H., Billinghurst, M., October 1999 “Marker tracking and HMD calibration for a video-based augmented reality conferencing system.”

Escola Universitària
Politècnica de Mataró

Centre adscrit a:

UNIVERSITAT POLITÈCNICA
DE CATALUNYA

Ingeniería Técnica en Informática de Gestión

Proyecto Final de Carrera

Aplicaciones de la Realidad Aumentada

Estudio Económico

Guido A. Ciollaro Rodrigo-Magro
PONENT: Catalina Juan Nadal

PRIMAVERA - 2011

TecnoCampus
Mataró-Maresme

Índice

1. Costos

1.1 Costos de Software

El software básico involucrado en el proceso de desarrollo de una aplicación de AR para Adobe Flash es el siguiente:

- ARToolKit – Librería de vision Trucking
- FLARToolKit – Port de ARToolKit para Flash
- Adobe Builder – SDK para Action Script
- 3D Studio Max – Modelador 3D

Tanto ARToolKit como FLARToolKit pueden descargarse gratuitamente para propósitos no comerciales. Así mismo, podemos descargar Adobe Builder gratuitamente si somos estudiantes desde el sitio https://freeriatools.adobe.com/.

Para modelar nuestros objetos 3D podemos conseguir versiones gratuitas de modeladores 3D al ser estudiantes, incluido 3D Studio Max. El formato COLLADA también puede descargarse gratuitamente.

Por lo tanto, el software básico que se necesita para desarrollar aplicaciones de AR es gratuito y no conlleva ningún coste.

1.2 Costos de Hardware

En cuanto a hardware, los únicos dispositivos imprescindibles para desarrollar aplicaciones de AR son un ordenador y una cámara. El precio de estos hardwares puede variar, pero rondaría los:

- Cámara web: 40€.
- Ordenador: 500€

Así mismo, si se quiere realizar el ejemplo de AR utilizando un controlador externo como el WiiMote, los costes serían los siguientes:

- WiiMote: 40€
- Wii sensor bar: 10€

El coste del software necesario para utilizar el WiiMote (WiiFlash) es gratuito.

Por lo tanto, el precio del hardware dependiendo de las aplicaciones que queramos desarrollar puede variar entre 0 y 590€

Lo normal es que el usuario ya cuente con parte de este equipo. Por ejemplo, un usuario que disponga de una laptop con cámara web integrada y no requiriese utilizar un controlador externo para su aplicación de AR, no tendría coste alguno en concepto de hardware.

1.3 Costos adicionales

El objetivo de este proyecto no es un objetivo comercial, sino más bien un objetivo docente o informativo. Por ello, no se han analizado los costes siguientes:

- Costos de recursos humanos
- Costos de amortización de equipos, instrumental y software
- Gastos indirectos, como el alquiler de un local
- Costos de fabricación de prototipo

De igual manera, el costo de venta en el mercado no se ha analizado puesto que las aplicaciones aquí desarrolladas no tienen carácter comercial. Para que una aplicación que utilice FLARToolKit pueda servir para uso comercial, habrá que adquirir una licencia comercial cuyo precio puede variar dependiendo quien y para que se utilizara.

Escola Universitària
Politècnica de Mataró

Centre adscrit a:

UNIVERSITAT POLITÈCNICA
DE CATALUNYA

Ingeniería Técnica en Informática de Gestión

Proyecto Final de Carrera

Aplicaciones de la Realidad Aumentada

Apéndices

Guido A. Ciollaro Rodrigo-Magro
PONENT: Catalina Juan Nadal

PRIMAVERA - 2011

TecnoCampus
Mataró-Maresme

Índice

Apéndices

Apéndice 1: Creando un proyecto FLARToolKit en Adobe Flash Builder/Eclipse

Adobe Flash Builder es una versión modificada del IDE gratuito Eclipse, que es la razón por la cual la instalación de librerías y la creación de aplicaciones en Fash Builder y Eclipse son muy similares. En este caso, se explicara como descargar, instalar y configurar Flash Builder para compilar proyectos de FLARTooKit en Flash Player 10 y windows.

