CAPÍTULO 1

Herramientas

1.1. Herramientas de Benchmark

Durante el desarrollo del proyecto se hizo presente la necesidad de contar con herramientas de medida del desempeño objetivas para la validación de los algoritmos que se utilizaron y desarrollaron. En particular se desea evaluar los algoritmos asociados a la detección de características y estimación de pose. Estos son LSD, filtrado de segmentos, determinación de correspondencias y Posit Coplanar.

El enfoque elegido para la validación de dichos algoritmos consiste en la generación de un conjunto de "datos de referencia" o *ground truth* de forma de poder comparar la detección y resultados obtenidos con los valores reales o verdaderos de los mismos. Este proceso es lo que se denomina en este proyecto como *Benchmark* o Comparativa. De esta forma se puede generar un número suficiente de datos de salida de forma de obtener algunas estadísticas de interés para la evaluación de los algoritmos.

Debido a la naturaleza del proyecto fue necesario generar un entorno para la evaluación el cual incluye la generación de imágenes sintéticas asociadas a esa *ground truth* en Blender. También fue necesario el desarrollo de un entorno en Matlab que permita una ejecución tipo batch para la generación de estas imágenes así como para la generación, análisis y despliegue de resultados en forma rápida y simple.

1.1.1. Generación de imágenes sintéticas: Blender

Para realizar el Benchmark se desarrolló un entorno de generación de imágenes sintéticas que utiliza herramientas gratuitas y de código abierto que permiten capacidades de *scripting* y ejecución en forma "batch" de forma de serializar la producción de renders en un proceso automático. El proceso de generación de los renders fue realizada mediante el uso del software Blender.

La generación de imágenes sintéticas consiste en producir una imagen 2D en base a un modelo 3D de una escena y una pose de la cámara respecto a esa escena o viceversa. Este proceso se denomina *Rendering*, o renderizado, y el mismo se trata en el Capítulo **??** para la herramienta para dispositivos móviles *ISGL3D*.

Blender es un *suite* de creación de contenidos gratuito y de código abierto. Está disponible para los principales sistemas operativos, Windows, Mac, Linux; bajo la Licencia Pública General GNU[**?**]. Algunas de sus prestaciones son modelado, *shading*, animación, renderizado, e interacción 3D. También presenta herramientas de *scripting* mediante el uso del lenguaje *Python* las cuales se explicarán más adelante. Se utilizó la versión de Blender 2.58 para el desarrollo siguiente ha sido utilizado satisfactoriamente con versiones más recientes como la 2.63.

1.1.1.1. Modelo 3D y escena

Para lograr hacer un render es necesario contar con una escena 3D y en particular con un modelo 3D del objeto de interés. Para realizar la detección de características se utiliza un marcador plano que se describe en el Capítulo **??**.

Se manejaron algunas alternativas para la generación del modelo 3D del marcador entre las cuales se encuentran, VTK, Paraview, Autodesk 3ds Max y el mismo programa utilizado por este proyecto para rendering en un PC, Blender. Finalmente debido a la simplicidad del marcador, y principalmente a que es un marcador plano, se utilizó un dibujo vectorial del marcador (formato *svg*) generado en Inkscape. Este dibujo vectorial puede ser importado a Blender y con él generar un modelo 3D que será un actor en la escena. Dentro de la interfaz de Blender se ajustó un factor de escala para que las medidas reales del marcador se correspondan con las unidades de Blender, por razones de facilidad en el manejo en Blender se estableció que la unidad de Blender corresponde a un centímetro en el mundo real.

En la Figura 1.1 se muestra la vista de *Perspectiva de usuario* en donde se puede ver el modelo del marcador importado, la cámara en el centro de coordenadas de Blender y el sistema de coordenadas interno. Se busca construir una escena mínima y controlada en la cual sólo actúa el objeto

