CAPÍTULO 1

Casos de Uso

1.1. Introducción

En este capítulo se presentan los distintos casos de uso que se implementaron con el fin de integrar los algoritmos comentados en capítulos anteriores en pequeñas aplicaciones que funcionen *de punta a punta*. Se buscó resolver individualmente los diferentes desafíos técnicos que una aplicación real de realidad aumentada para museos puede llegar a tener. Estas últimas no serán más que una combinación guionada de cada uno de estos casos de uso.

A lo cargo del capítulo se verán entonces los tres casos de uso implementados: “interactividad”, “video” y “modelos”. El primero presenta un modelo simple sobre el marcador que responde a toques con cierto movimiento y un audio en particular, el segundo soluciona el problema de proyectar un video sobre el marcador de forma consistente con el movimiento del usuario. El último caso de uso muestra cómo es posible importar modelos a ISGL3D de manera de lograr realidades aumentadas mucho más interesantes que si tan sólo se hicieran con las primitivas del *framework*, por detalles ver capítulo **??**.

1.2. Caso de uso “interactividad”

1.2.1. Comentarios sobre el caso de uso

En este caso de uso se implementa la parte interactiva de la aplicación. Al enfocar el marcador, se puede ver un cubo sobre el QlSet de la esquina superior izquierda. Ver Figura 1.1. Si el cubo es tocado a través de la pantalla del dispositivo, este se anima y se reproduce un audio que indica la posición del cubo en el instante de ser presionado. Inmediatamente después, es desplazado hacia el QlSet de la esquina superior derecha. Nuevamente, si el cubo es tocado a través de la pantalla del dispositivo, este se anima y se reproduce un audio que indica la posición del cubo en el instante de ser presionado. Inmediatamente después, este se desplaza hacia el QlSet restante. Lo anterior sucederá de forma cíclica, cada vez que se presione sobre el mpdelo.

Esta funcionalidad es fundamental si lo que se quiere implementar es por ejemplo una audioguía interactiva. Podría pensarse una aplicación en la que el cubo anterior se reemplace por flechas 3D, y que estas sean ubicadas conjuntamente en distintas partes de una obra. Entones, al seleccionar cada una de las flechas, se podría reproducir un audio con información referente a esa zona o punto en particular.

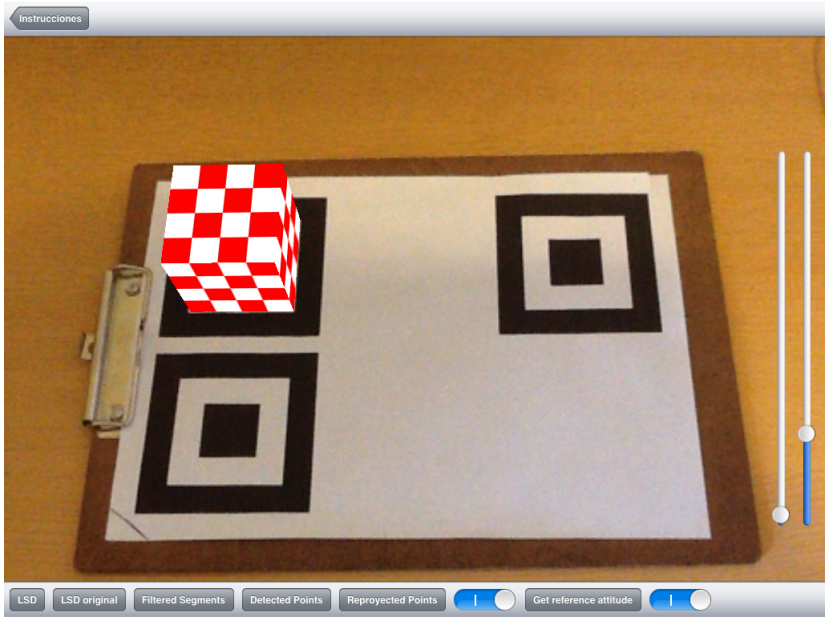

Figura 1.1: Captura de pantalla del caso de uso “interactividad’. Se puede ver al cubo apoyado sobre el QlSet de la esquina superior izquierda y los diferentes controles que ayudan a la depuración del código.