- Flash Builder puede ser descargado gratuitamente para uso no comercial desde el link siguiente https://freeriatools.adobe.com/. Para este proyecto, se utilizó Flash Builder 4. La versión más nueva hasta la fecha es la versión 4.5
- Una vez instalado Flash Builder 4, debemos proceder a descargar las librerías de FLARToolKit. Pueden ser descargadas usando un cliente SVN con el siguiente comando: *svn checkout http://www.libspark.org/svn/as3/FLARToolKit/trunk/ FLARToolKitread-only* También se pueden descargar directamente desde la web de FLARToolKit como un fichero ZIP desde el siguiente link: http://www.libspark.org/wiki/saqoosha/FLARToolKit/downloadlist.
- Para crear un nuevo proyecto debemos ir a File, New, y seleccionar "Action Script Project". Despues de escoger el nombre y ruta donde se guardara el proyecto, presionamos Continue y se crearan las carpetas que contendran el proyecto. En el campo de "Main Source Folder" debemos poner el nombre del directorio principal de nuestro proyecto (normalmente src). Una vez que el proyecto se ha creado, debemos crear el directorio "org" dentro de src y copiar las librerias de PaperVision3D libraries (incluidas en FLARToolKit), las de FLARToolKit y cualquier otras que el proyecto vaya a necesitar.
- Una vez que todo este listo, podemos empezar a trabajar. Podemos ver el "starter kit" incluido en FLARToolKit para tener un ejemplo de nuestra primera application en Realidad Aumentada.

Apéndice 2: Exportado ficheros COLLADA desde 3DStudioMax

Si queremos crear objetos 3D complejos y usarlos en la Realidad Aumentada, es muy difícil hacerlo utilizando solamente PaperVision3D u otro motor 3D. Esta es la principal razón por la que se utilizan otras apliacciones de modelado 3D como Blender, Maya o, como en este caso, 3DStudioMax. Aun teniendo en cuenta que PaperVision3D permite importar multiples formatos 3D, es mejor utilizar el formato de Collada porque es un formato open source.
Para ser capaces de utilizar el formato de Collada debemos instalarlo en nuestro software 3D. Collada tiene formatos específicos dependiendo que software 3D utilizamos. En este caso, el plugin instalado fue ColladaMax ODD versión 3.05B. La última versión puede ser descargada desde:
http://sourceforge.net/projects/colladamaya/files/

Una vez instalado, tan solo necesitamos crear nuestro modelo o animación en 3D studio, ir a File, export y seleccionar el formato "Collada .Dae" del desplegable. Un cuadro de diálogo aparecera donde podemos seleccionar las distintas opciones que querramos. Normalmente activaremos: Bake Matrices, Relative Paths y Triangulate. Si queremos

exportar un modelo que contiene animaciones, debemos activar también la opción "Enable Export".

Apéndice 3: Creando un marker para FLARToolKit

Una parte importante de FLARToolKit son los markers. En general, cada aplicación de Realidad Aumentada puede reconocer los markers indicados y, para ello, necesitamos referirnos a un fichero .pat que contiene la información acerca del marker a reconocer.

Para crear un marker personalizado, aparte de los que vienen por defecto, una buena solución es utilizar el Marker Generator Online que se puede encontrar en el siguiente link http://flash.tarotaro.org/blog/2009/07/12/mgo2/. Esta herramienta nos permite crear ficheros .pat para incluirlos en nuestra aplicación de Realidad Aumentada. La herramienta mencionada nos permite importar ficheros de imagenes con el marker en ellas o campturar directamente los markers con nuestra cámara web. Es recomendado capturar la imagen con la cámara, puesto que el resultado sera similar a lo que la cámara reconocera en una situación real.

La resolución del marker (por defecto 16x16) se puede definir así como el porcentaje del área central donde el patrón será detectado (por defecto 50%), parámetros que deberan coincidir luego en nuestra aplicación.

Una vez que el programa a reconocido el marker, presionando el botón "Get Pattern", el patrón central sera extraido y una imagen prévia se mostrara. Si la imágen prévia es correcta, usaremos el boton de "Save Current" para guardar el patrón en formato .pat.

