TESINA DE LICENCIATURA

Título: Esquemas para la administración de múltiples cámaras en aplicaciones de Realidad Aumentada
Autor: A.P.U. Mauro Agustín Espósito
Directora: Dra. Silvia Gordillo
Carrera: Licenciatura en Sistemas (P03)

Resumen

Teniendo en cuenta el auge de la Realidad Aumentada como tecnología aplicada al desarrollo de aplicaciones de entretenimiento, marketing, educación y otras áreas, esta tesis propone un nuevo concepto: el uso de múltiples cámaras para mejorar la experiencia del usuario.
A partir de esta idea se proponen varios esquemas para poder manejar esas cámaras, mencionando las ventajas y falencias de cada uno y en qué manera pueden utilizarse para adaptar aplicaciones convencionales orientadas a una única cámara.
El análisis de un caso de estudio particular permite demostrar qué aspectos deben ser tenidos en cuenta antes de elegir alguno de los esquemas propuestos y qué resultados puede obtener el usuario final.

Palabras Claves

Realidad Aumentada, múltiples cámaras, esquemas de administración, modelos 3d, markers, matriz de transformación

Trabajos Realizados

Se estudiaron las limitaciones de aplicaciones convencionales de Realidad Aumentada con una sola cámara. Se propuso la adaptación de esas aplicaciones a múltiples cámaras y se crearon diversos esquemas para poder administrarlas.
Se analizó el proceso de detección de patrones para sacar provecho de los datos que éste otorga, como ser la matriz de transformación, para el desarrollo de algunos de esos esquemas.

Conclusiones

Sobre la base de un framework para la creación de aplicaciones convencionales de RA en ActionScript, se desarrolló una extensión que permite incorporar más de una cámara y separar la implementación del modelo a renderizar. Esta librería provee diversos esquemas para manejar las cámaras, permitiendo desarrollar nuevos e incorporarlos fácilmente.
Se pudo obtener la distancia real entre una cámara web y el patrón detectado.

Trabajos Futuros

A partir del trabajo realizado y los resultados obtenidos, se plantean ciertos temas para su futura investigación:
- Estudio del recorrido de una persona a partir del cálculo de la distancia;
- Diseñar un sistema que aprenda, a partir del análisis del comportamiento de modelos animados, para adelantarse a las acciones y realizar mejores tomas;
- Incorporar las ventajas de las nuevas versiones de Adobe Flash, como el motor de renderizado 3d nativo, para optimizar las aplicaciones.

Fecha de la presentación: OCTUBRE 2011

Esquemas para la administración de múltiples cámaras en aplicaciones de Realidad Aumentada

Autor: A.P.U. Mauro Agustín Espósito
Directora: Dra. Silvia Gordillo

Tesina de grado
Licenciatura en Sistemas
Facultad de Informática
Universidad Nacional de La Plata

Agradecimientos

Quiero agradecer a la Dra. Silvia Gordillo, por permitir desarrollar mi tesis de grado bajo su tutoría, por el respaldo brindado en todo momento y su completa predisposición.

Al Dr. Federico Balaguer, por su gran ayuda y motivación. Sus ideas y consejos marcaron los pasos de esta tesis, aportando toda su experiencia para mejorar la calidad del trabajo realizado.

A mis compañeros de estudio y el personal del LIFIA, con quienes tuve la suerte de compartir este recorrido. Sin ellos el camino hubiese sido más largo y mucho menos ameno.

A los amigos de la vida, quienes siempre me apoyaron. Su aliento y compañía fue esencial para poder llegar a esta instancia.

Por último, agradezco profundamente a mi familia por haberme apoyado durante todos estos años. Ellos me ayudaron en todo momento, me incentivaron y me bancaron en el transcurso de toda la carrera. Gracias a ellos por permitirme desarrollarme como profesional pero, en especial, por servirme de ejemplo como personas.

Índice

Capítulo 1

Introducción

El concepto de "aumentar la realidad" agregándole una capa de información extra constituye una nueva forma de comunicar, que ha ido creciendo notablemente en los últimos años. Más y mejores aplicaciones aparecen continuamente sacando provecho de esta idea: planos de edificios que se yerguen sobre el papel, juguetes que pueden ser probados antes de comprarlos, catálogos de ropa que permiten variar el color de las prendas, fotos carnet que se dibujan en tarjetas personales y muchas más.

Según el público al que estén dirigidas, sus prestaciones, y otros parámetros, pueden desarrollarse para ser ejecutadas en computadoras personales, dispositivos móviles u otras plataformas menos convencionales, como por ejemplo visores dedicados. El abanico de contenidos que se añaden va desde simples textos informativos, pasando por fotos e imágenes en dos dimensiones hasta grandes animaciones en 3D.

Al ser una tecnología que aún no a llegado a un alto nivel de madurez, a menudo se presentan situaciones, tanto para el desarrollador como para el usuario, en las que no pueden sacar provecho de todo el todo el potencial que ésta provee. Muchas aplicaciones de las que se ejecutan en computadoras de escritorio o laptops pueden encontrarse con una importante limitación: cuando renderizan modelos tridimensionales de gran complejidad, o cuando involucran animaciones e interacciones que se extienden en el espacio, la percepción del usuario se ve afectada dado que, con las herramientas disponibles, no es posible mostrar en dos dimensiones todo lo que sucede en un espacio de tres. Esto se debe a que el mecanismo para proyectar modelos 3d en una pantalla de dos ejes, debe descartar aquellas caras que no se encuentran "mirando" hacia la cámara que captura la escena, y también todos los objetos que excedan a su ángulo de vista.

Se hace evidente que el usuario no puede acceder a la totalidad del contenido sin interactuar con la aplicación, ya sea moviendo la cámara o bien moviendo el patrón a reconocer, para que la proyección muestre lo que se encuentra oculto.

Este trabajo propone otra manera de acceder a esa información: incorporando más cámaras en la misma aplicación. Ubicándolas en puntos estratégicos del espacio, el usuario podría contemplar todas las partes, comportamientos e interacciones que hasta el momento resultaban inaccesibles.

Para hacer más clara esta idea se puede plantear un paralelismo con el ámbito cinematográfico. Durante el rodaje de una película, un director puede realizar diversas tomas de la misma escena. Al momento de compilarlas debe tomar una decisión en relación a cómo las va a mostrar, en base a la

situación que determina el guión. Por ejemplo, si se trata de un dialogo entre dos actores alternará primeros planos de sus caras, o si registra una persecución, las irá secuenciando de manera de tener siempre en escena al protagonista. El hecho de involucrar más de una cámara en aplicaciones de Realidad Aumentada, obliga a tomar decisiones similares. No existe una única manera de secuenciar las filmaciones dado que todas las aplicaciones son distintas y renderizan modelos con animaciones y complejidades diferentes.

Para hacer frente a esta situación, se presentan aquí diferentes estrategias o esquemas para el manejo de las cámaras. El tipo de aplicación y el comportamiento que ésta exponga será crucial para definir la elección de alguno de ellos. De esta manera, el desarrollador se desentiende de la tarea de pensar una nueva forma de mostrar las cámaras en cada aplicación que codifica, sino que continúa focalizando su atención en los mismos aspectos como si se tratase de un desarrollo convencional.

1.1 Tecnologías

Las primeras aplicaciones de Realidad Aumentada fueron codificadas originalmente en C++, lenguaje con el cual también fueron diseñadas las primeras librerías (*ARToolkit*[1] y *ARToolkitPlus*[2]). Estas permitieron a los desarrolladores la creación de aplicaciones dentro de un marco de trabajo que brindara facilidades en los aspectos más importantes: la conectividad con la cámara, el reconocimiento de imágenes y el renderizado e incorporación de modelos 3D.

Con el correr del tiempo, estos conceptos fueron trasladados a otros lenguajes como Java y ActionScript, incorporando así desarrolladores de otras comunidades. Esta expansión facilitó su inclusión en el desarrollo de Aplicaciones de Internet Enriquecidas (*RIA,* de *Rich Internet Applications*), que actualmente abundan en la web.

La librería propuesta en este trabajo junto con el caso de estudio que hace uso de ella, se encuentran codificados íntegramente en ActionScript.

1.2 Estructura de la tesina

En el capítulo 2 se introduce el concepto de Realidad Aumentada, junto con sus aplicaciones y una descripción de sus partes y su funcionamiento. Se describen también las tecnologías utilizadas a lo largo de este trabajo.

En el capítulo 3 se presenta la librería desarrollada. Se realiza una descripción general de su diseño y el detalle de cada uno de los esquemas implementados.

El capítulo 4 se muestra un caso de estudio utilizado para la prueba de la librería. Se describen los puntos más importantes a la hora de adaptar una aplicación convencional junto con una detallada explicación de la prueba de los esquemas propuestos.

En el capitulo 5 se presentan conclusiones sobre el trabajo desarrollado. Se brinda un análisis sobre los resultados obtenidos, aportes y trabajo pendiente.

Capítulo 2

Background

2.1 Realidad Aumentada

2.1.1 ¿Qué es la Realidad Aumentada?

La Realidad Aumentada (RA) es una técnica que se utiliza para mejorar la experiencia del usuario enriqueciendo contenidos interactivos [7]. Otorga una visualización en tiempo real de un entorno físico, capturado con algún dispositivo de entrada, al que se le agregan objetos virtuales mediante un proceso de software.

2.1.2 Sus plataformas

Una plataforma de RA requiere, como mínimo, tres elementos: una entrada de video, un procesador y una pantalla. Existe una gran variedad de plataformas [21] en los cuales pueden ejecutarse este tipo de aplicaciones, desde computadoras de escritorio y *laptops* (utilizando una *webcam* o la camára integrada) hasta dispositivos móviles tales como *Palms* o *smartphones,* incluyendo otros dispositivos menos convencionales como *Head Mounted Displays* (similares a cascos de Realidad Virtual). Este trabajo tiene su foco de atención en las primeras dos: pcs de escritorio y laptops, los medios más populares dado que están al alcance de la mayoría de los usuarios. Mediante ellas se pueden crear aplicaciones de RA de escritorio o *dekstop*.

2.1.3 Distintos usos

Hoy en día, la RA se ha convertido en una vía de comunicación ampliamente utilizada en diferentes sectores como en la educación, el entretenimiento y el marketing empresarial, dado que permite brindar información de una forma novedosa y capta la atención de muchos usuarios. A continuación se listan algunos ejemplos:

- Estando en un museo de Ciencias Naturales, una persona puede colocarse un visor de RA y observar, superpuesto sobre un esqueleto, el cuerpo de un dinosaurio;
- Una compañía de autos muestra sobre un anuncio un prototipo de un nuevo modelo, completamente en tres dimensiones;
- Observando edificios históricos a través de la cámara de un celular, se puede ver en pantalla información sobre su fecha de construcción, su importancia, su aporte a la historia, y mucha más información.

Figura 2.1. (Arriba - izquierda). Un dinosaurio saliendo de un libro didáctico de RA [22].
(Arriba - derecha). Un auto Mini Cooper sobre una revista de la compañía.
(Abajo) Información turística en un smartphone.

2.1.4 Elementos de RA desktop basada en markers

En esta sección se explican los elementos que entran en juego a la hora de utilizar esta técnica. Si bien existen distintos tipos de plataformas, distintos ambientes de programación y lenguajes, este conjunto es común a todos ellos.

2.1.4.1 Cámaras y escena 3d

Uno de los componentes de mayor importancia es la **cámara**. Es importante destacar que conviven 2 cámaras de distinto tipo: una cámara para capturar video y una cámara 3d. En primer lugar se necesita una entrada de video; cómo este trabajo se enfoca en las aplicaciones desktop, se utiliza una cámara web. En segundo lugar, se debe dibujar una escena en tres dimensiones, por lo tanto se hace uso de una cámara 3d, que será el punto de percepción de la escena, desde dónde se ve lo que está sucediendo

La **escena 3d** está determinada por el conjunto de todos los objetos 3d que se encuentran en el espacio 3d [16, 17]. Cada objeto que se quiere mostrar debe ser agregado en la escena.

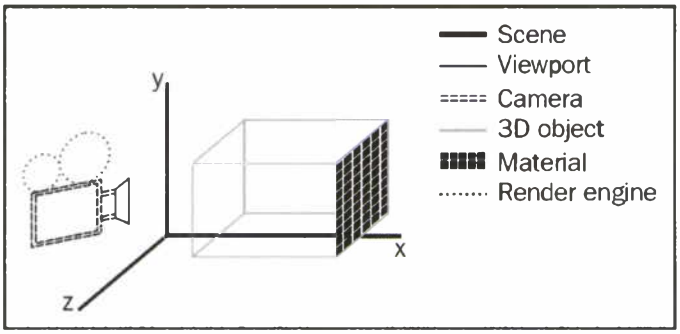


Figura 2.2. Elementos de la realidad aumentada

En un principio puede resultar difícil hacer esta diferenciación dado que la escena percibida por la cámara 3d se inyecta en el flujo de video otorgado por la cámara web.

2.1.4.2 El marker

Otro componente clave en la aplicación es el patrón que se provee, aquel que se espera que sea reconocido por la webcam. Se trata de una imagen con contrastes bien definidos, que debe ser asimétrica, para que se vea diferente en cada rotación de 90° y sea identificable su orientación.

A partir de esta imagen debe generarse un archivo binario, que la aplicación será capaz de procesar para buscar coincidencias (*pattern matching*), y además se debe proveer en un formato que permita su impresión (.jpg, .png, etc) para tenerlo en papel y mostrarlo ante la cámara.

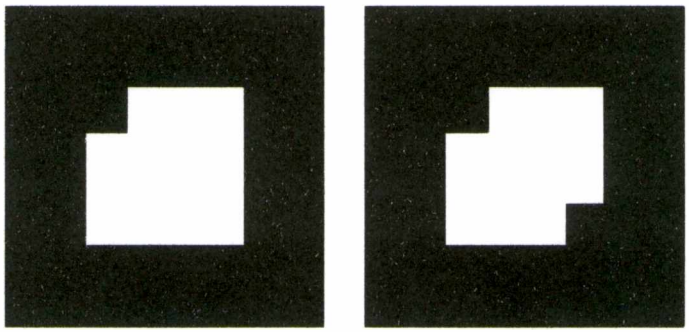


Figura 2.3. Se ha podido comprobar que es más efectiva la detección de *markers* cuadrados, de un tamaño aproximado de 8cm de lado. La imagen de la izquierda es un buen ejemplo, mientras que la de la derecha podría no identificarse correctamente

A partir del marker, ahora es responsabilidad del algoritmo de detección realizar las acciones pertinentes para intentar descubrirlo dentro de una imagen más compleja.

En principio, se toma cada *frame* del flujo de video que proviene de una cámara y se envía a procesar. Si se tiene en cuenta que el promedio de FPS (*frames* por segundo) es de 30, esta acción es imperceptible, dando la sensación de que se analiza el video en movimiento. Ahora lo que se tiene es un mapa de bits, de un tamaño especificado (según la resolución más apropiada para la cámara y otros parámetros) que será analizado.