Esta aplicación también se utilizó con fines de *debugging* o depuración de la integración de cada uno de los bloques. Se le agregaron las siguientes funcionalidades:

- La posibilidad de ver dibujados sobre la imagen los segmentos detectados por LSD. En sus versiones original y opimizada.
- La posibilidad de ver dibujados sobre la imagen los segmentos filtrados pertenecientes al marcador. Así como también las esquinas detectadas de cada uno de los cuadriláteros que lo forman.
- La posibilidad de variar el umbral utilizado para el filtrado de segmentos.
- La posibilidad de ver las esquinas de cada uno de los cuadriláteros que forman al marcador reproyectadas según la pose del dispositivo obtenida.
- La posibilidad de prender o apagar el filto de Kalman.
- La posibilidad de aumentar o disminuir el ruido de medición del filtro de Kalman.
- La posibilidad de elegir si usar o no la fusión de la estimación de pose con los sensores.

En la Figura 1.1 también se puede ver cómo es la interfaz de usuario de este caso de uso, en donde se puede elegir entre todas las funcionalidades anteriores. El mismo fue fundamental para evaluar el desempeño de los algoritmos utilizados funcionando en tiempo real. Gracias a estas funcionalidades se pudieron definir las condiciones para las cuales el conjunto de todos los bloques funciona mejor. Se fue variando la distancia al marcador y se ajustó el umbral para el filtro de segmentos. Además, se pudieron ajustar los parámetros del filtro de Kalman y se pudo comparar el desempeño de la estimación de pose utilizando solamente información de la cámara con el resultado obtenido de la fusión de sensores. Fue en este caso de uso que se evaluó cualitativamente el desempeño de la

versión optimizada de LSD, respecto del de la versión original.

Si bien todas estas pruebas y ajustes si hicieron previamente en una computadora y con imágenes de prueba, fue necesario contar con una aplicación en la que se pudiera ver sobre el dispositivo al conjunto de los algoritmos funcionando en tiempo real.

1.2.2. Detalles constructivos

1.2.2.1. Objetos ISGL3D interactivos

La manera de agregar interactividad a un nodo ISGL3D es bastante sencilla. En primer lugar, debe configurarse su propiedad *interactive* de forma positiva y luego se le debe ejecutar el método *addEvent3DListener*:

```
Isgl3dTextureMaterial * material = [Isgl3dTextureMaterial
materialWithTextureFile:@"red_checker.png" shininess:0.9
precision:Isgl3dTexturePrecisionMedium repeatX:NO repeatY:NO];

Isgl3dCube* cubeMesh = [Isgl3dCube  meshWithGeometry:60 height:60 depth:60 nx:40 ny:40];

Isgl3dNode * _cubito = [self.scene createNodeWithMesh:cubeMesh andMaterial:material];

_cubito.interactive =YES;

[_cubito1 addEvent3DListener:self method:@selector(objectTouched:) forEventType:TOUCH_EVENT];
```

En el código anterior, primero se crea un nodo llamado "_cubito" con la primitiva de un cubo y cierto material. Luego, se indica que sí se quiere que dicho nodo tenga interactividad y finalmente se lo configura para que cuando "_cubito" reciba eventos del tipo *TOUCH_EVENT*, o lo que es lo mismo, cuando se lo toque; se ejecute el método *objectTouched*, definido en la misma clase que esta escrito el código (*self*).

En este caso de uso lo que se hizo en *objectTouched* no fue más que cambiar la posición del cubo en la escena y reproducir un audio dependiente de la posición del mismo.

1.2.2.2. Reproducción de audio en Objective-C

Para reproducir audios en Objective-C primero la clase en la que se quiere reproducir el audio debe importar el *framework AVFoundation* y luego debe implementar el protocolo *AVAudioPlayer-Protocol*. El código que se debe escribir es el sigiuente:

```
NSURL *url =[NSURL fileURLWithPath:[NSString stringWithFormat:@"%@/%@",
        [[NSBundle mainBundle] resourcePath],audio.mp3]];

AVAudioPlayer * audioPlayer =[[AVAudioPlayer alloc] initWithContentsOfURL:url error:nil];

audioPlayer.numberOfLoops=0;

audioPlayer.delegate = self;

[audioPlayer play];
```

En la primera línea se genera un *url* que indica cuál es el audio a reproducir y luego se le asigna a una instancia de la clase *AVAudioPlayer*. Se dice que no se quiere reproducir el audio en bucle,

se asigna a la clase en la que se esta escribiendo el código como la delegada de *audioPlayer*, una instancia de *AVAudioPlayer,* y finalmente se le da inicio al audio. Luego de reproducido el audio, se ejecuta autamáticamente el método de firma:

```
- (void)audioPlayerDidFinishPlaying:(AVAudioPlayer *)player successfully:(BOOL)flag;
```

En este código es en donde se indica que la próxima vez que se presione sobre el cubo, se querrá reproducir un audio distinto.