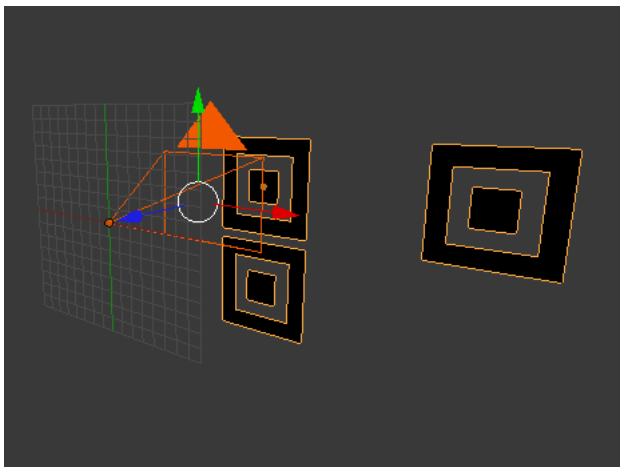


Figura 1.1: Vista de perspectiva de usuario para el modelo 3D del marcador y la cámara en la escena.

marcador. Con este fin es que se decide por una escena que contenga únicamente el marcador y una cámara. No se añaden luces ya que el efecto de estas en el marcador para alguna pose puede resultar en artefactos de iluminación indeseados. La no inclusión de luces en las imagen está ligada a que el modelo 3D del marcador es en realidad hueco en las zonas blancas y negro en las zonas negras por lo tanto el negro permanece negro y la parte hueca asume el color del fondo. En otro caso se podría utilizar algún tipo de iluminación de ambiente que sea pareja en todas las direcciones y lograr un efecto similar.

Blender utiliza el sistema de coordenadas que se ve en la Figura 1.1 en donde los ejes x, y, z se corresponden con los ejes de color rojo, verde y azul respectivamente y forman un sistema de coordenadas siguiendo la regla de la mano derecha, lo que se dice en lenguaje técnico: *right handed*.

Finalmente, la escena conteniendo el modelo 3D del marcador se resume en el archivo: **Marker.blend**.

1.1.1.2. Ajustes de escena y parámetros: GUI

El software Blender permite el ajuste de la escena y los parámetros y características asociados al render, la cámara y el objeto mediante su interfaz gráfica. Estos tienen como resultado final la imagen renderizada. Los parámetros y ajustes que fueron realizados y resultan ser los más relevantes para la reproducción del proceso de renderizado del marcador se explican a continuación.

- **Mundo y fondo**: Los ajustes realizados al mundo y fondo se explicaron en cierta medida anteriormente. Al no utilizar iluminación en la escena es importante definir el color del fondo ya que los huecos del modelo 3D asumirán ese color. Los ajustes realizados se muestran en la Figura 1.2(a) y consisten principalmente en definir el fondo de color blanco.

- **Parametros de Render**: Los principales parámetros ajustables en cuanto a las propiedades de render son el tamaño de imagen y el porcentaje de escala sobre el tamaño de imagen, el cual por defecto en proyectos nuevos se ubica en 50 % y en este caso interesa que sea 100 %. Estos ajustes se pueden ver en la Figura 1.2(b).

- **Parámetros de Cámara**: Sobre las propiedades ajustables de la cámara se encuentra un parámetro de interés como es el campo visual, *field of view*, o fov. Adicionalmente el *Shift* representa el corrimiento del centro de imagen de la misma y también se puede ajustar el *near* y el *far*. Los ajustes realizados se pueden ver en la Figura 1.2(c).