Los pasos que comprende el análisis son los siguientes:

1. A partir del mapa de bits de origen, se debe obtener una imagen binaria. Esto se conoce como ***thresholding***, dado que se logra en base a un número denominado **valor de *threshold*.** Los píxeles que constituyen el mapa de bits tienen un valor numérico. Este valor de threshold indica un umbral, de tal manera que aquellos píxeles con valores menores que él pasan a ser de un color, y los mayores, a otro color (negro y blanco). Por esta razón, cuando se trabaja con imágenes binarias, se facilita este proceso, obteniendo mejores resultados.

Figura 2.4. El thresholding convierte la imagen de origen en una imagen binaria, haciendo que el análisis computacional sea menos costoso.

2. Se deben buscar aéreas contiguas dentro de la imagen binaria, para poder "etiquetarlas". A cada área blanca encerrada se le asigna un color con el que se la identificará posteriormente. Este paso se conoce como ***labeling***.

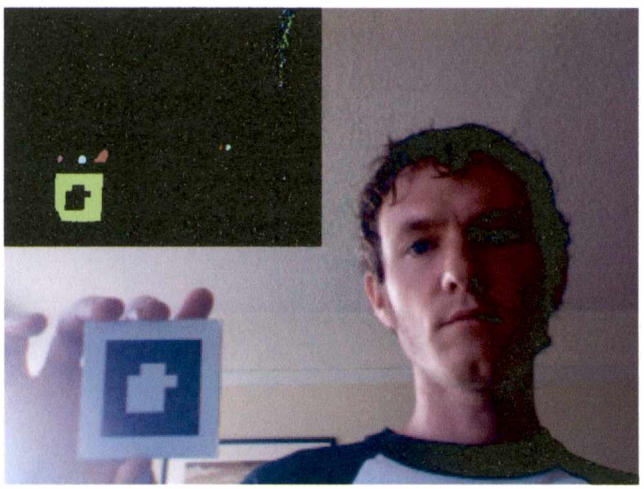


Figura 2.5. Cada área blanca contigua (correspondiente a un área oscura en la imagen de origen) es 'etiquetada' con un color diferente.

Las áreas que se han coloreado son los posibles candidatos a coincidir con el marker.

3. Por último se deben buscar formas que puedan ser convertidas a cuadrados, similares al patrón. Una vez que los cuadrados fueron identificados, se procede a comparar el contenido que se

encuentra dentro de las líneas externas con la imagen que se proporcionó como entrada para la aplicación. El algoritmo le asigna un *nivel de confianza* a cada reconocimiento que realiza, aquellos que tengan un nivel similar o superior al que se especifique en la aplicación, serán reportados como posibles coincidencias. Un nivel de confianza apropiado es 0.5 (debe ser un valor entre 0 y 1).

2.1.4.3 Modelo 3D

Un modelo 3D es una representación de un objeto lograda a partir de conjuntos de polígonos a los que se les aplican texturas. Mediante un proceso conocido como **renderizado** (del inglés *render*) es posible mapear estos objetos 3D a una pantalla 2D.

En las diversas aplicaciones que han sido creadas en RA se puede ver una gran variedad de objetos en 2D y 3D, textos o animaciones.

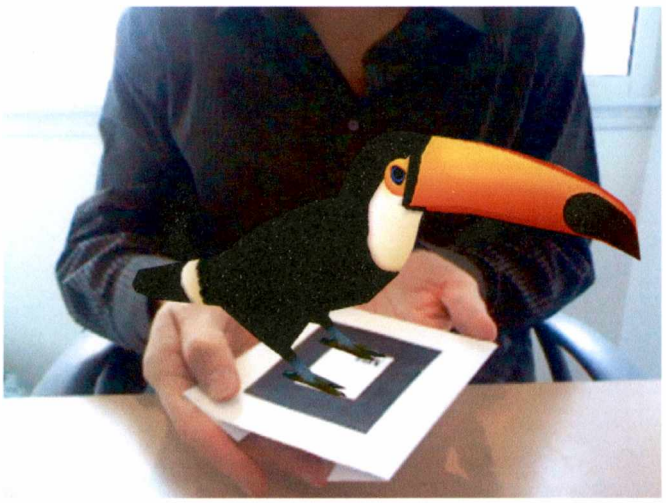


Figura 2.6. El modelo un tucán sobre el marker

Un efecto visual llamativo de la RA es la sensación de que el modelo renderizado esta "pegado" al patrón, manteniendo la relación de aspecto con el mismo. Esto se logra de la siguiente manera:

1. Si el proceso de detección del marker descrito en la sección anterior tuvo éxito, el algoritmo genera una matriz con datos sobre su ubicación en el espacio. Esta matriz se denomina **matriz de transformación** y constituye uno de los focos más importantes de este trabajo.
2. La aplicación dibuja el modelo 3d que se ha especificado, en la posición deseada (normalmente se dibuja como si estuviese apoyado sobre el marker).

3. Inmediatamente, al modelo se le aplica la matriz de transformación que se obtuvo del marker, esto quiere decir que se escala el tamaño para mantener relación con el patrón y se rota para que tengan la misma orientación.

Si se tiene en cuenta que los pasos anteriores se realizan en cada *frame* de video, cuando se tiene un buen seguimiento o *tracking* del patrón, se crea la sensación de que el modelo está enganchado al marker, porque aumenta de tamaño cuando se acerca a la cámara y disminuye cuando se aleja, o rota cuando se gira el marker sobre algún eje.

2.1.4.4 La matriz de transformación

La matriz de transformación es un arreglo multidimensional que determina cómo se asignan puntos de un espacio de coordenadas a otro (en este caso, de 2 dimensiones a 3 dimensiones) [15]. Contiene información importante relacionada con la ubicación y orientación del patrón, mas precisamente, sus dimensiones (en milímetros), los valores de su posición en los tres ejes (X, Y y Z) y su rotación en cada uno de ellos.

Es posible realizar varias transformaciones gráficas en un objeto (traslación, escala, rotación, inclinación) mediante la definición de valores en una matriz y luego aplicando dicha matriz al objeto.

Traslación

$$\begin{vmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Desplaza el modelo
- **dx** unidades en el eje **X**
- **dy** unidades en el eje **Y**
- **dz** unidades en el eje **Z**

Rotación

$$\begin{vmatrix} \cos(\Theta) & -\sin(\Theta) & 0 & 0 \\ \sin(\Theta) & \cos(\Theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Rota el modelo Θ grados en el eje **Z**

Escala	
$\begin{vmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$	Escala el modelo **n** veces su tamaño

Las tablas anteriores muestran como se plasman las transformaciones en la matriz. De igual manera, cuando se hacen modificaciones directamente en las propiedades de un objeto, éstas se interpretan como operaciones a realizar sobre la matriz del objeto para reflejar los cambios.

```
1. modelo.x = modelo.x + 10;    //mueve el modelo 10 unidades a la derecha
2. modelo.scale = 3;            //escala el modelo al triple de su tamaño
3. modelo.rotationY = 90;       //rota el modelo 90 grados en el eje Y
```

Tabla 2.4. Sentencias de transformación del modelo

2.1.4.5 Análisis de la matriz de transformación

Es de interés para una de las soluciones propuestas en este trabajo, conocer cuál es la distancia real entre la cámara web y el marker de papel que se está filmando. Para esto, resulta de mucha utilidad estudiar los valores de la matriz de transformación. Los siguientes pasos demuestran cómo es posible obtener la información que se necesita.

Si se tiene la siguiente imagen, la matriz refleja que el punto es en el que se encuentra el marker es el (0,0,0) y los ángulos de rotación son de 0° para los tres ejes.

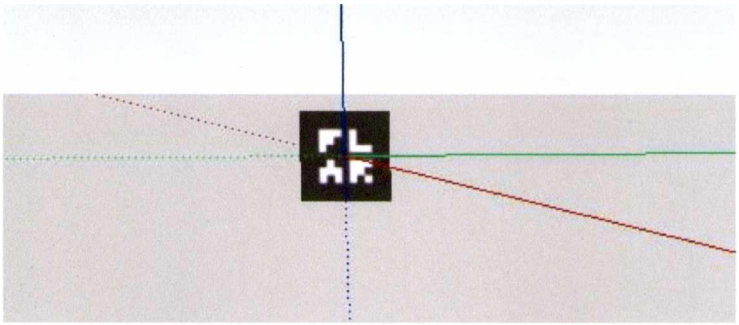


Figura 2.7. El marker sin rotaciones en el punto (0, 0, 0)

Ahora bien, en la siguiente imagen, donde se puede ver que el marker ha rotado levemente, la matriz correspondiente refleja que éste se encuentra en el punto (0, 0, 0) pero ahora sus ángulos de rotación son 0° para el eje **X**, 0° para el eje **Y** y 45° para el eje **Z** (rotación hacia la derecha).

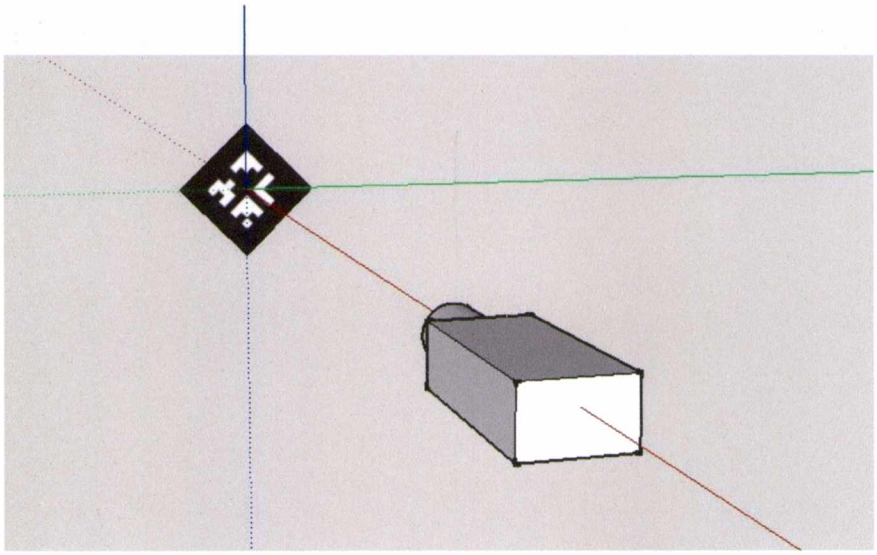


Figura 2.8. El marker se percibe rotado 45° hacia la derecha

Otro punto de vista

Teniendo la misma imagen, otra posible interpretación consiste en pensar que el marker no ha rotado, sino que la cámara lo ha hecho, pero en sentido opuesto, es decir 45° hacia la izquierda en el eje **Z**. Si se pudiera tener una matriz que represente los valores de la cámara, sería similar a la del marker pero invertida, es decir, se podría percibir que hay una rotación de -45° para el eje **Z**.

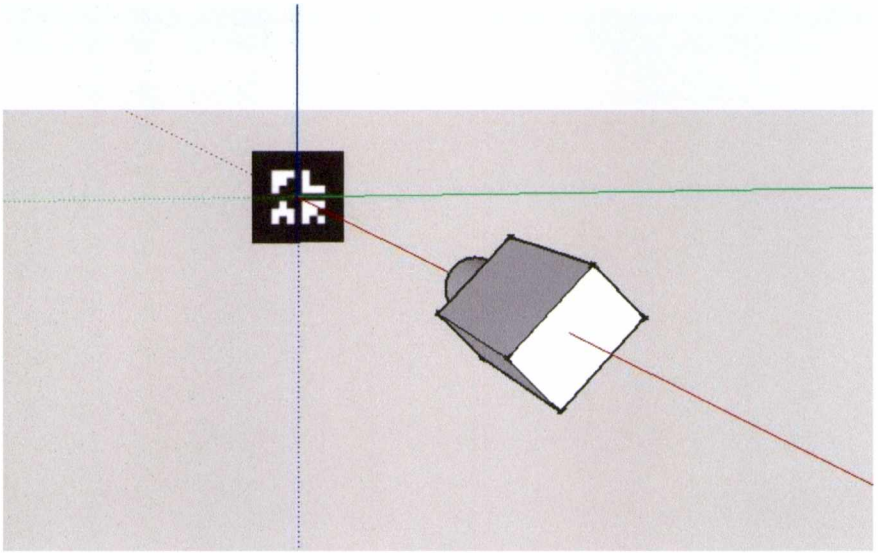


Figura 2.9. La cámara se considera rotada 45° hacia la izquierda

La respuesta consiste, entonces, en invertir la matriz de transformación, para trabajar con valores que representen la **posición y rotación de la cámara con respecto al marker**.

La matriz que se obtiene tiene la siguiente estructura:

$$\begin{vmatrix} m11 & m12 & m13 & m14 \\ m21 & m22 & m23 & m24 \\ m31 & m32 & m33 & m34 \\ m41 & m42 & m43 & m44 \end{vmatrix}$$

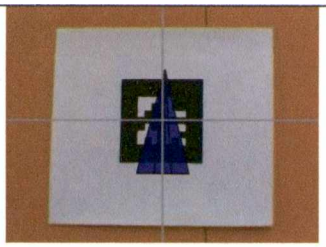
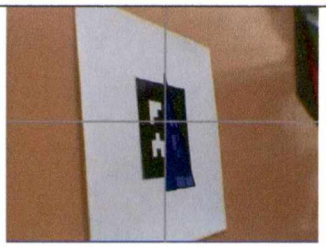
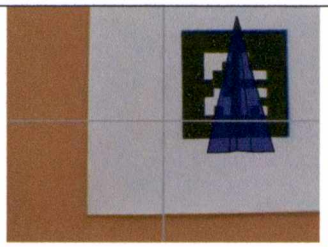
Los primeros tres valores de la última columna representan las coordenadas de la cámara en los tres ejes:

Posición = [**X** = *m14*, **Y** = *m24*, **Z** = *m34*];

Para obtener la rotación de la cámara se debe aplicar una función que recibe la matriz completa y devuelve un punto de 3 valores con los ángulos de Euler (**x**, **y** y **z**):

Rotación = *convertMatrixToEuler(matriz)*

En el siguiente ejemplo se imprimen en pantalla los valores de posición y rotación obtenidos para distintas ubicaciones de la cámara:

 Cámara 1	Posición [x = 0, y = 0, **z = 50**] Rotación [x = 0, y = 0, z = 0]
 Cámara 2	Posición [**x = -250**, y = 0, **z = 20**] Rotación [x = 0, **y = -50**, z = 0]
 Cámara 3	Posición [**x = 250**, y = 0, **z = 20**] Rotación [x = 0, **y = 50**, z = 0]
 Cámara 4	Posición [**x = -10, y = -10, z = 50**] Rotación [x = 0, y = 0, z = 0]

A partir de estos valores, ahora es momento de calcular la distancia concreta. La función que se debe utilizar es muy conocida en matemáticas [13] y permite saber la distancia entre dos puntos cualesquiera del espacio:

$$d_{\overline{AB}} = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2 + (z_B - z_A)^2}$$

Figura 2.10. Cálculo de la distancia entre dos puntos del espacio de tres dimensiones

Esta función retorna la distancia real (en milímetros) entre la cámara web y el marker.

2.2 Actionscript

El desarrollo propuesto esta completamente codificado en ActionScript [4] en su versión 3.0 (AS3). Se trata de un lenguaje de programación Orientado a Objetos utilizado en especial en aplicaciones web animadas realizadas en el entorno *Adobe Flash*.