Apéndice 4: Implementación de la clase PV3DARApp

```
package {

  import flash.display.Sprite;
  import flash.events.Event;

  import org.libspark.flartoolkit.core.transmat.FLARTransMatResult;
  import org.libspark.flartoolkit.support.pv3d.FLARBaseNode;
  import org.libspark.flartoolkit.support.pv3d.FLARCamera3D;
  import org.papervision3d.render.LazyRenderEngine;
  import org.papervision3d.scenes.Scene3D;
  import org.papervision3d.view.Viewport3D;

  public class PV3DARApp extends ARAppBase {

    protected var _base:Sprite;
    protected var _viewport:Viewport3D;
    protected var _camera3d:FLARCamera3D;
    protected var _scene:Scene3D;
    protected var _renderer:LazyRenderEngine;
    protected var _markerNode:FLARBaseNode;

    protected var _resultMat:FLARTransMatResult = new
FLARTransMatResult();

    public function PV3DARApp() {
```

```
    }

    protected override function init(cameraFile:String,
codeFile:String, canvasWidth:int = 320, canvasHeight:int = 240,
codeWidth:int = 80):void {
      addEventListener(Event.INIT, _onInit, false, int.MAX_VALUE);
      super.init(cameraFile, codeFile, canvasWidth, canvasHeight,
codeWidth);
    }

    private function _onInit(e:Event):void {
      _base = addChild(new Sprite()) as Sprite;

      _capture.width = 640;
      _capture.height = 480;
      _base.addChild(_capture);

      _viewport = _base.addChild(new Viewport3D(320, 240)) as
Viewport3D;
      _viewport.scaleX = 640 / 320;
      _viewport.scaleY = 480 / 240;
      _viewport.x = -4; // 4pix ???

      _camera3d = new FLARCamera3D(_param);

      _scene = new Scene3D();
      _markerNode = _scene.addChild(new FLARBaseNode()) as
FLARBaseNode;

      _renderer = new LazyRenderEngine(_scene, _camera3d, _viewport);

      addEventListener(Event.ENTER_FRAME, _onEnterFrame);
    }

    private function _onEnterFrame(e:Event = null):void {
      _capture.bitmapData.draw(_video);

      var detected:Boolean = false;
      try {
        detected = _detector.detectMarkerLite(_raster, 80) &&
_detector.getConfidence() > 0.5;
      } catch (e:Error) {}

      if (detected) {
        _detector.getTransformMatrix(_resultMat);
        _markerNode.setTransformMatrix(_resultMat);
        _markerNode.visible = true;
      } else {
        _markerNode.visible = false;
      }

      _renderer.render();
    }

    public function set mirror(value:Boolean):void {
      if (value) {
        _base.scaleX = -1;
        _base.x = 640;
      } else {
        _base.scaleX = 1;
        _base.x = 0;
```

```
      }
    }

    public function get mirror():Boolean {
      return _base.scaleX < 0;
    }
  }
}
```