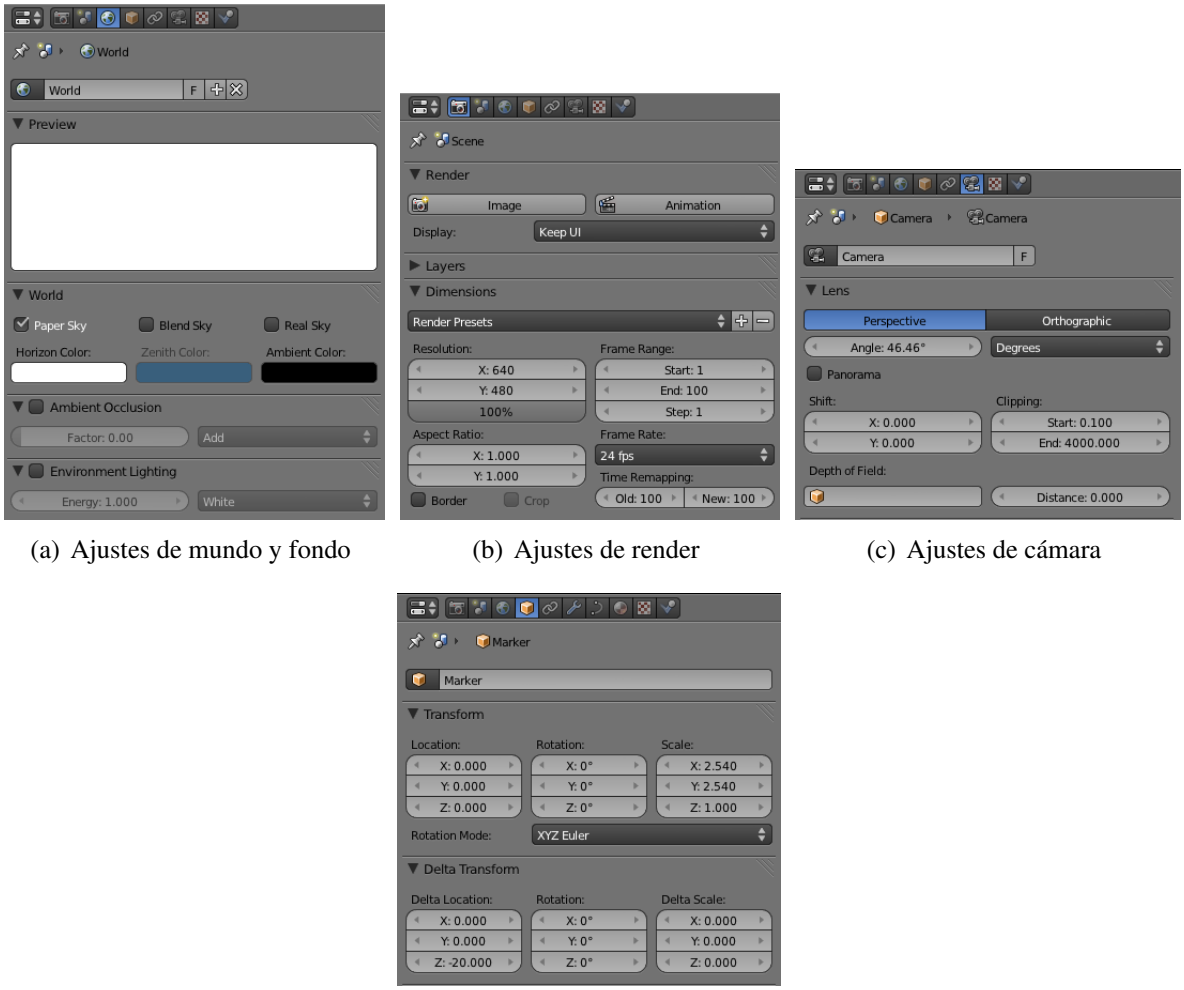
- **Parámetros de Objeto**: Las propiedades ajustables de mayor interés sobre el objeto se refieren a su pose. La locación y rotación del objeto pueden ser asociadas a la pose del objeto respecto a la cámara si se asume una interacción del tipo cámara fija, objeto movil. Se aclara aquí que un ajuste idéntico se puede realizar sobre el objeto cámara. La cámara se fija para todos los casos en el origen de coordenadas de Blender con orientación nula en las tres direcciónes. Esto resulta en una cámara alineada con el eje z que mira hacia los z negativos. Volviendo a los ajustes del objeto marcador, se presentan los ajustes de escala ya mencionados en el ajuste de unidades del modelo 3D y también un factor relevante como es el orden de las rotaciones respecto al tipo de rotación: *XYZ Euler*. Todos los ajustes aquí descritos para el objeto marcador se pueden ver en la Figura 1.2(d).

Vale la pena observar que los ajustes de Render y Cámara constituyen una forma de establecer los **parámetros intrínsecos** de la cámara. Mientras que los ajustes de Objeto para la Cámara y el Marcador representan la **pose del objeto** en el modelo de vinculación adoptado. Se aclara que los **parámetros extrínsecos** de la cámara respecto a las coordenadas de Blender es nula en rotación y traslación.