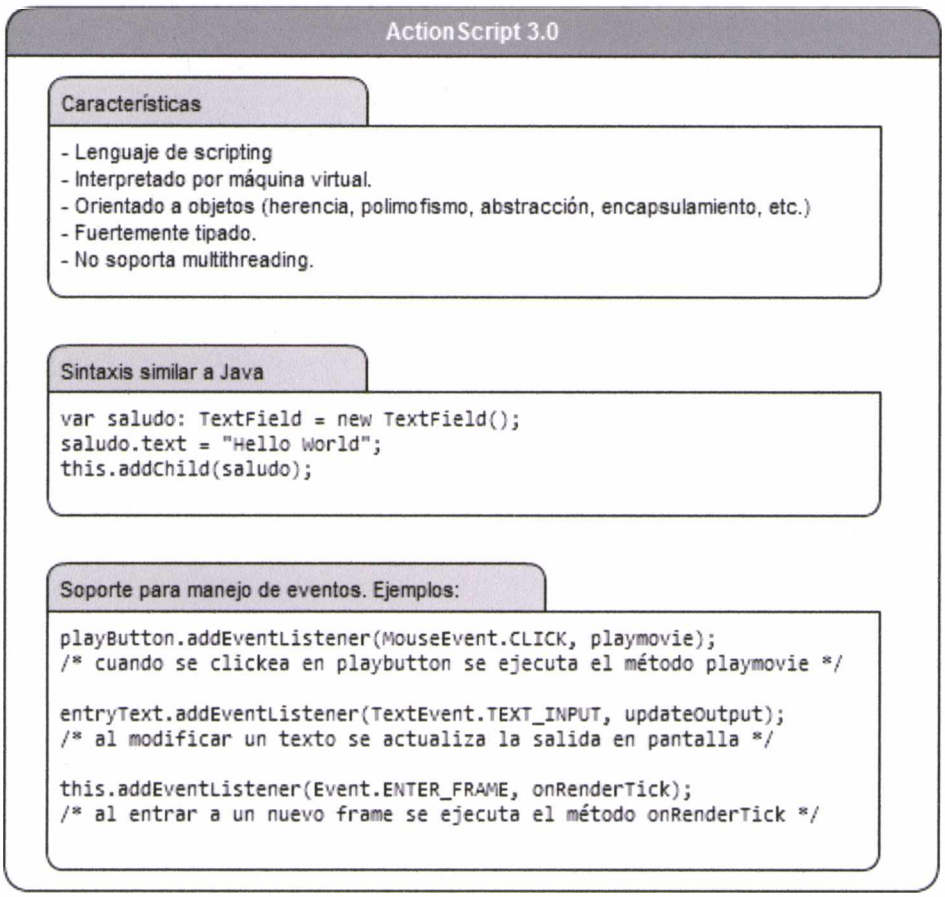


Figura 2.11. Resumen de Actionscript 3.0

Para implementar la solución propuesta se utilizó el framework FLARToolkit [11, 12], una versión adaptada para ActionScript de la librería ARToolkit, en combinación con *Papervision3D*, un motor de renderizado de gráficos en tres dimensiones para ActionScript.

2.3 Adobe AIR

Adobe AIR [5] es una tecnología versátil que permite crear, a partir de lenguajes orientados a la programación de RIA (aplicaciones de internet enriquecidas), como Flash, ActionScript, HTML o JavaScript, ejecutables similares a los programas tradicionales de escritorio.

Existe una importante diferencia entre las RIA y las aplicaciones generadas con AIR, dado que en las primeras, al ser implementadas en un navegador, no se requiere de una instalación, mientras que en las segundas se requiere el empaquetamiento, la firma digital, y la instalación en el sistema de archivos del usuario. Esto proporciona acceso al sistema de archivos y al almacenamiento local, mientras que las aplicaciones basadas en navegador son más limitadas en cuanto a dónde y cómo se almacena y se accede a los datos guardados. En la mayor parte de los casos, las RIAs almacenan datos del usuario en sus propios servidores, pero la posibilidad de trabajar con datos en el sistema de archivos local del usuario le otorga a Adobe AIR mayor flexibilidad.

2.4 FLARToolkit

FLARToolkit [9, 10] es un framework que brinda el marco adecuado para crear aplicaciones de RA con un marker y una cámara web. Se trata de una versión portada a AS3 de ARToolkit, originalmente codificado en C++.

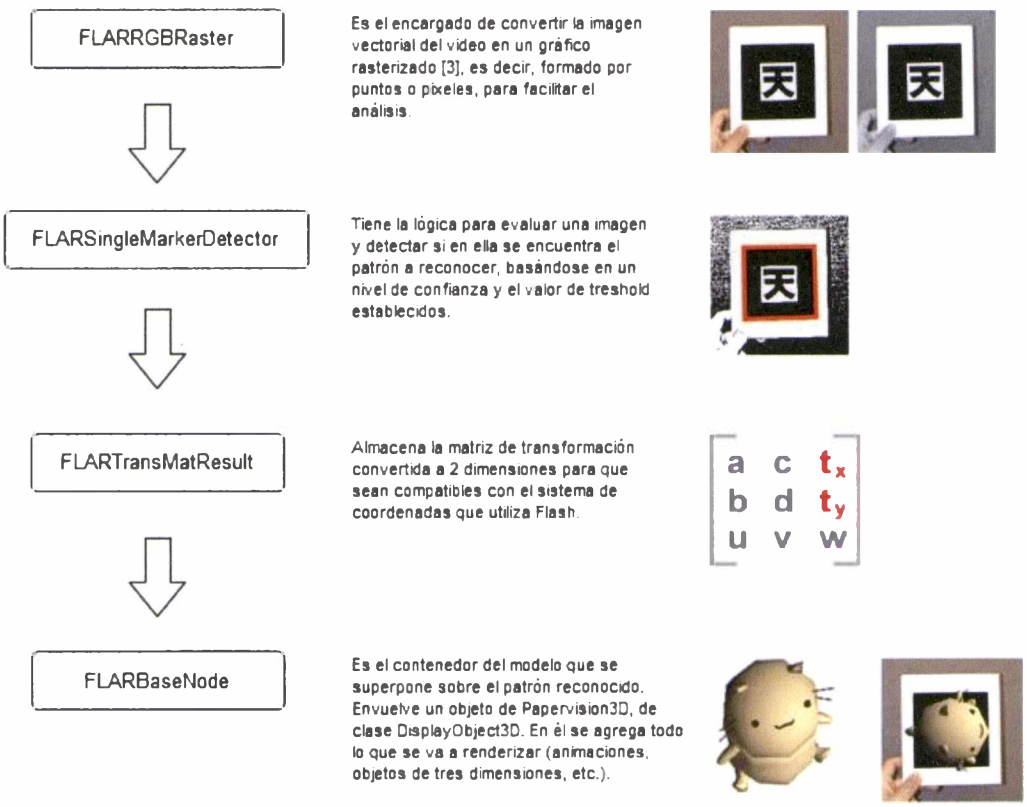


Figura 2.12. Las clases mas relevantes de FLARToolit

A partir de un archivo de imagen y una captura de video, aplica algoritmos para detectar esa imagen dentro del video, otorgando las coordenadas del plano tridimensional que forma.

Al ser combinado con algún motor de renderizado 3D (Papervision3d, Away3d, Sandy, Alternativa3d) se puede *aumentar la realidad* manteniendo siempre una consistencia geométrica.

Además de las clases mencionadas en la Figura 2.12, FLARToolkit provee un método crucial para el manejo de eventos: el método *onRenderTick()*.Es disparado por el evento EVENT.ENTER_FRAME, es decir que se ejecuta cada vez que la aplicación cambia de *frame* (si se setean 30FPS en la aplicación, el método se ejecutará 30 veces por segundo). Al crear una aplicación de RA con FLARToolkit *es necesario* redefinir éste método.

2.5 Papervision3d

Flash no tiene funcionalidad 3D *per se*, pero permite el manejo de líneas y puntos, lo cual es la base de cualquier sistema de gráficos sobre un objeto bidimensional como una pantalla. Usando esos puntos y líneas se desarrolló la API de Papervision3d[6] (PPV), una librería con el objetivo de hacer render dinámico sobre Flash y simplificar la tarea de crear un escenario con objetos tridimensionales.

2.5.1 Composición de figuras

En la geometría de 3 dimensiones, cada objeto se compone de un conjunto de vértices, y a partir de ellos se forman triángulos.

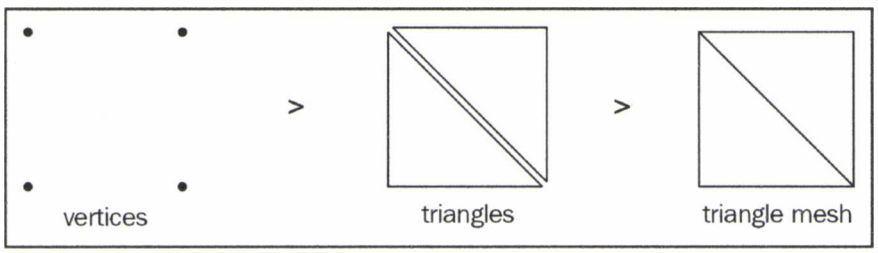


Figura 2.13. Vértices y triángulos

Si agrupamos un conjunto de triángulos podemos crear todo tipo de figuras, de dos o tres dimensiones y de cualquier tamaño. Papervision3D brinda una serie de primitivas básicas que pueden ser utilizadas por separado o en la composición de figuras más complejas.

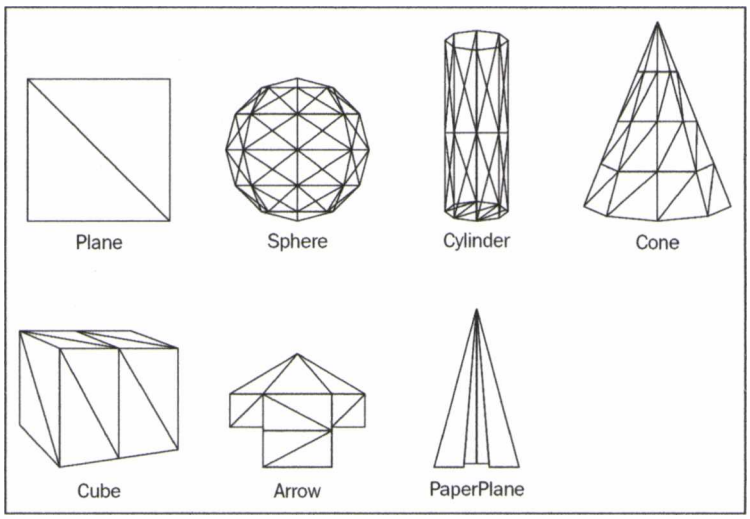


Figura 2.14 Formas básicas en Papervision3D a partir de triángulos

2.5.2 Soporte de modelos externos

Además de las figuras que otorga la librería, existe la posibilidad de importar modelos 3d creados con otras aplicaciones (Blender, 3D Studio Max y Google Sketchup son ejemplos de estas herrramientas). Para poder importar correctamente los modelos con sus texturas, utiliza el estandar COLLADA (COLLAborative Design Activity). Este formato permite intercambiar modelos entre diferentes aplicaciones, mediante la definición de un archivo XML con la información necesaria para su correcta interpretación.

2.5.3 Cámara, escena y proyección

Flash, al igual que PPV, hacen uso del sistema de coordenadas cartesianas [13, 14]. En Flash, un objeto puede ser mostrado en la pantalla indicando sus valores en los ejes X e Y.

Al incorporar objetos 3D en un escenario que originalmente solo maneja dos dimensiones, surge una diferencia obvia que es la aparición de un nuevo eje de coordenadas: el eje Z. Esto logra darnos una sensación de profundidad, como si los objetos pudieran acercarse o alejarse de nuestro punto de vista.

Por ejemplo, cuando un objeto 3D que representa una caja cambia el valor de su eje Z de 0 a 300 (alejándose) PPV lo único que hace es disminuir su tamaño manteniendo la relación de aspecto entre los tres ejes.

Para poder dibujar esos objetos 3D en una pantalla de 2D, es necesario hacer un mapeo de los mismos, lo que se denomina *proyección*. PPV toma las coordenadas de cada vértice y las proyecta en un plano de 2 dimensiones, calculando cuál debería ser la posición de cada vértice en la pantalla. Al tener

los vértices ubicados en perspectiva, se crean así los triángulos que finalmente formarán los objetos deseados.

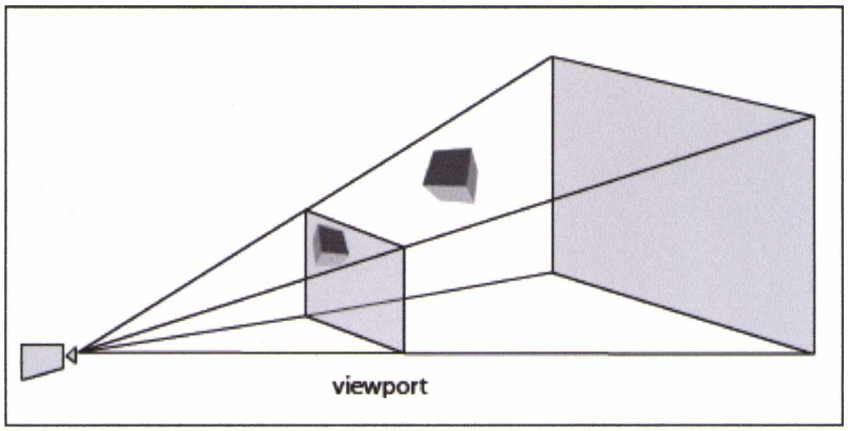


Figura 2.15. El mapeo de objetos de 3 a 2 dimensiones

La proyección se realiza sobre un elemento denominado ***viewport***, que puede definirse como la "ventana" a través de la cual la cámara ve lo que sucede en la escena. Por defecto tendrá el mismo tamaño de la escena, pero puede ser redimensionado, e incluso pueden convivir más de uno en una misma pantalla (por ejemplo para juegos multiusuario).

De esta manera, PPV resulta un complemento ideal para enriquecer a ActionScript, generando nuevos componentes mediante las bases de la computación de gráficos 3D.

Nota: Las figuras **2.2**, **2.13**, **2.14** y **2.15** fueron utilizadas con permiso de los autores y responsables de la publicación del libro *Papervision3D Essentials (Packt Publishing)*.

Capítulo 3

Administración de múltiples cámaras

3.1 Los límites de la Realidad Aumentada

En todas las experiencias con aplicaciones de RA se hace visible una importante limitación. Cuando se utilizan dispositivos de mano, como PDAs y celulares, o dispositivos montados en la cabeza, el usuario puede moverse alrededor del marker e ir percibiendo distintos ángulos del modelo, de manera de sentir que se encuentra realmente frente a un objeto palpable.

Un claro ejemplo de esta situación se expuso en un estudio realizado en 2005 [10], en el cual varios grupos de niños participaban en un juego colaborativo. Cada niño se encontraba equipado con una PDA con GPS y WiFi que les permitía saber la ubicación física de varias marcas (patrones) en un campo de futbol. Al aproximarse a esas marcas, sus PDAs aumentaban la realidad mostrando en sus pantallas figuras de diversos animales. Al ser la PDA un dispositivo portátil, cada niño podía rodear la marca apuntándola con su PDA y tener una visión completa de todas las caras del animal.