Apéndice 5: Implementación de la clase ARAppBase

```
package {

  import flash.display.Bitmap;
  import flash.display.BitmapData;
  import flash.display.PixelSnapping;
  import flash.display.Sprite;
  import flash.events.Event;
  import flash.events.IOErrorEvent;
  import flash.events.SecurityErrorEvent;
  import flash.media.Camera;
  import flash.media.Video;
  import flash.net.URLLoader;
  import flash.net.URLLoaderDataFormat;
  import flash.net.URLRequest;

  import org.libspark.flartoolkit.core.FLARCode;
  import org.libspark.flartoolkit.core.param.FLARParam;
  import
org.libspark.flartoolkit.core.raster.rgb.FLARRgbRaster_BitmapData;
  import org.libspark.flartoolkit.detector.FLARSingleMarkerDetector;

  [Event(name="init",type="flash.events.Event")]
  [Event(name="init",type="flash.events.Event")]
  [Event(name="ioError",type="flash.events.IOErrorEvent")]
  [Event(name="securityError",type="flash.events.SecurityErrorEvent")]

  public class ARAppBase extends Sprite {

    private var _loader:URLLoader;
    private var _cameraFile:String;
    private var _codeFile:String;
    private var _width:int;
    private var _height:int;
    private var _codeWidth:int;

    protected var _param:FLARParam;
    protected var _code:FLARCode;
    protected var _raster:FLARRgbRaster_BitmapData;
    protected var _detector:FLARSingleMarkerDetector;

    protected var _webcam:Camera;
    protected var _video:Video;
    protected var _capture:Bitmap;

    public function ARAppBase() {
    }

    protected function init(cameraFile:String, codeFile:String,
canvasWidth:int = 320, canvasHeight:int = 240, codeWidth:int =
80):void {
```

```
      _cameraFile = cameraFile;
      _width = canvasWidth;
      _height = canvasHeight;
      _codeFile = codeFile;
      _codeWidth = codeWidth;

      _loader = new URLLoader();
      _loader.dataFormat = URLLoaderDataFormat.BINARY;
      _loader.addEventListener(Event.COMPLETE, _onLoadParam);
      _loader.addEventListener(IOErrorEvent.IO_ERROR, dispatchEvent);
      _loader.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
dispatchEvent);
      _loader.load(new URLRequest(_cameraFile));
    }

    private function _onLoadParam(e:Event):void {
      _loader.removeEventListener(Event.COMPLETE, _onLoadParam);
      _param = new FLARParam();
      _param.loadARParam(_loader.data);
      _param.changeScreenSize(_width, _height);

      _loader.dataFormat = URLLoaderDataFormat.TEXT;
      _loader.addEventListener(Event.COMPLETE, _onLoadCode);
      _loader.load(new URLRequest(_codeFile));
    }

    private function _onLoadCode(e:Event):void {
      _code = new FLARCode(16, 16);
      _code.loadARPatt(_loader.data);

      _loader.removeEventListener(Event.COMPLETE, _onLoadCode);
      _loader.removeEventListener(IOErrorEvent.IO_ERROR,
dispatchEvent);
      _loader.removeEventListener(SecurityErrorEvent.SECURITY_ERROR,
dispatchEvent);
      _loader = null;

      // setup webcam
      _webcam = Camera.getCamera();
      if (!_webcam) {
        throw new Error('No webcam!!!!');
      }
      _webcam.setMode(_width, _height, 30);
      _video = new Video(_width, _height);
      _video.attachCamera(_webcam);
      _capture = new Bitmap(new BitmapData(_width, _height, false, 0),
PixelSnapping.AUTO, true);

      // setup ARToolkit
      _raster = new FLARRgbRaster_BitmapData(_capture.bitmapData);
      _detector = new FLARSingleMarkerDetector(_param, _code,
_codeWidth);
      _detector.setContinueMode(true);

      dispatchEvent(new Event(Event.INIT));
    }

    protected function onInit():void {
    }
  }
}
```

Apéndice 6: Ejemplo de un fichero de configuración de FLARManager

```
<!-- this file specifies configurations for FLARManager. -->
<!-- to use this file to initialize FLARManager, pass its path into
FLARManager.initFromFile(). -->
<!-- note that all relative paths listed here must be relative to the
.swf location; absolute paths may also be used. -->

<flar_config>
  <!-- source settings -->
  <flarSourceSettings
    sourceWidth="320"
    sourceHeight="240"
    displayWidth="640"
    displayHeight="480"
    framerate="30"
    downsampleRatio="1" />

  <!-- miscellaneous FLARManager settings -->
  <flarManagerSettings
    mirrorDisplay="true"
    smoothing="3">
    <smoother className="FLARMatrixSmoother_Average" />
    <thresholdAdapter className="DrunkWalkThresholdAdapter" speed="0.3"
bias="-0.1" />
  </flarManagerSettings>

  <!-- location of camera parameters file, e.g. FLARCameraParams.dat
or camera_para.dat. -->
  <cameraParamsFile path="FLARCameraParams.dat" />

  <!-- list of file paths of patterns for FLARToolkit to detect. -->
  <!-- @resolution specifies the resolution at which the patterns were
generated. -->
  <patterns resolution="16" patternToBorderRatio="0.5"
minConfidence="0.5">
    <pattern path="patt001.pat" />
    <pattern path="patt002.pat" />
    <pattern path="patt003.pat" />
    <pattern path="patt004.pat" />
    <pattern path="patt005.pat" />
    <pattern path="patt006.pat" />
    <pattern path="patt007.pat" />
    <pattern path="patt008.pat" />
    <pattern path="patt009.pat" />
    <pattern path="patt010.pat" />
    <pattern path="patt011.pat" />
    <pattern path="patt012.pat" />
  </patterns>

</flar_config>
```