1.2.2.3. Dibujar en ISGL3D

Lo que se hizo fue crear una clase nueva, del tipo *UIVIew*, a la que se la llamó "claseDibujar". Esta fue agregada como *subView* de la *view* en donde se muestra el video por detrás de lo que dibuja ISGL3D. Dicha clase se configuró para que fuera transparente y del mismo tamaño que la pantalla del *iPad*. *claseDibujar* cuenta con una cantidad de propiedades a las que se les asignan los diferentes puntos o segmentos que se quieren dibujar; son del tipo "puntero a entero" y "puntero a *float*" respectivamente. Luego, un método llamado *drawRect* es el que se encarga de dibujar cada uno de los puntos y segmentos. Los puntos se dibujan con las siguientes líneas de código:

```
CGContextRef context = UIGraphicsGetCurrentContext();

CGContextStrokeRect(context, CGRectMake(punto_X, punto_Y, 4, 4));
```

En la primera línea de código se crea un contexto. Un contexto contiene ciertos parámetros y toda la información especifica del dispositivo, requerida para poder dibujar. En la segunda línea se dibuja cada punto como un rectángulo centrado en el punto en cuestión y con 4 píxeles de ancho y largo. Los segmentos se dibujan con las siguientes líneas de código:

```
CGContextRef context = UIGraphicsGetCurrentContext();

CGContextStrokeLineSegments(context, puntos, 2);
```

En la primera línea de código se crea un contexto (este paso puede saltearse si ya fue creado anteriormente), y en la segunda se dibuja la línea. La variable "puntos" es un arreglo de dos variables del tipo *CGPoint*, cada una de ellas tiene dos valores en precisión simple correspondientes a las coordenadas de un punto. Además, se le configura al segmento una anchura de 2 píxeles.

Finalmente, es bueno aclarar que *claseDibujar* se instancia y se destruye cuadro a cuadro; el método *drawRect* se invoca cada vez que se instancia la clase.

1.3. Caso de uso "video"

1.3.1. Comentarios sobre el caso de uso

Este caso de uso proyecta un video sobre el QlSet de la esquina superior izquierda del marcador de manera consistente con la pose del dispositivo. Ver Figura 1.2. La aplicación de esta solución técnica es directa. Tan sólo ajustando un par de parámetros el video podría ser proyectado dentro del marco de un cuadro, sobre uno de sus extremos, sobre una pared blanca o incluso sobre un mapa. Esto puede ser de gran interés para un museo, por ejemplo como complemento a una audioguía. A continuación se explican brevemente algunos detalles técnicos que fue necesario solucionar para lograr implementar este caso de uso.

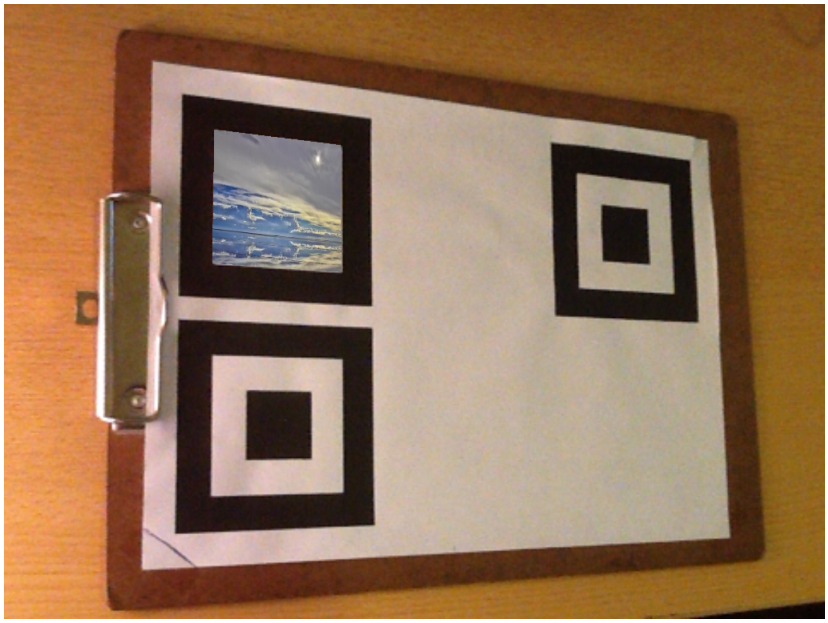

Figura 1.2: Captura de pantalla del caso de uso “video”. Se puede ver al video proyectado sobre el QlSet de la esquina superior izquierda