En las aplicaciones *desktop* esto es más complicado de lograr dado que es necesario reubicar la cámara web o rotar el marker, acercarlo y alejarlo para percibir mejor todas las caras del modelo o sus detalles. Se genera entonces una degradación en la performance y en la calidad de la experiencia del usuario (QoE), quien debe participar de una manera más activa.

Por ejemplo, en una aplicación comercial de una empresa de aerolíneas (Figura 3.1), se dibuja un modelo animado compuesto por un avión y el planeta tierra. El avión realiza un vuelo dejando pasajeros alrededor del planeta. Cuando el avión se sitúa detrás del globo terráqueo, queda fuera del punto de vista de la cámara. Luego sigue su recorrido y nuevamente se sitúa frente a la cámara. Esto quiere decir que cuando el avión se detiene en los países de la cara no visible del globo terráqueo, se hace imposible percibirlo.

3.2 Incorporando N cámaras

La limitación planteada es el disparador para la hipótesis de trabajo: colocar más de una cámara web, en distintos puntos del espacio, apuntando al marker simultáneamente. Se espera que esta variación otorgue una visión mucho mas completa y realista de la aplicación, sin tener que mover una

única cámara o el marker para poder ver que sucede desde otros ángulos, como por ejemplo una vista de la cara posterior, una vista panorámica desde arriba, los laterales, etc.

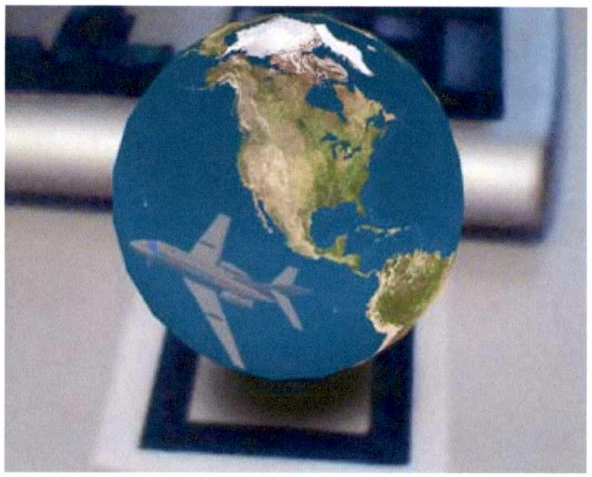


Figura 3.1. La aplicación de la aerolínea en acción

En la aplicación de la aerolínea, tener una segunda cámara enfrentada que también dirija su foco hacia el marker, permitiría mostrar lo que la primera no ve. Cuando el avión se va hacia aquellos países que están "detrás" de la imagen que percibe la primer cámara, es el momento en que la segunda debería entrar en acción. De este modo el avión siempre sería registrado por alguna de las dos, quienes complementándose una a otra mostrarían el recorrido completo.

Pero, como sucede con toda posible solución, se generan en torno a ella ciertos interrogantes:

¿Es posible escalar una aplicación de RA a más de una cámara?
¿Hay una manera de administrar todos los videos en tiempo real?
¿Se verá afectada la performance de la aplicación?

Al igual que en una película, tener varias "tomas" de lo que esta sucediendo da la posibilidad de ver una misma escena desde distintos ángulos. De aquí se desprende que se debe determinar como manejar esas tomas, y adquirir un rol de "director de cámara".

Haciendo un paralelismo con la filmación de una película, se puede decir que un director de cine se encuentra ante la misma encrucijada, pero existe una sutil diferencia, y está dada porque el director *sabe que es lo que va a ocurrir en cada instante de tiempo*. La existencia de un guión y un orden de las acciones que se van a suceder, le permite escoger en un momento dado, cual es la toma donde mejor se aprecia la situación que esta ocurriendo. Es posible entonces definir una secuencia de las tomas de cámara.

En el caso de este trabajo, esto no se da así, porque si se setea una secuencia fija de tomas y posteriormente se sustituye el modelo animado por otro que se comporte de manera distinta, la

secuencia podría no adaptarse correctamente. De hecho, seguramente no se adaptará y el resultado serán tomas inapropiadas en todo momento.

De esto se deduce que el usuario debe poder elegir según su conveniencia, la estrategia de cámaras que más se adecue a la aplicación que quiere desarrollar. Es decir que no habrá un esquema único, sino varios [18, 19], que requerirán mayor o menor participación del usuario, que tendrán mayor o menor escalabilidad, consumo de recursos, facilidad de adaptación, etcétera.

3.3 El diseño de la solución

A continuación se explicará el diseño de la solución propuesta. Se trata de un conjunto de clases que permiten al desarrollador generar una aplicación de RA con n cámaras y definir alguno de los esquemas disponibles para manejar sus flujos de video.

El siguiente diagrama presenta las clases más relevantes de la librería:

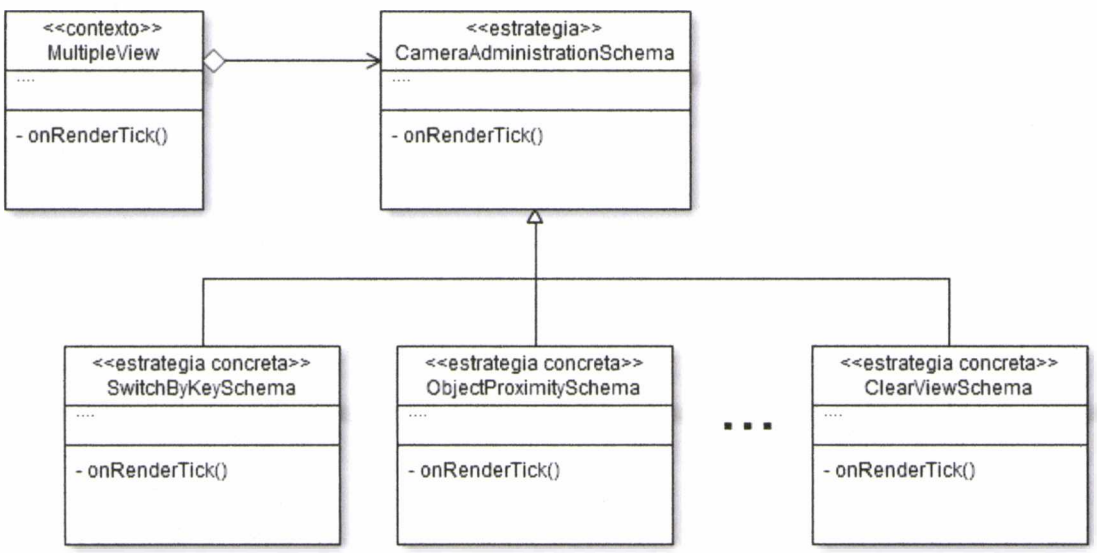


Figura 3.2. Diagrama de clases.

La **clase principal** se denomina *MultipleView*. Ella se encarga de hacer todas las inicializaciones necesarias para que comience la ejecución. Entre esas inicializaciones se incluyen:

- Los parámetros de la cámara y la imagen del marker.
- Las cámaras y videos.
- El esquema elegido
- El modelo a renderizar

Esta clase redefine el método *onRenderTick()*, agregando en él el comportamiento de la aplicación. De este modo, en cada *frame,* el esquema escogido se encarga de evaluar la información proveniente de las cámaras y, en base a ello, setear la cámara adecuada (o dejar la que ya estaba seleccionada).

El **modelo 3D** está envuelto por la clase *Model3D.* Esto facilita la incorporación de modelos desde archivos externos (Collada, Sketchup, etc.) fijos o animados, como así también la creación de los mismos mediante Papervision3D. A la hora de crear el objeto sólo se necesita generar una subclase que extienda de *Model3D* y esta heredará las facilidades para renderizar el objeto en pantalla. Puede estar formado por un solo objeto o por una composición de ellos. En el caso de tratarse de una composición, cada objeto particular debe ser envuelto por una clase (para encapsular el manejo del recurso) y luego *Model3D* se plantea como un objeto compuesto por instancias de esas clases.

Para la relación entre la aplicación y los esquemas se aplica el patrón **Strategy**. Este patrón define una familia de algoritmos, encapsula cada uno de ellos y los hace intercambiables [8]. Sus participantes son: el *contexto*, la *estrategia abstracta* y una serie de *estrategias concretas*.

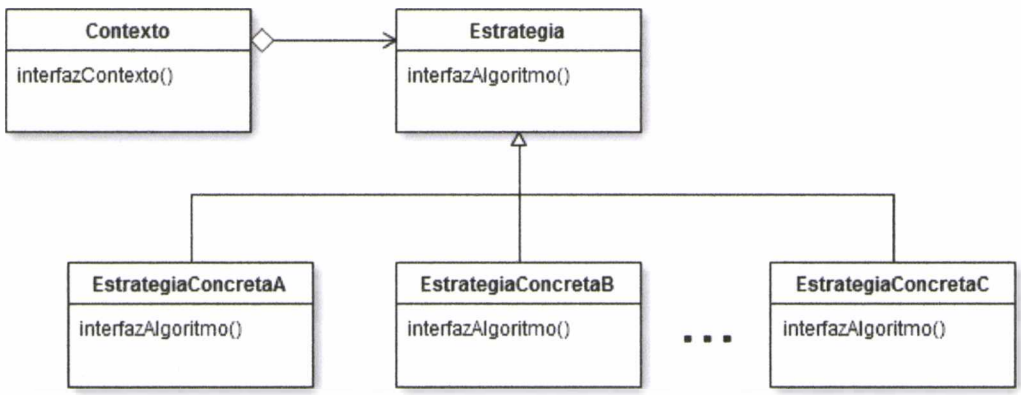


Figura 3.3. El patrón Strategy.

El contexto redirige sus peticiones a su estrategia. La instancia de estrategia que posee el contexto es de alguna de las subclases concretas, que implementan el método de interés.

Como se puede apreciar en el diagrama de la Figura 3.2, la clase *MultipleView* representa al contexto y *CameraAdministrationSchema* es la estrategia abstracta. Cada esquema que hereda de *CameraAdministrationSchema* es una posible estrategia concreta. Esto permite desacoplar el esquema de la clase principal para poder intercambiarlos fácilmente y también incorporar otros nuevos.

Existe además un punto de entrada a la aplicación, que es la clase *Main,* en la que se deben especificar la cantidad de cámaras disponibles y el modelo a utilizar. Ella se encarga de instanciar un objeto de clase *MultipleView* y darle comienzo a la aplicación.

En un momento determinado, solo un esquema puede ser instanciado para ser utilizado por la aplicación. La clase abstracta *CameraAdministrationSchema* define métodos comunes a todos los esquemas y uno en particular que cada esquema redefinirá de acuerdo a su comportamiento:

```
onRenderTick():

Es aquí donde se concentra  la lógica del esquema. Es el método clave, donde se decide que cámara va
a estar activa durante el próximo frame.
```

Si se quisiera agregar uno nuevo bastaría con definir una clase que extienda a *CameraAdministrationSchema* e implemente dicho método.

3.4 Adaptando FLARToolkit

La librería FLARToolkit está diseñada para funcionar con una sola webcam, es decir que sólo se cuenta con una instancia de la clase que envuelve la cámara, una instancia de la clase que maneja el video y una para la clase que envuelve al *raster*, que evaluará si se detecta o no el marker.

Para poder tener varios flujos de video es necesario manejar *arrays* con instancias de todas esas clases, es decir, un arreglo de **n** instancias de cámara, n de video y **n** *rasters*. A continuación se muestra el fragmento de código descrito anteriormente:

```
1. this.camerasArray = new Array();
2. this.videosArray = new Array();
3. this.bitmapsArray = new Array();
4. this.rastersArray = new Array();
5.
6. for (var i:int = 0; i < this.camerasQuantity; i++ ) {
7.     trace("Inicializando una nueva cámara..");
8.     this.camerasArray[i] = Camera.getCamera(String(i));
9.     this.camerasArray[i].setMode(anchoImagen, altoImagen, 30);
10.    this.videosArray[i] = new Video(anchoImagen, altoImagen);
11.    this.videosArray[i].attachCamera(this.camerasArray[i]);
12.    this.bitmapsArray[i] = new BitmapData(anchoImagen, altoImagen, true);
13.    this.bitmapsArray[i].draw(this.videosArray[i]);
14.    this.rastersArray[i] = new FLARRgbRaster_BitmapData(this.bitmapsArray[i]);
15.    }
16. }
```

La instrucción de la línea 7, `Camera.getCamera(indice)`, recupera la primer cámara que reconoce el sistema [9]. Si llamamos al método sin parámetros, éste devuelve la cámara por defecto. En caso contrario, chequea los puertos usb por posibles webcam y por último, de ser una computadora portátil, devuelve la cámara incorporada si la hubiese.

A continuación, se crea un flujo de video a partir de la cámara que se obtuvo: cuando se ejecuta la línea 10 el reproductor Flash muestra siempre una advertencia y pide permiso al usuario para activar la cámara y comenzar la filmación. Esto es parte de un método de protección de recursos que es necesario cuando se ejecutan aplicaciones en exploradores web, pero al utilizar la plataforma Adobe AIR, esta comprobación se omite y se accede directamente a los recursos.

Al ejecutarse la aplicación, todas las webcam se inicializan proporcionando un flujo de video constante. En las sucesivas entradas al método *onRenderTick()* lo que se hace es decidir cual de todos esos flujos disponibles va a ser el que se renderizará. Esta decisión no está librada al azar, sino que constituye un punto muy importante en este trabajo.

3.5 Esquemas de coordinación que se plantean

A continuación se explican los esquemas que fueron implementados y otros posibles, para un trabajo a futuro.

3.5.1 Esquema manual

El caso más simple que se puede pensar es un esquema que va rotando las cámaras manualmente, sin ser influido por ninguna variable externa.

La orden de cambiar de cámara puede manifestarse con un evento que la aplicación pueda capturar, como por ejemplo, presionar una tecla, un click del mouse, un botón en un control remoto, o un comando por voz, de ser posible su detección. Se puede asignar una tecla única para cambiar secuencialmente o bien un número a cada cámara para seleccionar una en particular.

Figura 3.4. En el esquema manual, las cámaras cambian secuencialmente, similar a una arquitectura token ring.

```
1.  public function keyDownListener(e:KeyboardEvent):void{
2.
3.  // en este caso, se captura el evento de presionar cualquier tecla
4.
5.  // saca la cámara activa
6.  this.removeChild(this.videosArray[this.activeCameraIndex]);
7.
8.  // obtiene el índice de la próxima cámara
9.  this.activeCameraIndex = (this.activeCameraIndex + 1) %
                            this.camerasQuantity;
10. // agrega la nueva cámara activa
11. this.addChildAt(this.videosArray[this.activeCameraIndex +1],
                                  this.getChildIndex(this.viewport));
12. }
```

3.5.2 Esquema de quantum o slice de tiempo

En este esquema, cuando la aplicación comienza su ejecución se pone en marcha un reloj o timer que es chqueado en cada iteración; al llegar a cierto valor (por ejemplo 1 minuto), se cede el control a la siguiente cámara y se resetea el timer para comenzar a contar nuevamente.