Apéndice 7: Implementación de interacción en AR

```
package {
  /* Tweener Class [http://code.google.com/p/tweener/] */
  import alternativa.engine3d.primitives.Plane;

  import caurina.transitions.Tweener;
```

```
  import com.transmote.flar.FLARManager;
  import com.transmote.flar.marker.FLARMarker;
  import com.transmote.flar.marker.FLARMarkerEvent;
  import com.transmote.flar.utils.geom.FLARPVGeomUtils;

  import flash.display.Graphics;
  import flash.display.Shape;
  import flash.display.Sprite;
  import flash.events.Event;
  import flash.events.MouseEvent;
  import flash.filters.GlowFilter;
  import flash.geom.Point;
  import flash.utils.Dictionary;

  import org.libspark.flartoolkit.support.pv3d.FLARCamera3D;
  import org.papervision3d.lights.PointLight3D;
  import org.papervision3d.materials.shadematerials.FlatShadeMaterial;
  import org.papervision3d.materials.utils.MaterialsList;
  import org.papervision3d.objects.DisplayObject3D;
  import org.papervision3d.objects.parsers.DAE;
  import org.papervision3d.objects.primitives.Cube;
  import org.papervision3d.objects.special.Graphics3D;
  import org.papervision3d.render.LazyRenderEngine;
  import org.papervision3d.scenes.Scene3D;
  import org.papervision3d.view.Viewport3D;

  import org.wiiflash.Wiimote;
  import org.wiiflash.events.ButtonEvent;
  import org.wiiflash.events.WiimoteEvent;
  import flash.events.*;

  /* Change output settings */
  [SWF(width="640", height="480", frameRate="25",
backgroundColor="#000000")]

  public class Multiple_Markers extends Sprite {
    /* FLARManager pointer */
    private var fm:FLARManager;
    /* Papervision Scene3D pointer */
    private var scene3D:Scene3D;
    /* Papervision Viewport3D pointer */
    private var viewport3D:Viewport3D;
    /* FLARToolkit FLARCamera3D pointer */
    private var camera3D:FLARCamera3D;
    /* Papervision render engine pointer */
    private var lre:LazyRenderEngine;
    /* Papervision PointLight3D pointer */
    private var pointLight3D:PointLight3D;

    /* Save the patternID of the selected Obj */
    private var selectedObj:int = -1;

    /* Initialise glow filter to add a white border around selected
objects in our scene */
    private var glow:GlowFilter = new GlowFilter(0xFFFFFF, 1, 7, 7, 30,
1, false, false);

    /* Vector storing references to all markers on screen, grouped by
pattern id */
    private var markersByPatternId:Vector.<Vector.<FLARMarker>>;
```

```
    /* Dictionary storing references to marker containers, indexed by
relevant marker object */
    private var containersByMarker:Dictionary;

    /* Dictionary storing references to the corner nodes for each
marker, indexed by relevant marker object */
    private var nodesByMarker:Dictionary;

    // Create a new Wiimote
    private var myWiimote:Wiimote = new Wiimote();

    //private var mySynchronize:Synchronize = new Synchronize();

    /* Constructor method */
    public function Multiple_Markers() {
      /* Run augmented reality initialisation */
      this.initFLAR();
    }

    /* Augmented reality initialisation */
    private function initFLAR():void {

      // connect wiimote to WiiFlash Server
      myWiimote.connect ();

      /* Initialise FLARManager */
      this.fm = new FLARManager("flarConfig.xml");

      /* Temporary declaration of how many patterns are being used */
      var numPatterns:int = 12;
      /* Initialise markerByPatternId vector object */
      this.markersByPatternId = new
Vector.<Vector.<FLARMarker>>(numPatterns, true);
      /* Loop through each pattern */
      while (numPatterns--) {
        /* Add empty Vector to each pattern */
        this.markersByPatternId[numPatterns] = new
Vector.<FLARMarker>();
      }

      /* Initialise empty containersByMarker dictionary object */
      this.containersByMarker = new Dictionary(true); //weakKeys = true

      /* Initialise empty nodesByMarker dictionary object */
      this.nodesByMarker = new Dictionary(true);

      /* Event listener for when a new marker is recognised */