1.1.1.3. Ajustes de parámetros: Python scripting

Python es un lenguaje de programación interpretado, interactivo y orientado a objetos. Incorpora módulos, excepciones, *tipeo* dinámico, tipos de datos de muy alto nivel y clases. Python logra combinar notablemente poder con claridad en la sintaxis [**?**]. La mayoría de las áreas de Blender pueden ser programadas mediante *scripts*, incluyendo Animación, Renderizado, Importación y Exportación, Creación de objetos y Tareas repetitivas. Para la interacción con Blender se hace uso de una API "ajustada a medida".

La API de Blender para Python se encuentra empaquetada en el módulo `bpy`. Mediante este módulo se puede acceder a los objetos y sus propiedades de la escena. De esta forma se puede de forma programática ajustar las propiedades y parámetros explicados anteriormente. Por lo tanto mediante él intérprete de comandos de Blender se puede prescindir de la interfaz gráfica de Blender. En la



(a) Ajustes de mundo y fondo (b) Ajustes de render (c) Ajustes de cámara

(d) Pose del objeto

Figura 1.2: Ajustes en la escena en Blender para realizar el render.

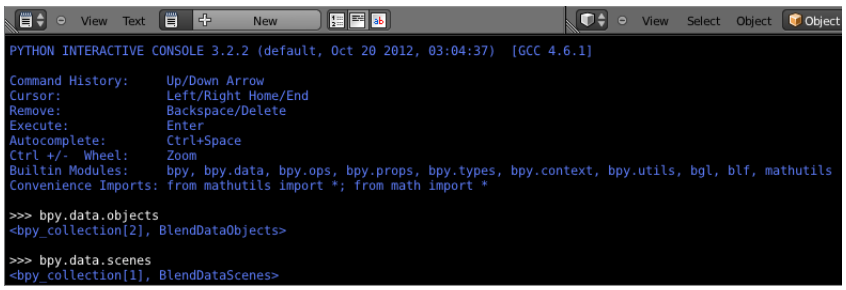


Figura 1.3: Consola interactiva de Python incrustada en Blender.

Figura 1.3 se muestra un *screenshot* de la consola interactiva de Python en Blender. A continuación se muestran las principales funcionalidades que se utilizaron en la manipulación de Blender mediante la consola interactiva. Estas se van ejecutando en serie hasta la última orden que realiza el renderizado y guarda la imagen.

- Obtener referencias a la escena y al marcador.

```
>>> scene = bpy.data.scenes["Scene"]
>>> marker = bpy.data.objects["Marker"]
```

- Seleccionar modo de rotación y ubicar y orientar cámara como se desea.

```
>>> scene.camera.rotation_mode = 'XYZ'
>>> scene.camera.rotation_euler = [0,0,0]
>>> scene.camera.location = [0,0,0]
```

- Asignación de valores asociados a **parámetros intrínsecos** de la cámara.

```
>>> scene.render.resolution_x = width
>>> scene.render.resolution_y = height
>>> scene.camera.data.angle = fov*(pi/180.0)
```

- Asignación de valores asociados a la **pose** del marcador.

```
>>> marker.rotation_mode = 'XYZ'
>>> marker.rotation_euler = [rx,ry,rz]
>>> marker.location = [tx,ty,tz]
```

- Asignación de nombre de imagen y ejecución de renderizado para guardar la imagen.

```
>>> bpy.data.scenes["Scene"].render.filepath = str(fname)
>>> bpy.ops.render.render( write_still=True )
```

En la Figura 1.4 se muestra el resultado de esta secuencia de comandos para los valores parámetros de cámara y render: `width = 480`, `height = 360`, `fov = 45.5` y la pose del objeto como `(rx,ry,rz) = (0,30,0)` y `(tx,ty,tz) = (0,0,-100)`.

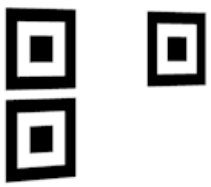


Figura 1.4: Imagen del render realizado con Blender. El mismo resultado se puede lograr mediante interfaz gráfica o mediante la consola interactiva de Python de Blender.

1.1.1.4. Ejecución por consola de comandos

La ejecución de Blender por consola de comandos es útil en situaciones en donde se quieren realizar acciones como el Renderizado remotamente sobre una computadora potente que actúe de servidor. También puede ser utilizado para realizar acciones repetitivas sobre alguna escena o modelo y esta es la utilidad que se le dio.