Esta solución también resulta simple dado que el único valor que se requiere controlar es el tiempo que debe alcanzar el timer.

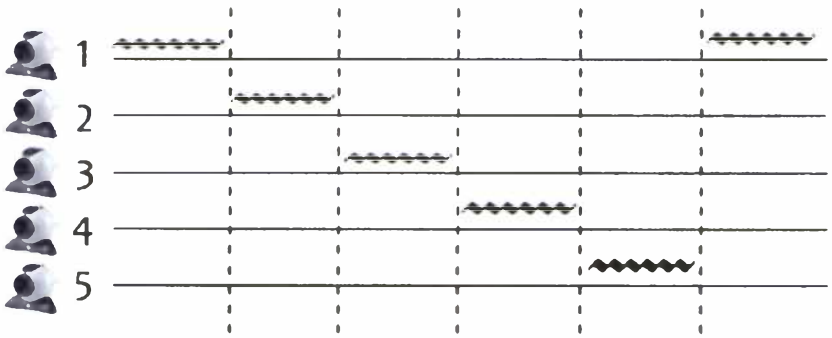


Figura 3.5. Luego de cumplirse su slice de tiempo, la cámara activa delega el renderizado a la siguiente.

```
1.  override public function onRenderTick(obPosition:Number3D):void{
2.
3.    // se chequea el tiempo de la cámara actual
4.    if (this.timer > QUANTUM_TIME){
5.
6.       this.timer.reset();
7.
8.      // saca la cámara activa
9.      this.removeChild(this.videosArray[this.activeCameraIndex]);
10.
11.     // obtiene el índice de la próxima cámara
12.     this.activeCameraIndex = (this.activeCameraIndex + 1) %
                                 this.camerasQuantity;
13.
14.     // agrega la nueva cámara activa
15.     this.addChildAt(this.videosArray[this.activeCameraIndex +1],
                                        this.getChildIndex(this.viewport));
16.    }
17. }
```

3.5.3 Esquema de proximidad al modelo

Si se trata de un modelo animado, por ejemplo un avión que realiza un vuelo o un personaje que se mueve por el espacio, en un momento éste podría alejarse mucho de la cámara y no se tendría una buena percepción de lo que está sucediendo. En ese caso resultaría útil cambiar *automáticamente* a

una cámara que se encuentre más próxima al modelo. A continuación se explica cómo esto puede llevarse a cabo.

En el capítulo anterior se explicó cómo a partir de la matriz de transformación, realizando una serie de cálculos, es posible obtener la distancia entre la cámara y el marker. La particularidad de este esquema es que permite individualizar un objeto en movimiento dentro del modelo 3D para simular que esta siendo "seguido" por las cámaras. Por ejemplo, las cámaras podrían ir cambiando a medida que el avión se aproxima a ellas. Para esto es necesario obtener las coordenadas del avión y sumarlas a las del marker, dado que éste esta estático en la escena y lo que se mueve es el avión que constituye una parte del modelo.

Si se tienen n cámaras, en cada frame se deben calcular las posiciones de todas ellas para posteriormente compararlas y escoger aquella que sea menor.

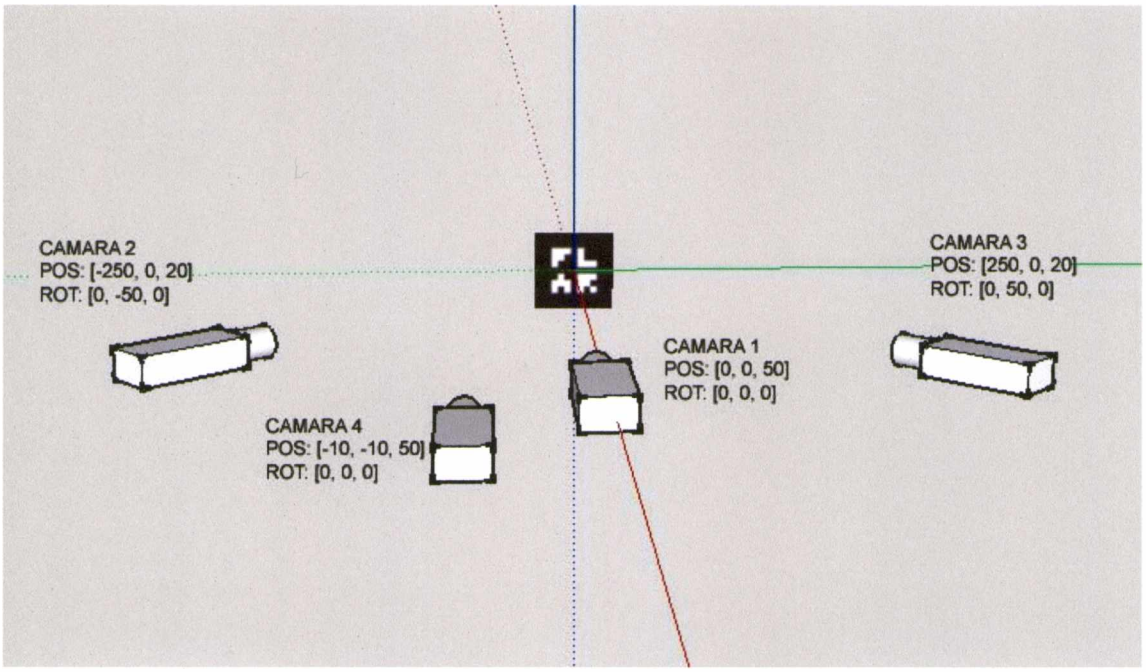


Figura 3.6. 4 cámaras detectando el mismo marker

Cada vez que se calcula una distancia se chequea cual es el mínimo actual. Si la nueva distancia es menor que la que se tenía hasta el momento, se actualiza el valor mínimo y se actualiza el índice de esa cámara. Por lo tanto, si se tienen 4 cámaras con índices 0, 1, 2 y 3 y con valores de distancias de 50, 82.3, 82.3 y 65.7, el algoritmo comparará todos estos valores:

50	82.3	82.3	65.7
0	1	2	3

En este caso, la distancia mínima es de 50mm, de modo que se escoge el índice 0 y se setea la cámara con ese índice como la activa en ese frame.

```
1. override public function onRenderTick(obPosition:Number3D):void{
2.
3.    // Inicializa la posición de la cámara (fija)
4.    if (this.positionsArray[this.activeCameraIndex] == null) {
5.       this.positionsArray[this.activeCameraIndex] = obPosition;
6.    }
7.
8.    // Calcula las distancias entre la cámara y el modelo
9.    for (var i:int = 0; i < this.camerasQuantity; i++ ) {
10.      if (this.positionsArray[i] != null) {
11.         var position:Number3D = this.positionsArray[i];
12.         var distCam_i:Number = Math.sqrt(Math.pow((position.x - this.modelo.x),2)
                                        + Math.pow((position.y - this.modelo.y),2) +
                                          Math.pow((position.z + this.modelo.z),2));
13.         this.distancesArray[i] = distCam_i;
14.      }
15.   }
16.
17.   // la función getClosestCameraIndex() busca la mínima distancia
18.   var indiceDelMasCercano:int = this.getClosestCameraIndex();
19.   if (indiceDelMasCercano != this.activeCameraIndex) {
20.      this.mv.removeChild(this.videosArray[this.activeCameraIndex]);
21.      this.mv.addChildAt(this.videosArray[indiceDelMasCercano],
                                this.mv.getChildIndex(this.viewport));
22.      this.activeCameraIndex = indiceDelMasCercano;
23.      }
24.    }
```

Como se puede suponer, este esquema tiene en cuenta muchas más variables y realiza cálculos más costosos que los anteriores. A continuación se lista en detalle el tiempo de ejecución de este esquema:

- *Inicializar la posición de la cámara* = constante (solo para la primer vez).
- *Calcular la distancia de las cámaras* = n + constante (n = # de cámaras)
- *Obtener el mínimo y setearlo como activo* = n + constante

T(n) = 2n + 3c => O(n)

El **orden de complejidad** del algoritmo es **lineal**, esto quiere decir que depende de la cantidad de cámaras. Mientras más se agregan, se necesitan hacer más cálculos y comparaciones entre las distancias, por lo tanto se genera un decremento importante en la performance, reduciendo la cantidad de FPS.

3.5.4 Esquema de proximidad a un evento

Haciendo uso de los mismos cálculos mencionados en el esquema anterior, relacionados con la distancia entre el marker y las cámaras, este esquema sugiere algo similar: se selecciona la cámara que está más próxima a un evento que sucede en la escena.
Por ejemplo si se tiene un conjunto de partículas que se mueven y reaccionan entre sí, cuando algunas de ellas se juntan podrían "calentarse" y cambiar su estado, producir una explosión, cambiar de color, etc.

En este caso se trabaja con un modelo compuesto por varios objetos (partículas). Si estos objetos se mueven aleatoriamente, en cada iteración podrían tener una posición distinta, y aproximarse o alejarse entre si. Cuando ocurra algún evento de interés, por ejemplo, que la distancia entre 10 de ellos sea menor que 1 cm, se produce un agrupamiento de objetos, que puede generar un suceso nuevo (por ejemplo, la composición de un nuevo objeto). Al dispararse este suceso, este esquema calcula cuál es la cámara que está más próxima a él y automáticamente es seleccionada para mostrar la escena.

3.5.5 Esquema de mejor perspectiva (clearview).

En la siguiente imagen se puede identificar la cámara y una serie de objetos, de los cuales el avión superior se encuentra fuera del ángulo de vista de la cámara; el planeta Tierra se encuentra dentro y el avión inferior, parcialmente dentro.

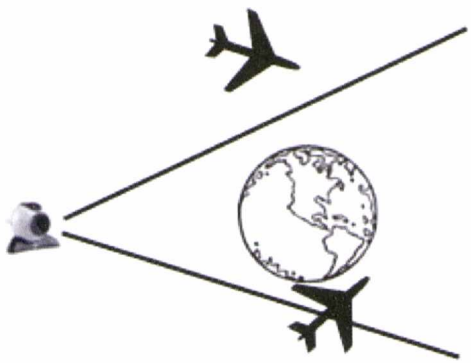


Figura 3.7. Distintos objetos: dentro, fuera y parcialmente dentro del ángulo de vista de la cámara

Culling es el proceso de identificar qué objetos se encuentran dentro y fuera del ángulo de vista de la cámara. Papervision3D provee distintos tipos de *culling*, uno de ellos es el denominado *frustum*

culling, que evita el renderizado de objetos que están completamente fuera de nuestro punto de vista, en este caso, los de color gris claro.

Mediante este proceso se puede consultar a un objeto para saber si está siendo afectado por el culling, de manera de cambiar a una cámara que en ese momento tenga una vista completa y clara del mismo.

```
1. override public function onRenderTick(model:Model3D):void{
2.      // si el modelo no se percibe en su totalidad, la propiedad culled
       del mismo
3.     // se encuentra en false
4.     if (model.culled){
5.        // saca la cámara activa
6.        this.removeChild(this.videosArray[this.activeCameraIndex]);
7.        // obtiene el índice de la próxima cámara
8.        this.activeCameraIndex = (this.activeCameraIndex + 1) %
       this.camerasQuantity;
9.        // agrega la nueva cámara activa
10.       this.addChildAt(this.videosArray[this.activeCameraIndex +1],
                  this.getChildIndex(this.viewport));
11.       }
12.}
```

Este esquema presenta una contra importante: si durante un lapso de tiempo ninguna cámara tiene una vista completa del objeto, se realiza un cambio de cámara en cada frame hasta que se pueda detectar una vista clara. Si nunca se presenta una buena perspectiva del modelo, el esquema continua haciendo infinitos cambios de cámara. Esta situación puede compararse con la hiperpaginación [20] en un sistema operativo, en el que los ciclos del procesador se invierten en llevar y traer páginas entre la memoria principal y la virtual y no en el procesamiento de la aplicación.

Capítulo 4

Caso de estudio

4.1 Introducción

En este capítulo será descrito un caso de estudio en el cuál se utiliza la librería desarrollada. En principio se mencionarán las modificaciones realizadas en las clases de la librería para poder adaptar una aplicación de una sola cámara. Luego se enumerarán los pasos que se suceden cuando se pone en marcha la aplicación. Por ultimo se detallará como es el funcionamiento para cuatro de los esquemas mencionados, destacando las ventajas de cada uno por sobre el resto.

4.2 Adaptando una aplicación convencional

Para el caso de estudio se tomó una aplicación convencional de Realidad Aumentada, que funciona con una sola cámara. Se trata de una aplicación *desktop* en la cuál se renderiza un modelo en 3 dimensiones de una abeja que vuela alrededor de una flor en busca de polen.

4.2.1 El marker

Se utilizó un marker cuadrado de 8cm de lado, blanco y negro y asimétrico. El mismo se muestra en la figura siguiente.

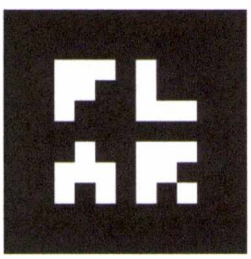


Figura 4.1. El marker del caso de estudio

El diseño del interior posee formas con cierta complejidad para evitar que el algoritmo detector confunda otros objetos del ambiente con posibles coincidencias.

4.2.2 Las cámaras

Para esta evaluación se trabajó con cuatro webcams distintas:

- Una cámara *Genius I-LOOK 300* con buena definición y que se comporta muy bien ante los cambios de luz.
- Una cámara *Logitech Quickcam Express 100*, un modelo limitado y bastante básico, diseñada para aplicaciones de mensajería instantánea.
- Una cámara *Logitech Quickcam Express 150*, muy similar al anterior pero con una definición apenas mejorada.
- Una cámara *Creative* integrada en una laptop, con buena definición pero muy limitada en cuanto a movimientos y respuesta ante cambios lumínicos.