      fm.addEventListener(FLARMarkerEvent.MARKER_ADDED, this.onAdded);
      /* Event listener for when a marker is removed */
      fm.addEventListener(FLARMarkerEvent.MARKER_REMOVED,
this.onRemoved);
      /* Event listener for when the FLARManager object has loaded */
      fm.addEventListener(Event.INIT, this.onFlarManagerLoad);

      /* Event listener for the Wiimote.CLICK */
      stage.addEventListener( MouseEvent.CLICK, selectObj);

      /* Display webcam */
      this.addChild(Sprite(fm.flarSource));
    }
```

```
    private function selectObj(e:MouseEvent):void {

      //myWiimote.rumbleTimeout = 1000;
      trace("SensorX:"+ myWiimote.sensorX);
      trace("MouseX:"+ stage.mouseX);
      myWiimote.mouseControl = false;
      /*if(myWiimote.mouseControl) myWiimote.mouseControl = false;
      else myWiimote.mouseControl = false;*/

      var numPatterns:int = markersByPatternId.length;

      while (numPatterns--){

        var markerList:Vector.<FLARMarker> =
this.markersByPatternId[numPatterns];

        if (markerList.length > 0){
          var marker:FLARMarker = markerList[0];

          var corners:Vector.<Point> = marker.corners;
          var p:Point = new Point(e.stageX, e.stageY);

          if (insidePolygon(corners, p)){
            trace("Hit!! Instance: "+ marker.patternId );

            if (selectedObj != -1){
              var auxMarkerList:Vector.<FLARMarker> =
this.markersByPatternId[selectedObj];
              var auxContainer:DisplayObject3D =
this.containersByMarker[auxMarkerList[0]];
              var auxChild:DisplayObject3D =
auxContainer.removeChildByName("c");
              if (auxChild != null){
                auxChild.scale = 1;
                auxContainer.addChild(auxChild);
              }
            }
            selectedObj = marker.patternId;
            var container:DisplayObject3D =
this.containersByMarker[marker];
            var child:DisplayObject3D =
container.removeChildByName("c");
            if (child != null){
              child.scale = 5;
              container.addChild(child);
            }
          }
        }
      }
    }

    /* Run if FLARManager object has loaded */
    private function onFlarManagerLoad(e:Event):void {
      /* Remove event listener so this method doesn't run again */
      this.fm.removeEventListener(Event.INIT, this.onFlarManagerLoad);
      /* Run Papervision initialisation method */
      this.initPaperVision();
    }

    /* Run when a new marker is recognised */
    private function onAdded(e:FLARMarkerEvent):void {
```

```
      /* Run method to add a new marker */
      this.addMarker(e.marker);
    }
    /* Run when a marker is removed */
    private function onRemoved(e:FLARMarkerEvent):void {
      /* Run method to remove a marker */
      this.removeMarker(e.marker);
    }

    /* Add a new marker to the system */
    private function addMarker(marker:FLARMarker):void {
      /* Store reference to list of existing markers with same pattern
id */
      var markerList:Vector.<FLARMarker> =
this.markersByPatternId[marker.patternId];
      /* Add new marker to the list */
      markerList.push(marker);

      /* Initialise the marker container object */
      var container:DisplayObject3D = new DisplayObject3D();
      if (marker.patternId == 0){
        // load the model.
        // (this model has to be scaled and rotated to fit the marker;
every model is different.)
        var model:DAE = new DAE(true, "model", true);
        model.load("scout.dae");
        model.rotationX = 90;
        model.rotationZ = 90;
        model.scale = 0.5;
        model.name = "c";

        container.addChild(model);
        container.visible = true;
        scene3D.addChild(container);

      }else{
        /* Prepare material to be used by the Papervision cube based on
pattern id */
      var flatShaderMat:FlatShadeMaterial = new
FlatShadeMaterial(pointLight3D, getColorByPatternId(marker.patternId),
getColorByPatternId(marker.patternId, true));