1.3.2. Detalles constructivos

Para lograr lo propuesto para este caso de uso se implementó un proyecto que proyecta el video en uno de los cuadrados del marcador. De esta manera, de toda la lógica de estimación de pose, solamente se hace uso de la detección y filtrado. En particular no se hace uso de los resultados del posit. Teniendo entonces detectados los cuatro puntos en los que se quiere reproducir el video parecería que el problema está resuelto. Sin embargo, xcode no permite posicionar en forma directa una vista de video en cualquier conjunto de cuatro puntos.
Si simplemente se quiere reproducir un video, y no se quiere procesar el contenido, lo más cómodo para hacerlo es utilizar la clase *MPMoviePlayerController* que hereda de *NSObject*. Una alternativa similar es haciendo uso de la clase *MPMoviePlayerViewController* que hereda de *UIViewController* y tiene como única propiedad una del tipo *MPMoviePlayerController*.
MPMoviePlayerController tiene un atributo *view* del tipo *UIView* que es la vista y es este atributo el que se quiere posicionar en los cuatro puntos detectados por el filtro. Un atributo del tipo *UIView* tiene un atributo *frame* que es del tipo *CGRect*

```
theMovie.view.frame = CGRectMake(0, 0, 60, 60);
```

En el código anterior *theMovie* es del tipo *MPMoviePlayerController*. De esta manera, se tiene el inconveniente de que en principio cualquier video parecería que solamente puede ser reproducido sobre rectángulos y no en cualquier polígono de cuatro puntos por ejemplo. Sin embargo algo que sí se puede hacer a las instancias de la clase *UIView* es una transformación afin o incluso, de manera más genérica, una homografía.

1.3.3. *CGAffineTransform* y *CATransform3D*

La clase *UIView* tiene una propiedad llamada *transform* que es del tipo *CGAffineTransform*. Las primeras letras de esta clase (*CG*) refieren a la API **Core Graphics** utilizada ampliamente como herramienta para resolver *rendering* y cualquier tipo de transformación en 2D.
La clase *UIView* también tiene una propiedad llamada *layer* que es del tipo *CALayer* y que permite

realizar transformaciones del tipo *CATransform3D*. Las primeras letras de estas dos clases (*CA*) refieren a la API **Core Animation** que es utilizada para generar animaciones y transformaciones sobre objetos 3D solamente indicando un punto inicial y final para el objeto (también es posible agregar efectos para la transición). En definitiva para resolver el problema del caso de uso existen a priori dos alternativas posibles: *CGAffineTransform* y *CATransform3D*.
Se pueden generar fácilmente instancias transformaciones afines invocando la siguiente función:

```
CGAffineTransform CGAffineTransformMake (
   CGFloat a,
   CGFloat b,
   CGFloat c,
   CGFloat d,
   CGFloat tx,
   CGFloat ty
);
```

que toma 6 *CGFloats* y crea una *CGAffineTransform*, donde cada uno de los valores anteriores se corresponde con los elementos de una matriz transformación afín de la siguiente manera:

$$\begin{pmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{pmatrix}$$

Así entonces, de los 9 valores de la matriz, 2 de ellos son nulos por tratarse de una transformación afín y otro de ellos es unitario como valor de escala. Resolviendo el sistema como se muestra en la sección 1.3.4 y obteniendo los restantes 6 valores, se le puede asignar transformaciones a la propiedad *transform* y realizar la trasnformación deseada. Este método tuvo como inconveniente el hecho de que .....
EXPLICAR POR QUE NO FUNCIONÓ
Por su parte también es sencillo generar instancias de transformaciones 3D debido a que existe el tipo de dato definido para generar la matriz *CATransform3D* como:

```
struct CATransform3D
{
CGFloat m11, m12, m13, m14;
CGFloat m21, m22, m23, m24;
CGFloat m31, m32, m33, m34;
CGFloat m41, m42, m43, m44;
};
typedef struct CATransform3D CATransform3D;
```

donde m_{ij} corresponde al elemento de la matriz ubicado en la fila i columna j. Así entonces también es posible, conociendo los valores de la homografía, completar los elementos de esta matriz 4x4 y asignársela a la propiedad *layer* de la *UIView*. Esta opción de generar una transformación 3D permite incluir transformaciones más generales que una homografía o una transformación afín. Si lo que se busca es que esta matriz represente una homografía (2D), es necesario entonces que la coordenada z sea nula, es decir

$$\begin{pmatrix} m_{11} & m_{12} & 0 & m_{14} \\ m_{21} & m_{22} & 0 & m_{24} \\ 0 & 0 & 1 & 0 \\ m_{41} & m_{42} & 0 & m_{44} \end{pmatrix}$$

donde a su vez m_{44} se asume de valor unitario por ser un factor de escala. De la misma manera que para la transformación afín, resolviendo la homografía como se ve en la sección 1.3.4 se obtienen los 8 valores restantes de la matriz y es posible asignarle una homografía a un objeto *UIView* para resolver el problema presente.