La resolución escogida en la aplicación fue de 320x240 dado que es soportada por las cuatro cámaras y se consigue una mejor relación entre definición y cantidad de cuadros por segundo. Las siguientes fotografías denotan el escenario de trabajo:

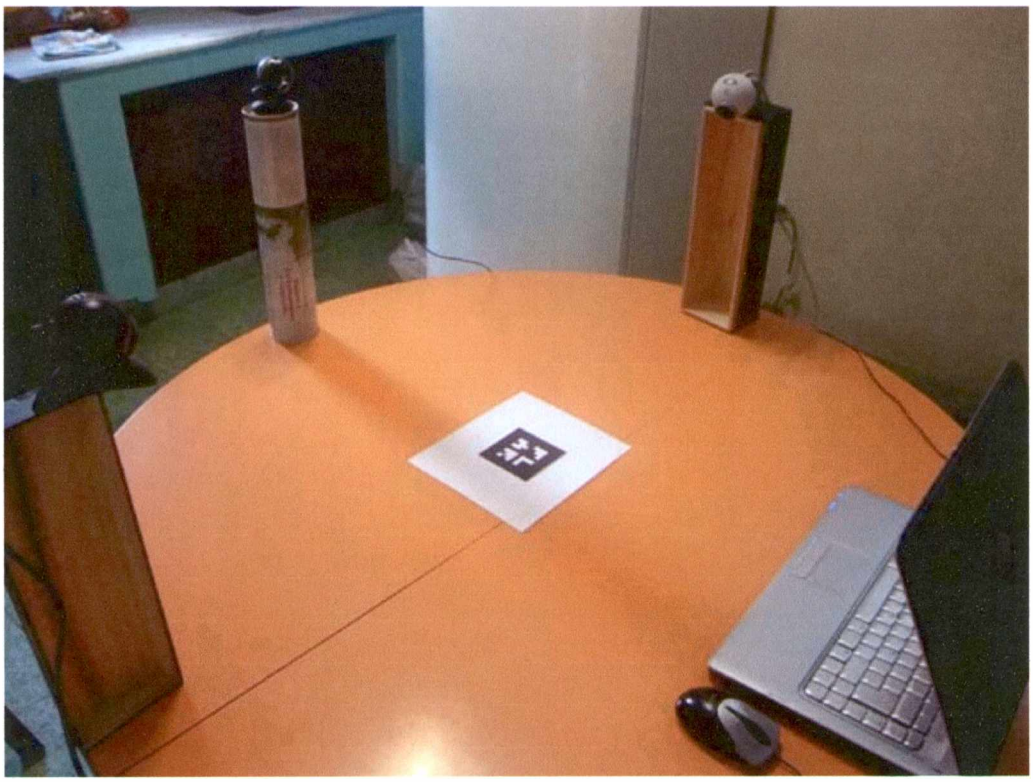


Figura 4.2. El escenario de trabajo

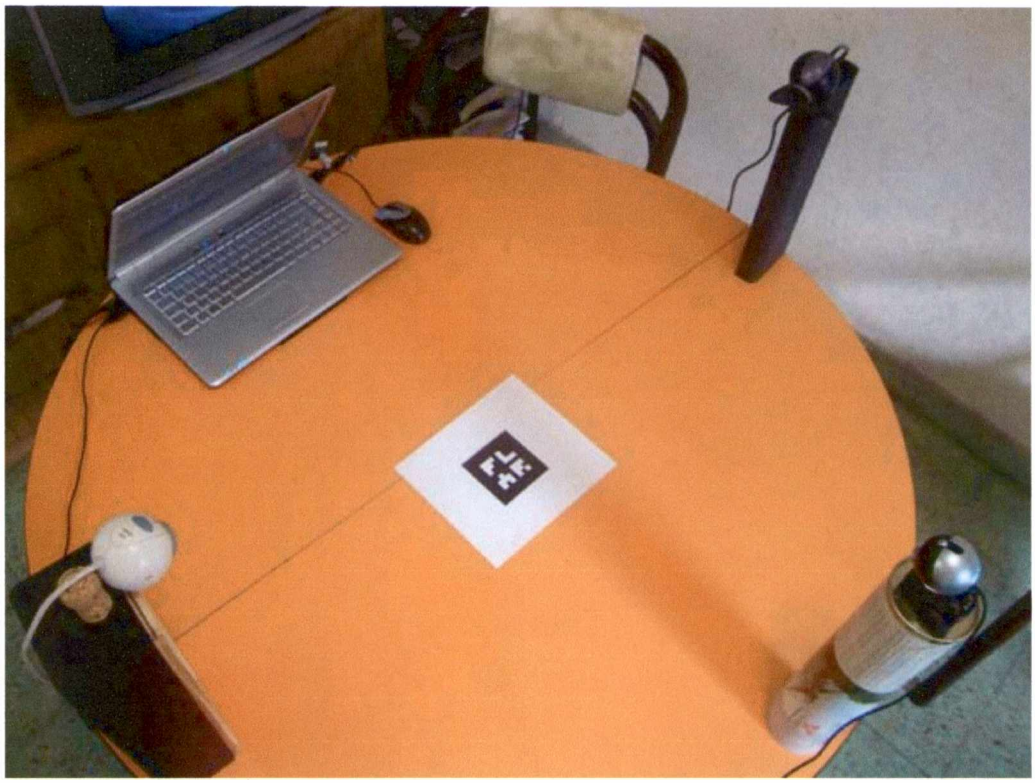


Figura 4.3. El escenario de trabajo desde otro ángulo

Como se puede apreciar en las Figuras 4.2 y 4.3, las cuatro cámaras se disponen una en cada uno de los puntos cardinales del marker, con cierta altura para poder tener una mejor perspectiva del modelo.

4.2.3 El modelo

La animación que se dibuja se compone de dos partes: una flor y una abeja. Como en este caso se enfoca la atención sólo el comportamiento de una de las partes (el vuelo de la abeja), el modelo 3D está representado por una composición de dos objetos. Por un lado se creó la clase *BeeModel* para encapsular el modelo de la abeja (Figura 4.3) y por otro lado, la clase *FlowerModel* para encapsular el modelo de la flor (Figura 4.4). Existe una tercer clase, llamada *Model3D* que instancia una abeja y una flor, y los ubica en las posiciones correctas (Figura 4.5). Esta separación permite individualizar a la abeja dentro del modelo completo.

En cada entrada al método *onRenderTick*(), se delega a la clase *Model3D* la responsabilidad de realizar su animación. Por lo tanto, en *Model3D* se ejecutan las instrucciones para recrear el vuelo de la abeja (en cada iteración gira pocos grados y avanza, realizando un recorrido elíptico).

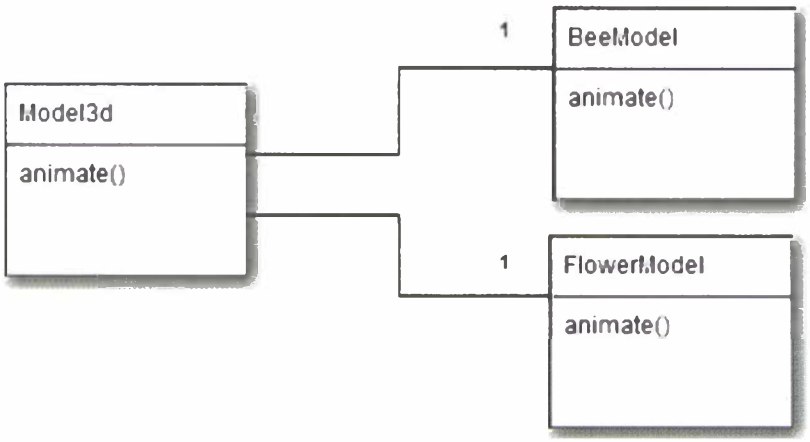


Figura 4.4. El diagrama del modelo

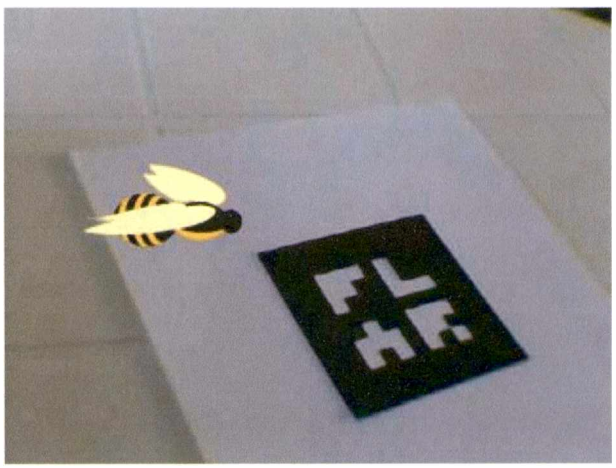


Figura 4.5. La abeja sola.

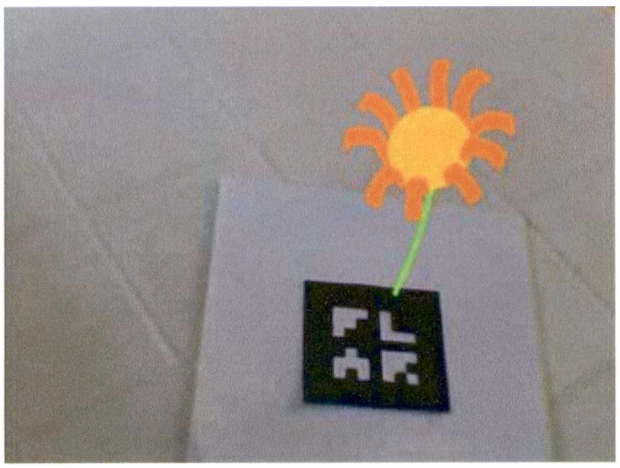


Figura 4.6. La flor sola.

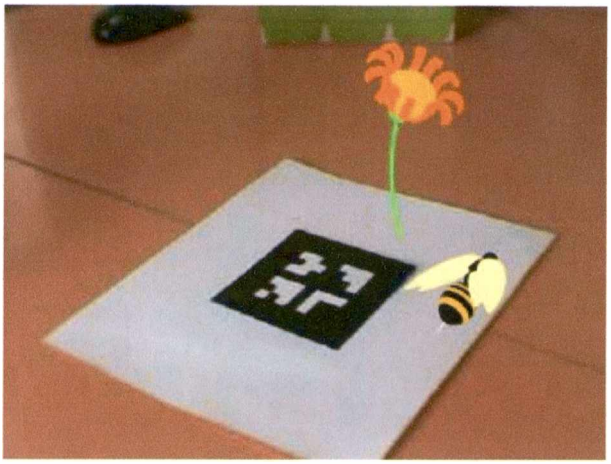


Figura 4.7. El modelo completo con abeja y flor.

4.3 Funcionamiento general

Para ver más claramente como es el funcionamiento del desarrollo propuesto, el siguiente cuadro resume en diversos pasos desde el inicio hasta que la aplicación es finalizada:

Configuración	1. Cuando la aplicación se inicia se debe ingresar el número de cámaras disponibles y seleccionar de una lista qué esquema se utilizará. (Fig. 4.x)
Inicialización	2. Se levantan los parámetros de video, la imagen a reconocer y se realiza la instanciación del esquema y del modelo 3D.
Bucle principal	3. En el momento en que la cámara activa proporciona un flujo de video, se captura un frame para buscar coincidencias con el objetivo. 4. En base al Bitmap relacionado a ese frame, si se detecta el patrón, se obtiene la matriz de transformación. 5. La matriz es aplicada sobre el modelo para mantener la coherencia geométrica. 6. Se anima el modelo. 7. Se delega al esquema la tarea de seleccionar la cámara proporcionándole todos los parámetros que éste requiera. 8. De ser necesario, el esquema *cambia la cámara* devolviendo el flujo de video que corresponde.
Finalización	9. Se termina la captura de video, no se renderiza el modelo y se cierra la aplicación.

Los pasos 3 a 7 se repetirán mientras tengamos al menos una cámara activa, hasta que finalice la aplicación. En cambio, los pasos 1, 2 y 8 solo se ejecutan una vez durante todo el ciclo de vida.

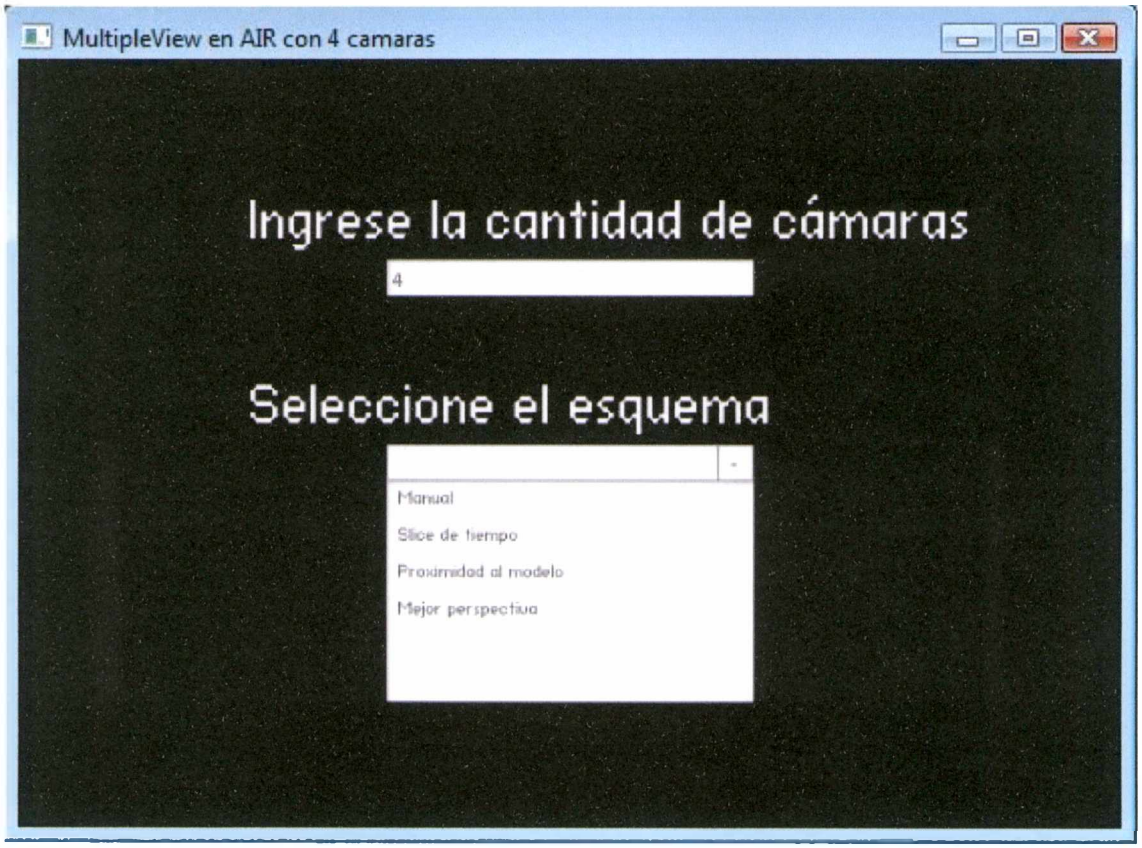


Figura 4.8. Pantalla de configuración de MultipleView

4.4 Evaluación de los esquemas

La aplicación fue evaluada utilizando cuatro de los cinco esquemas planteados. Se escogieron aquellos que presentan significativas diferencias de diseño y comportamiento.

4.4.1 Esquema manual

Cuando comienza la ejecución, se selecciona por defecto la primer cámara inicializada. El orden de inicialización de las cámaras esta ligado a la posición de las cámaras en el array interno de ActionScript (primero todas las conectadas vía USB y por último la cámara integrada).

La configuración del esquema manual, por defecto permite cambiar de cámara disponible presionando la letra **C** (*Camera*) del teclado. Esta tecla ha estado históricamente asociada al cambio de cámara en la mayoría de los juegos de rol y de carreras. Además, cada número del teclado se corresponde con una cámara, según el orden en que se hayan inicializado. Es decir que si tenemos 4 webcams podemos presionar los números de 1 al 4. De esta manera, existe la posibilidad de que el cambio sea secuencial o directo.

Si se realiza el cambio secuencial con la tecla **C**, en ciertas ocasiones se obtienen flujos de videos de cámaras que pueden no tener una buena apreciación del modelo, por lo cual resulta práctico poder setear directamente una en particular. De esta manera no es necesario hacer la recorrida por todas hasta llegar a la deseada.

El cambio de cámara se realiza inmediatamente, dado que el esquema no controla ninguna variable y no requiere inicializaciones.