      /* Add material to all sides of the cube */
      var cubeMaterials:MaterialsList = new MaterialsList({all:
flatShaderMat});

      /* Initialise the cube with material and set dimensions of all
sides to 40 */
      var cube:Cube = new Cube(cubeMaterials, 40, 40, 40);

      /* Shift cube upwards so it sits on top of paper instead of being
cut half-way */
      cube.z = 0.5 * 40;

      /* Scale cube to 0 so it's invisible */
      cube.scale = 0;

      /* Add animation which scales cube to full size */
      Tweener.addTween(cube, {scale: 1, time:0.5,
transition:"easeInOutExpo"});

      /* Set cube to be individually affected by filters */
```

```
      cube.useOwnContainer = true;
      /* Add cellshaded border using glow filter */
      cube.filters = [this.glow];

      cube.name = "c";
      /* Add finished cube object to marker container */
      container.addChild(cube);
      /* Add marker container to the Papervision scene */
      this.scene3D.addChild(container);

      }

      /* Add marker container to containersByMarker Dictionary object
*/
      this.containersByMarker[marker] = container;
    }

    /* Remove a marker from the system */
    private function removeMarker(marker:FLARMarker):void {
      /* Store reference to list of existing markers with same pattern
id */
      var markerList:Vector.<FLARMarker> =
this.markersByPatternId[marker.patternId];
      /* Find index value of marker to be removed */
      var markerIndex:uint = markerList.indexOf(marker);
      /* If marker exists in markerList */
      if (markerIndex != -1) {
        /* Remove marker from markersByPatternId */
        markerList.splice(markerIndex, 1);
      }

      /* Store reference to marker container from containersByMarker
Dictionary object */
      var container:DisplayObject3D = this.containersByMarker[marker];
      /* If a container exists */
      if (container) {
        /* Remove container from the Papervision scene */
        this.scene3D.removeChild(container);
      }
      /* Remove container reference from containersByMarker Dictionary
object */
      delete this.containersByMarker[marker];

      /* Clear any corner nodes for this marker from the display */
      this.nodesByMarker[marker].graphics.clear();
      /* Remove reference to corner nodes for this marker from
nodesByMarker Dictionary object */
      delete this.nodesByMarker[marker];
    }

    /* Papervision initialisation method */
    private function initPaperVision():void {
      /* Initialise a new Papervision scene */
      this.scene3D = new Scene3D();
      /* Initialise a new FLARCamera3D object to enable full AR
goodness */
      this.camera3D = new FLARCamera3D(this.fm.cameraParams);

      /* Define a new Papervision viewport object */
      this.viewport3D = new Viewport3D(640, 480, true);
      /* Add viewport to the main scene */
```

```
      this.addChild(this.viewport3D);

      /* Define a new Papervision point light */
      this.pointLight3D = new PointLight3D(true, false);
      /* Set light position */
      this.pointLight3D.x = 1000;
      this.pointLight3D.y = 1000;
      this.pointLight3D.z = -1000;
      /* Add light to the Papervision scene */
      this.scene3D.addChild(pointLight3D);

      /* Initialise the Papervision render engine */
      this.lre = new LazyRenderEngine(this.scene3D, this.camera3D,
this.viewport3D);

      /* Create event listner to run a method on each frame */
      this.addEventListener(Event.ENTER_FRAME, this.onEnterFrame);
    }

    /* Method to run on each frame */
    private function onEnterFrame(e:Event):void {
      /* Loop through corner nodes */
      for (var marker:Object in nodesByMarker) {
        /* Clear any corner nodes for this marker from the display */
        nodesByMarker[marker].graphics.clear();
      }

      /* Run method to update markers */
      this.updateMarkers();

      /* Render the Papervision scene */
      this.lre.render();
    }