1.3.4. Resolución de Homografía

A continuación se hace el desarrollo de la resolución del sistema de ecuaciones que se tuvo que resolver para hallar los parámetros de la homografía que transforma una imagen de referencia en la imagen que se tiene en cada momento como resultado de la captura de la cámara. Se asume entonces que se conocen los puntos de referencia y los puntos de referencia transformados (los detectados luego del filtrado de segmentos) y lo que se quiere averiguar es la matriz h que logra dicha transformación. Esta homografía 2D-2D se puede expresar en forma matricial, en coordenadas homogéneas de la siguiente manera:

$$\begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} i \\ j \\ k \end{pmatrix}$$

donde la matriz h_{3x3} representa la transformación homográfica, el vector $(x,y,z)^t$ representa los puntos de referencia a ser transformados y el vector $(i,j,k)^t$ respresenta los puntos detectados cuadro a cuadro como las esquinas del marcador.
Asumiendo un valor unitario para las coordenadas z y k la resolución del sistema se simplifica mucho y no se pierde generalidad. Imponiendo esto entonces, el sistema anterior se puede expresar de la siguiente forma:

$$xh_{11} + yh_{12} + h_{13} = i \tag{1.1}$$

$$xh_{21} + yh_{22} + h_{23} = j \tag{1.2}$$

$$xh_{31} + yh_{32} + h_{33} = 1 \tag{1.3}$$

Multiplicando la ecuación (1.3) por i e igualándola a la ecuación (1.1) se obtiene lo siguiente:

$$xh_{11} + yh_{12} + h_{13} = ixh_{31} + iyh_{32} + ih_{33} \tag{1.4}$$

o lo que es lo mismo:

$$xh_{11} + yh_{12} + h_{13} - ixh_{31} - iyh_{32} - ih_{33} = 0 \tag{1.5}$$

Procediendo de manera análoga y multiplicando la ecuación (1.3) por j e igualándola a la ecuación (1.2) se obtiene lo siguiente:

$$xh_{21} + yh_{22} + h_{23} = jxh_{31} + jyh_{32} + jh_{33} \tag{1.6}$$

o lo que es lo mismo:

$$xh_{21} + yh_{22} + h_{23} - jxh_{31} - jyh_{32} - jh_{33} = 0 \tag{1.7}$$

Las ecuaciones (1.3) y (1.7) se pueden expresar en forma matricial, de la siguiente manera:

$$\begin{pmatrix} x & y & 1 & 0 & 0 & 0 & -ix & -iy & -i \\ 0 & 0 & 0 & x & y & 1 & -jx & -jy & -j \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Teniendo entonces 4 parejas de puntos referencia y puntos transformados y asumiendo h_{33} de valor unitario se tiene entonces 8 ecuaciones y 8 incógnitas, lo que lo vuelve un sistema compatible determinado que se puede expresar de la siguiente manera:

$$\begin{pmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 & -i_0x_0 & -i_0y_0 \\ 0 & 0 & 0 & x_0 & y_0 & 1 & -j_0x_0 & -j_0y_0 \\ x_1 & y_1 & 1 & 0 & 0 & 0 & -i_1x_1 & -i_1y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -j_1x_1 & -j_1y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -i_2x_2 & -i_2y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -j_2x_2 & -j_2y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -i_3x_3 & -i_3y_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -j_3x_3 & -j_3y_3 \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix} = \begin{pmatrix} i_0 \\ j_0 \\ i_1 \\ j_1 \\ i_2 \\ j_2 \\ i_3 \\ j_3 \end{pmatrix}$$

Así entonces, lo que se hace para resolver la homografía es cuadro a cuadro tener detectados los puntos en los que se quiere presentar la vista del video que se corresponden con cuatro puntos detectados por el filtro y tener las correspondencias con el marcador real, se posiciona la vista en la posición de referencia y se le aplica la homografía hallada que vincula la posición referencia con los puntos detectados.

1.4. Caso de uso “modelos”

1.4.1. Comentarios sobre el caso de uso

1.4.2. Detalles constructivos

[**?**].