Si se quisiera cambiar la configuración bastaría con editar la clase *SwitchByKeySchema*, que encapsula su comportamiento, y modificar, por ejemplo, cuál es la tecla que nos permite cambiar la cámara. Para esto, existe un *listener* de los eventos de teclado, donde se pregunta por el código de tecla presionada (67 es el código de la letra **C**) y en caso de ser la que interesa, se procesa el cambio.

```
1. public function keyDownListener(e:KeyboardEvent):void{
2.     if (e.keyCode == 67){
3.           this.changeCamera();
4.     }
...
```

La figura 4.7 muestra la pantalla de la aplicación MultipleView en funcionamiento:

- En la esquina superior izquierda se muestra el flujo de video actual.
- La primer leyenda muestra información del punto en el que se encuentra en cada frame el modelo a seguir.
- La segunda y tercer leyenda muestran respectivamente la posición y rotación de la cámara que está filmando en ese frame (denominada *cámara activa*).
- La última leyenda denota el índice de la cámara activa dentro del arreglo de cámaras.

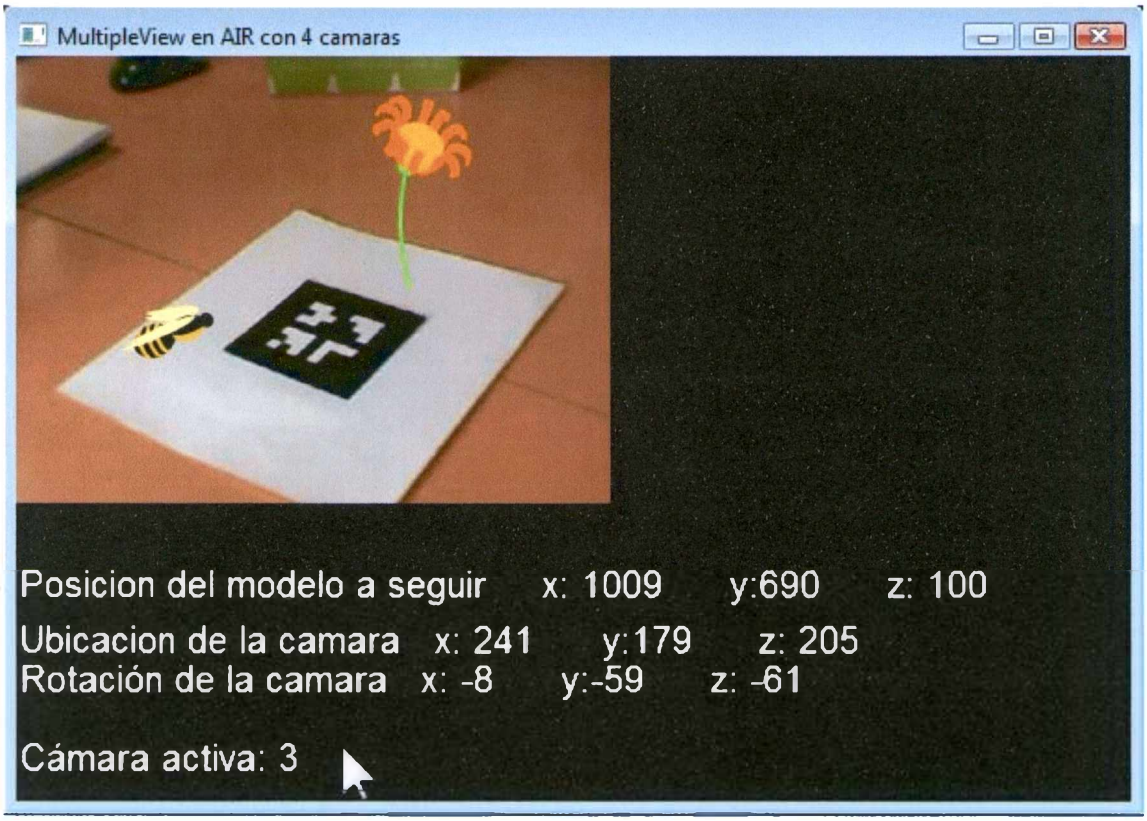


Figura 4.9. El esquema manual en funcionamiento

4.4.2 Esquema de slice de tiempo

El esquema de slice de tiempo es el primero de los *automáticos* a ser analizados. Por automáticos se hace referencia a aquellos esquemas en los que el usuario no tiene participación alguna una vez que la aplicación comienza su ejecución.

Al ser un esquema en el que el cambio de cámaras es únicamente secuencial, presenta el mismo problema que el esquema manual: en un momento determinado puede estar activa una cámara que no tiene una buena perspectiva de lo que está sucediendo.

El espacio de tiempo por el cual cada cámara estará activa solo puede configurarse antes de que se ejecute la aplicación, de modo que no se puede alargar o acortar durante la ejecución. Si se tratase de una aplicación cuyo modelo tiene una animación con tiempos preestablecidos y de igual duración (por ejemplo una abeja que ronda por varias flores y se posa 1 minuto en cada una en búsqueda de polen), este esquema resultaría útil dado que no requiere que el usuario haga un cambio manual de cámara cada cierto tiempo.

El cambio de cámaras se realiza de manera casi inmediata dado que solo se controla la variable que cuenta el tiempo.

4.4.3 Esquema de proximidad al modelo

Este esquema resulta uno de los más interesantes en la aplicación evaluada. Para sacarle el máximo provecho, las cuatro cámaras fueron dispuestas en círculo (Figura 4.8), todas apuntando hacia el marker, donde se renderiza la flor, de manera que el vuelo de la abeja se realiza dentro del espacio delimitado por las cámaras.

En principio el esquema realiza una inicialización de todas las cámaras; para esto setea como activa a cada una de ellas hasta que detecten el marker. La inicialización es necesaria para poder saber cuál es, al momento de iniciar la aplicación, la cámara que se encuentra más próxima al modelo.

Una vez que todo esta dispuesto, la abeja comienza su recorrido elíptico alrededor de la flor. A lo largo de su vuelo, va aproximándose a cada una de las cámaras. En cada nuevo frame, se envía al esquema la información de su ubicación y a partir de allí se calcula su posición dentro del espacio y se cambia a la cámara que en ese momento se encuentra mas cerca de ella. El esquema dota a la aplicación de una cierta inteligencia, simulando que existe un director de cámaras siguiendo a la abeja y tomando sus mejores planos.

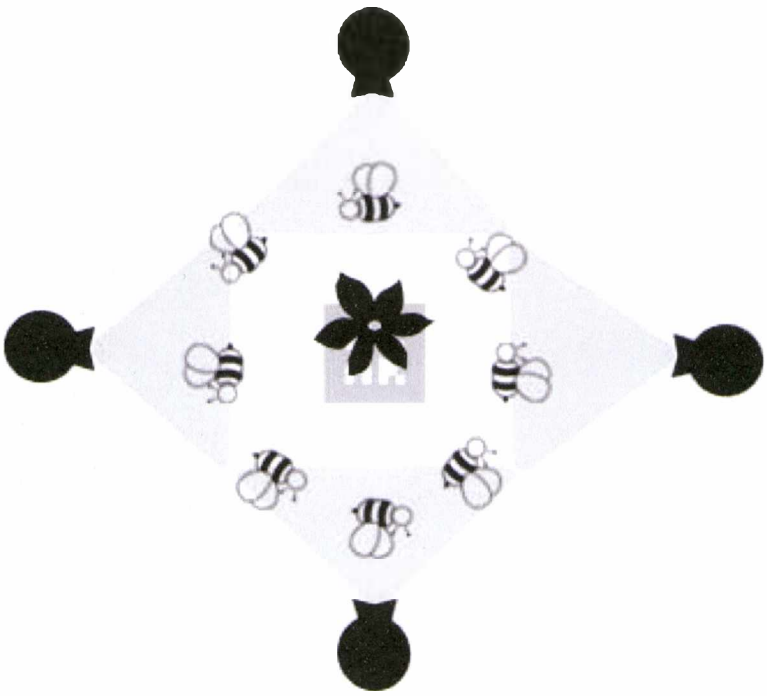


Figura 4.10. Las 4 cámaras rodeando el recorrido de la abeja

Sin embargo, una de las contras que se presenta radica en el procesamiento que se debe hacer en cada iteración. Existe una diferencia de tiempo entre el momento en que la información se envía para ser procesada y su posterior resultado (cuando es necesario cambiar de cámara). Si el vuelo de la

abeja es demasiado rápido se puede dar una situación inconsistente dado que siempre se cambia a una cámara que ya perdió de vista la abeja. Este factor se hace más evidente con cada cámara que se agrega, de manera que este esquema no permite una gran escalabilidad.

4.4.4 Esquema de mejor perspectiva

Al momento de ejecutar la aplicación cuando este esquema es seleccionado, se debe realizar una inicialización, similar al caso anterior, para determinar con qué cámara se comenzará a capturar. Para esto, se chequea cuál es la primer cámara que tiene una apreciación completa de la abeja y se setea como activa. La disposición de las cámaras es la misma que en el esquema de proximidad al modelo.

En cada iteración, y a medida que la abeja avanza en su recorrido, va quedando fuera del campo de visión de las cámaras activas, por lo tanto se vuelve a ejecutar el algoritmo para escoger la cámara apropiada.

La Figura 4.9 muestra que, para esa disposición de pares de cámaras opuestas, en cada momento existen dos cámaras con buena perspectiva de la abeja. Las líneas horizontales delimitan el campo de vista de las cámaras 2 y 4 y las líneas verticales delimitan el campo de las cámaras 1 y 3. Por lo tanto, hay dos potenciales cámaras activas. Para resolver esta situación, el esquema escoge un índice al azar entre los posibles. Es decir que para seguir el recorrido se van alternando entre (1 ó 3) y (2 ó 4).

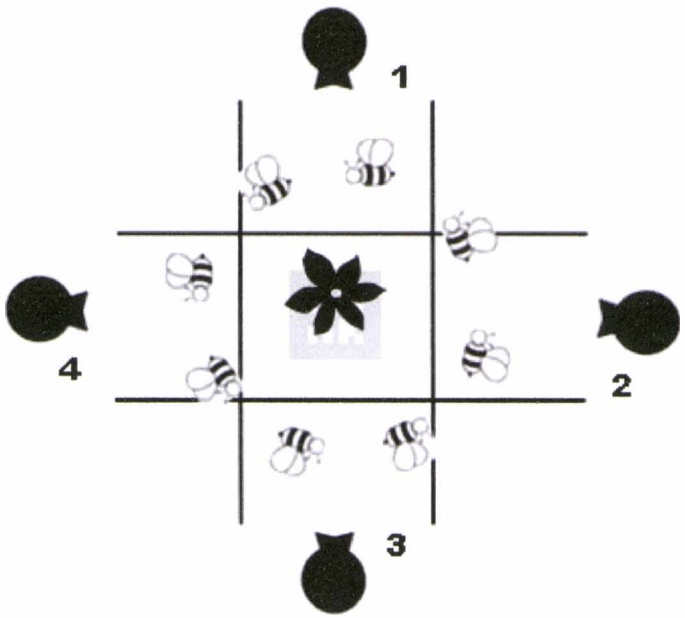


Fig 4.11. Las cámaras pares e impares tienen buena perspectiva

Si bien deben realizarse cálculos y comparaciones en cada iteración, éstos no acarrean la misma complejidad que los del esquema anterior, por lo tanto se estima que es escalable a un número razonable de cámaras, aproximadamente 7 u 8 sin notar degradaciones de performance.

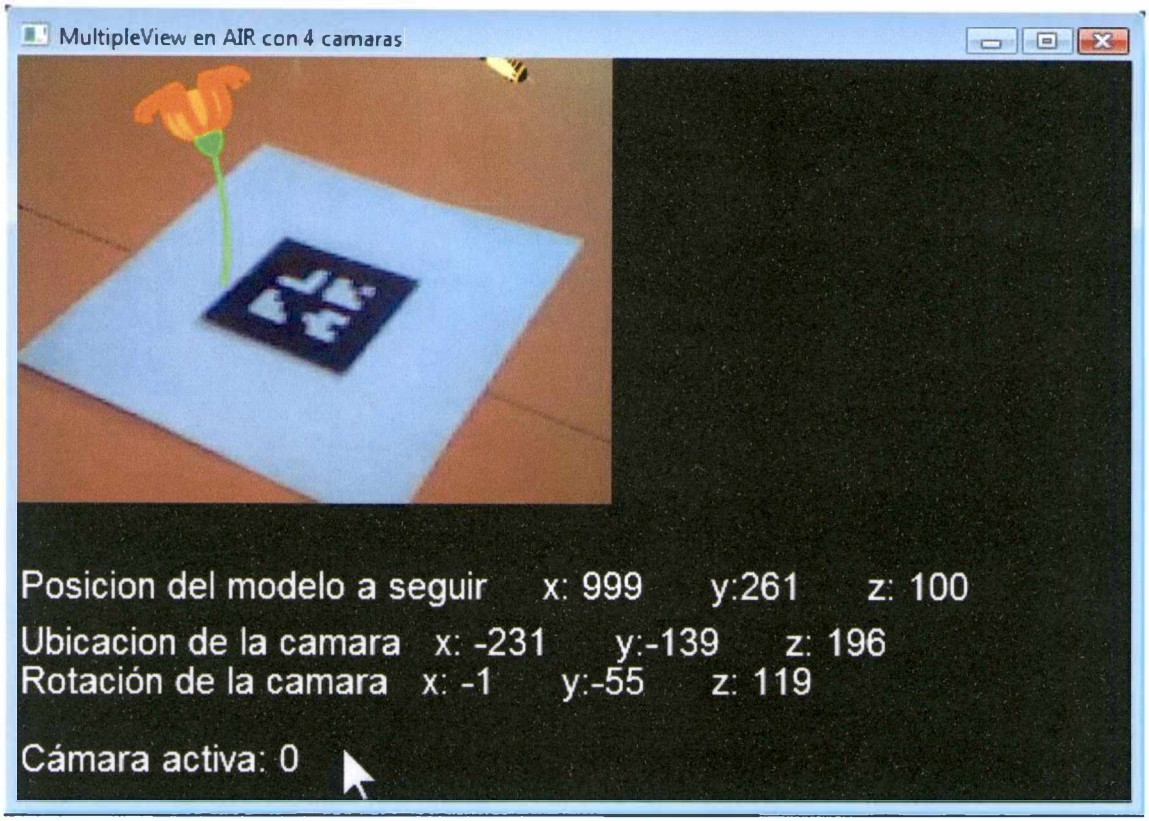


Figura 4.12. La abeja se escapa del ángulo de vista de la cámara activa

El esquema de mejor perspectiva evita la situación que presenta la Figura 4.10 (capturada del esquema manual). En la imagen se puede apreciar que la abeja no se muestra en su totalidad, sino que está a punto de desaparecer por la parte superior del video. En el primer *frame* en el que la abeja comienza a renderizarse parcialmente, el control es cedido a otra cámara que tenga una visión completa de la misma.