    /* Update markers method */
    private function updateMarkers():void {
      /* Store reference to amount of patterns being tracked */
      var i:int = this.markersByPatternId.length;
      /* Store reference to list of existing markers */
      var markerList:Vector.<FLARMarker>;
      /* Empty marker variable */
      var marker:FLARMarker;
      /* Empty container variable */
      var container:DisplayObject3D;
      /* Empty integer */
      var j:int;

      /* Loop through all tracked patterns */
      while (i--) {
        /* Store reference to all markers with this pattern id */
        markerList = this.markersByPatternId[i];
        /* Amount of markers with this pattern */
        j = markerList.length;
        /* Loop through markers with this pattern */
        while (j--) {
          /* Store reference to current marker */
          marker = markerList[j];

          /* Initialise a new Shape object */
          var nodes:Shape = new Shape();
          /* Define line style for corner nodes */
```

```
          nodes.graphics.lineStyle(3,
getColorByPatternId(marker.patternId));
          /* Store reference to coordinates for each corner of the
marker */
          var corners:Vector.<Point> = marker.corners;
          /* Empty coordinate variable */
          var vertex:Point;
          /* Loop through rest of corner coordinates */
          for (var c:uint=0; c<corners.length; c++) {
            /* Store reference to current corner coordinates */
            vertex = corners[c];
            /* Draw a 2D circle at these coordinates */
            nodes.graphics.drawCircle(vertex.x, vertex.y, 5);
          }
          /* Add reference to corner nodes to nodesByMarker Dictionary
object */
          this.nodesByMarker[marker] = nodes;
          /* Add corner nodes to the main scene */
          this.addChild(nodes);

          /* Find reference to marker container in containersByMarker
Dictionary object */
          container = this.containersByMarker[marker];
          /* Transform container to new position in 3d space */
          container.transform =
FLARPVGeomUtils.convertFLARMatrixToPVMatrix(marker.transformMatrix);
        }
      }
    }

    /* Get colour values dependent on pattern id */
    private function getColorByPatternId(patternId:int, shaded:Boolean
= false):Number {
      switch (patternId) {
        case 0:
          if (!shaded)
            return 0xFF1919;
          return 0x730000;
        case 1:
          if (!shaded)
            return 0xFF19E8;
          return 0x730067;
        case 2:
          if (!shaded)
            return 0x9E19FF;
          return 0x420073;
        case 3:
          if (!shaded)
            return 0x192EFF;
          return 0x000A73;
        case 4:
          if (!shaded)
            return 0x1996FF;
          return 0x003E73;
        case 5:
          if (!shaded)
            return 0x19FDFF;
          return 0x007273;
        default:
          if (!shaded)
            return 0xCCCCCC;
```

```
            return 0x666666;
        }
    }

    /* Private function to know if a Point is inside a polygon */
    private function insidePolygon(pointList:Vector.<Point>,
p:Point):Boolean
    {
      var counter:int = 0;
      var i:int;
      var xinters:Number;
      var p1:Point;
      var p2:Point;
      var n:int = pointList.length;

      p1 = pointList[0];
      for (i = 1; i <= n; i++)
      {
        p2 = pointList[i % n];
        if (p.y > Math.min(p1.y, p2.y))
        {
          if (p.y <= Math.max(p1.y, p2.y))
          {
            if (p.x <= Math.max(p1.x, p2.x))
            {
              if (p1.y != p2.y) {
                xinters = (p.y - p1.y) * (p2.x - p1.x) / (p2.y - p1.y)
+ p1.x;
                if (p1.x == p2.x || p.x <= xinters)
                  counter++;
              }
            }
          }
        }
        p1 = p2;
      }
      if (counter % 2 == 0)
      {
        return(false);
      }
      else
      {
        return(true);
      }
    }
  }
}
```