4.5 Resultados de la experiencia del usuario

El siguiente cuadro resume la información recabada luego de ejecutar la misma aplicación con los distintos esquemas:

Esquema	Inicialización	Interacción del usuario
Esquema Manual	No requiere.	Cada vez que desea cambiar la cámara debe presionar la tecla C.
Esquema de slice de tiempo	No requiere.	No requiere.
Esquema de proximidad al modelo	Se debe esperar a que la abeja se aproxime a todas las cámaras para inicializarlas.	No requiere.
Esquema de mejor perspectiva	Se debe esperar a que todas las cámaras tengan una buena perspectiva para inicializarlas.	No requiere.

Esquema	Cambio de cámaras		
	Decidir si debe cambiar	Escoger la cámara	Hacer el swap
Esquema Manual	Estas acciones no demandan tiempo, el usuario lo determina al presionar la tecla y la próxima cámara es la que sigue en el arreglo.		**c** Tiempo constante de FLARTookit
Esquema de slice de tiempo	Chequear el timer (condición simple)	No demanda tiempo, la próxima cámara es la que sigue en el arreglo.	**c** Tiempo constante de FLARTookit
Esquema de proximidad al modelo	**n . t Pipeline()** Renderizar la vista de cáda cámara para obtener el valor de su distancia.	**t QuickSort(n)** Ordenar los valores y devolver el mínimo.	**c** Tiempo constante de FLARTookit
Esquema de mejor perspectiva	**n . t Pipeline()** Renderizar la vista de cáda cámara para saber si puede ser seleccionada.	**t Random(values)** Escoger un valor al azar entre todas las que tienen una buena perspectiva.	**c** Tiempo constante de FLARTookit

NOTAS:

t Pipeline() = Tiempo que requiere para mostrar en pantalla el flujo de una cámara y renderizar el modelo.

t QuickSort() = Tiempo de ejecución de un algoritmo de ordenación sobre la estructura que almacena los valores de las distancias.

t Random() = Tiempo de ejecución de una función que escoge un valor al azar entre los que recibe como parámetro.


FOLIO
65

Capítulo 5

Conclusiones y trabajo a futuro

En este documento se describió una solución para una importante limitación que presenta la Realidad Aumentada. En algunos casos, cuando sus aplicaciones incluyen una sola webcam, la calidad de la experiencia del usuario puede verse afectada dado que los modelos que se renderizan pueden ser muy complejos o incluyen animaciones que exceden al campo de visión de la cámara, que se encuentra estática. Para tal fin se incorporaron más cámaras y se proporcionaron diversos esquemas para poder manejarlas. Se obtuvo un dato relevante para la implementación de algunos de esos esquemas: la distancia real que existe entre la cámara y el marker, de manera de poder realizar comparaciones entre sus ubicaciones físicas. Sobre la base de FLARToolkit, un framework para la creación de aplicaciones convencionales de RA en ActionScript, se desarrolló una extensión que permite separar la implementación del modelo a renderizar y provee diversos esquemas, permitiendo desarrollar nuevos e incorporarlos fácilmente.

A continuación se realiza una comparación de los esquemas que han sido propuestos. Luego se enumeran aspectos importantes que deben ser tenidos en cuenta y potenciales problemas que de ellos se desprenden. Más adelante se relatan los aportes realizados por este trabajo y por último, se describen posibles trabajos que pueden dar continuación a éste.

5.1 Comparativa de esquemas

Podemos establecer una comparativa entre los diferentes esquemas y definir valores concretos para sus parámetros:

Manual	
Uso de variables	No requiere
Escalabilidad	Alta. Una nueva cámara no aumenta el uso de variables.
Resumen	El esquema es simple pero no provee una sensación realista dado que el usuario es quien controla lo que sucede y podría no escoger la cámara correcta

Slice de tiempo	
Uso de variables	Controlado. Sólo se controla el *timer* para poder cambiar la cámara.
Escalabilidad	Alta. El *timer* sigue siendo único por mas que se incremente el número de cámaras
Resumen	Es otro esquema simple, que libera al usuario del cambio de cámara pero puede no elegir la toma correcta.

Proximidad al modelo	
Uso de variables	Alto. Por cada cámara se realizan muchos cálculos en cada *frame* de video.
Escalabilidad	Baja. Agregar una nueva cámara implica realizar muchos cálculos más lo cual degrada la performance y ralentiza la aplicación.
Resumen	El uso de recursos es mayor pero brinda automatización y asegura tomas correctas siempre.
Otro	Posee las mismas características que el **esquema de proximidad a un evento**

Mejor perspectiva (clearview)	
Uso de variables	Moderado. Por cada cámara se llama a una función para chequear su perspectiva
Escalabilidad	Normal. Cada cámara solo llama a una función simple.
Resumen	Este esquema puede ser productivo si siempre se tiene una cámara con una buena perspectiva; el consumo no es grande y la escalabilidad es buena. En algunos casos puede derivar en un bloqueo de la aplicación.

5.2 Aspectos a tener en cuenta

Todos los elementos en las aplicaciones de RA en desktop son de vital importancia, pero uno de ellos puede afectar notablemente el desempeño de las mismas: la cámara web. En aquellas aplicaciones que hacen uso de una única cámara, interesa tener un flujo de video sin intermitencias y una buena detección del marker. Para esto es necesario focalizar la atención en los siguientes parámetros:

Cantidad de cuadros por segundo (FPS)

Es la medida de la frecuencia con la cual la cámara genera distintos fotogramas o cuadros (*frames*). Cada cuadro se compone de un conjunto de pixeles y la continua sucesión de éstos produce a la vista la sensación de movimiento, fenómeno dado por las pequeñas diferencias que hay entre cada uno de ellos. Es decir que mientras más grande sea el valor de los FPS, mayor fluidez tendrá el video.

Un valor aceptable es 30 FPS.

Resolución

Es la medida del número de pixeles que entran en un frame. A mayor resolución se tiene una imagen con más detalles y mejor definición, lo cual simplifica la tarea del detector del marker.

Un valor óptimo es 640 x 480 pixeles, aunque también es aceptable una resolución de 320 x 240 pixeles.

Óptica

Se refiere al enfoque de la cámara. Algunas tienen un anillo de enfoque que se debe ajustar manualmente para poder usarla a distintas distancias. Otras poseen un 'iris' que les permiten adaptarse automáticamente a los distintos tipos de iluminación.

Conexión

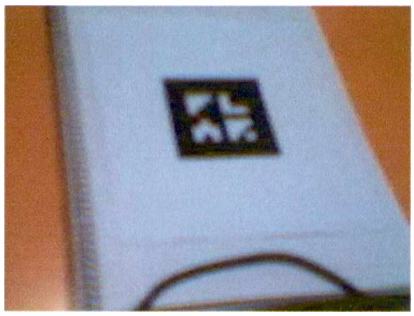
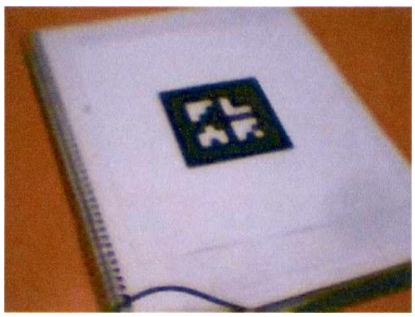
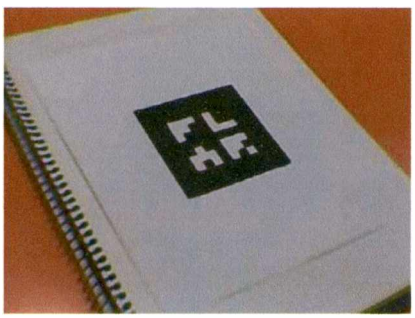
La mayoría de las cámaras ofrecen una conexión USB 2.0. Hoy en día ésta es la más rápida, permitiendo transferencias de hasta 480 Mb/s. Sin embargo, existen aún algunas cámaras con conexión USB 1.1, las cuales ofrecen una baja calidad de imagen.

Múltiples cámaras

En este desarrollo intervienen más de una cámara, por lo tanto lo ideal sería que todas ellas brindaran prestaciones similares. Si esto no fuese así se degradaría notablemente la performance. Las siguientes situaciones lo ejemplifican:

- Si una cámara otorga 30 FPS y otra 10 FPS, al momento de capturar con aquella de 10 FPS, la aplicación se ralentiza, la captura es entrecortada y la detección no es óptima.
- Lo mismo sucede si existe una diferencia de resoluciones (Cuadro 5.1): cuando el esquema escoge una cámara con una resolución inferior a la que se encontraba anteriormente, la

captura se vuelve más borrosa, el marker no se encuentra bien definido y, nuevamente, la detección puede fallar.

	***Logitech*** *QuickCam Express 100* FPS: 30. Resolución: 640x480px.
	***Logitech*** *QuickCam Express 150* FPS: 30. Resolución: 640x480px. Difiere de la anterior ya que posee un controlador mejorado.
	***Genius*** *I-LOOK 300* FPS: 30. Resolución: 1.3 Mpx.

Cuadro 5.1. Comparativa de resoluciones de cámaras web

Iluminación

Además de los parámetros propios de cada cámara, debemos tener en cuenta ciertos factores externos que inciden en la aplicación. Uno de los más importantes es **la luz**. Todas las cámaras web responden de manera distinta frente a la iluminación que existe en la imagen percibida, y si esta luz es artificial (lámparas de techo, veladores) la calidad se degrada mucho en comparación con la luz natural. Para una mejor respuesta de la cámara conviene siempre aprovechar la luz solar en caso de ser posible, dado que provee una iluminación más pareja.

Cantidad de conexiones

Es importante mencionar que al ser todas cámaras que se conectan mediante USB, en principio se podrán agregar tantas como puertos disponibles haya. En caso de querer incorporar más, se debe hacer uso de cámaras integradas (si se dispone de ellas), un HUB de puertos USB, o bien utilizar cámaras que se comuniquen vía Bluetooth o infrarrojo.

5.4 Resumen de los aportes

A continuación se enumera una lista de los aportes realizados por este trabajo:

- Se introdujo un nuevo concepto en Realidad Aumentada, incorporando más de una cámara en aplicaciones desktop.
- Se implementaron esquemas para poder manejar esas múltiples cámaras y se propusieron otros a ser implementados.
- Se desarrolló sobre la base de un framework que en principio solo admite una sola webcam, una librería para poder escalar una aplicación a **n** cámaras y que todas ellas funcionen simultáneamente.
- Se obtuvo a partir de los datos que provee la detección de un patrón con una cámara, la posición de la cámara dentro del espacio.

5.5 Trabajo a futuro

5.5.1 Flash 11 y su motor de renderizado 3d nativo

La versión 11 de Adobe Flash incluirá un motor de renderizado de gráficos 3d. Esto significa que no será necesaria la inclusión de librerías externas (en el caso de este trabajo, Papervision3D) para poder dibujar modelos en 3 dimensiones. Sin duda representa un significativo avance ya que se lograrán mejores performances en lo que a FPS respecta y una integración de modelos 3d externos mas práctica. Uno de los principales motivos por los cuales los desarrolladores deciden llevar a cabo esta nueva implementación radica en el éxito de las aplicaciones de Realidad Aumentada.

5.5.2 Estudio del recorrido

Al haber realizado un cálculo que permite conocer la distancia entre la cámara y el patrón, se puede hacer uso de esta información para saber como interactúan estos elementos. Si el dispositivo de captura no se encontrara estático y, por ejemplo, estuviera adosado al usuario, se podrían saber sus movimientos respecto del patrón.

Para ejemplificarlo mas claramente se plantea el siguiente escenario: en una sala de un museo se dispone un marker, ubicado en el suelo y centrado. Se encuentra entonces en las coordenadas [0, 0, 0], tanto de posición como de rotación. Al entrar, una persona se coloca un dispositivo montado en la cabeza mediante el cual percibe el marker y, sobre él, el modelo que se renderiza. Si la persona se mueve alrededor del marker, en cada momento se puede saber su posición dentro de la sala. Si se guardase el recorrido en una base de datos, se podría comparar contra otros y saber, por ejemplo, en qué lugares se detienen las personas, para apreciar cuáles puntos de vista del modelo.

5.5.3 Un sistema que aprenda

De la misma manera que se puede analizar el recorrido de la cámara que interactúa frente al marker, es posible enfocar la atención en el comportamiento del modelo. Si se tratase de un modelo animado, tal como se demostró en uno de los esquemas, se puede saber, en cada instante, la posición de alguno de los objetos que lo componen. Con esta información es posible establecer cuál es el recorrido que realiza una y otra vez. Luego de cierta cantidad de ciclos, el sistema podría ser capaz de deducir cuál sería la próxima toma de video y hacer un cambio de cámara anticipándose al comportamiento del modelo.

En ciertos casos, por ejemplo con modelos que se desplazan rápidamente, el cambio de cámara resulta brusco y cuando es el turno de la nueva cámara, el modelo puede haberse movido, resultando en una toma breve que sólo logra desconcertar al usuario. Con esta mejora, la toma sería mas larga, dado que comenzaría unos instantes antes, cuando el modelo comienza a aproximarse a la cámara en cuestión.

5.6 Bibliografía

[1] http://www.hitl.washington.edu/artoolkit, *Web del proyecto ARToolkit.*

[2] http://handheldar.icg.tugraz.at/artoolkitplus.php, *Web del proyecto ARToolkitPlus.*

[3] L. Surhone, M. Timpledon, S. Markesen, *Rasterization.* Betascript Publishing 2010.

[4] C. Moock, *Essential ActionScript 3.0.* O'Reilly 2007.

[5] M. Chambers, D. Dura, K. Hoyt, *Adobe Integrated Runtime (AIR) for JavaScript Developers Pocket Guide.* O'Reilly 2010.

[6] J. Winder, P. Tondeur, *Papervision3D Essentials.* Septiembre 2009.

[7] F. Balaguer, M. Esposito, *A multiple camera approach to desktop augmented reality applications.* WAVI 2010.

[8] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design patterns.* Addison Wesley 1995.

[9] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3, *API de ActionScript3.*

[10] S. Benford, D. Rowland, M. Flintham, A. Drozd, R. Hull, J. Reid, J. Morrison, K. Facer, *Life on the Edge: Supporting Collaboration in Location-Based Experiences.* Abril 2005.

[11] http://www.libspark.org/wiki/saqoosha/FLARToolKit, *Web del proyecto FLARToolkit.*

[12] http://words.transmote.com/wp/flarmanager, *Blog del proyecto FLARManager.*

[13] J. Gonzalez Cobas, *Mathematics of 3D Graphics.* Blender Conference 2004.

[14] L. Leithold, *El cálculo, 7ma edición.* Oxford 1998.

[15] S. Cawood, M. Fiala, *Augmented Reality. A practical guide.* The pragmatic programmers 2009.

[16] Bares, William, Scott McDermott, Christina Boudreaux, Somying Thainimit, *Virtual 3D camera composition from frame constraints.* 2000.

[17] M. Christie, R. Machap, J. Norm, P. Olivier, J. Pickering, *Virtual Camera Planning: A survey.* 2005.

[18] H. Li-wei, M. Cohen, D. Salesin, *The Virtual Cinematographer: A Paradigm for Automatic Real-Time Camera Control and Directing.* 1996

[19] T. Cuypers, *A-Multi-Camera System for Interactive Videogames.* 2007.

[20] Gagne, Galvin, Silberschatz, *Fundamentos de Sistemas Operativos.* Mc Graw Hill 2005.