Asumiendo que se tiene acceso a la aplicación Blender como el ejecutable `blender`, el mismo puede ser ejecutado desde consola por el comando:

```
> blender
```

Esta orden da lugar a la ejecución de Blender con su interfaz gráfica. La opción `-b` o `--background` permite la ejecución de Blender sin interfaz gráfica. Si se quiere utilizar este modo se debe especificar el nombre del archivo a cargar, en este caso **Marker.blend**:

```
> blender -b "Marker.blend"
```

Esto no produce ningún resultado visible, solamente abre Blender sin interfaz grafica, carga el modelo y se cierra. Es una prueba del tipo *Hello World* y un chequeo de sanidad para comprobar que todo anda bien, en otro caso se mostrará algún mensaje de error de Blender.

Adicionalmente se pueden añadir las funcionalidades de *scripting* en Python previamente explicadas se pueden agrupar en un *script* para la ejecución mediante consola de comandos sin interfaz gráfica. La opción `-P` o `--python` seguida del nombre del archivo del script permite la ejecución del mismo dentro de la ejecución de Blender.

```
> blender -b "Marker.blend" -P "moveMarker.py"
```

La ejecución de la orden de arriba produce una imagen renderizada del marcador. El script **moveMarker.py** agrupa los comandos previamente explicados para su ejecución desde la consola interactiva integrada en Blender. Para que el script pueda hacer uso de lo módulos de Blender se debe importar el módulo, `bpy`, al comienzo del script . También se importa el módulo `sys` para la posibilidad de tener un script de Python con argumentos de línea de comando.

```
import bpy
import sys
...
```

Finalmente se pueden agregar argumentos de línea de comando sobre el script de Python dejando accesibles las variables de interés para el proceso de renderización serial. Por ejemplo:

```
> blender -b "Marker.blend" -P "moveMarker.py" -- <tx> <ty> <tz>
  <rx> <ry> <rz> <fov> <heigh> <width> <fname>
```

En donde el `<fname>` determina el nombre de la imagen producto del renderizado y `--` especifica que el script de Python contiene argumentos. El parseo de estos argumentos de Python se debe hacer dentro del script en el cual se deben realizar las asignaciones y las conversiones a tipos de datos apropiadas.

1.1.1.5. Comentarios sobre el script de Python

A continuación se realizan algunos comentarios respecto al desarrollo del script en Python para lograr coherencia entre los sistemas de coordenadas y dimensiones que se utilizan entre el algoritmo de estimación de pose y Blender.

Debido a que el sistema de coordenadas que utiliza Blender difiere con el adoptado por el algoritmo de estimación de pose es que debe ajustar en algún punto los ejes de alguno de los dos sistemas para trabajar de forma coherente. Lo razonable y el enfoque que se eligió es el de ajustar los ejes que se van a utilizar en Blender dentro del script de Python. Por lo tanto se aplica una transformación sobre los valores de las rotaciones y traslaciones correspondientes a esta convención de ejes:

$$x_{blender} = x_{posit}$$
$$y_{blender} = -y_{posit}$$
$$z_{blender} = -z_{posit}$$

Esto se traduce en invertir el sentido de las rotaciones y el valor de las traslaciones de los ejes z e y.

Otra aclaración importante refiere a que las unidades de Blender fueron ajustadas para corresponder con centímetros mientras que a lo largo del proyecto las unidades utilizadas para las coordenadas del modelo 3D y del mundo son los milímetros. Por esto es que las traslaciones dentro del script son ajustadas para corresponder con centímetros.

Con estos dos ajustes la realización de imágenes sintéticas con el entorno desarrollado es *transparente* al usuario ya que si este es coherente con las convenciones tomadas para el algoritmo de estimación de pose no se tendrán que realizar reajustes.

1.1.2. Ejecución Batch: Matlab

Se generó un entorno de ejecución secuencial tipo batch en Matlab en donde se ejecuta Blender, en su modo de ejecución por consola de comandos con script de Python con argumentos , mediante el uso de la función de Matlab `system` la cual realiza una llamada al sistema.

El entorno desarrollado para generación automática de juegos de imágenes sintéticas tipo batch, que utiliza Matlab como interfaz hacia el usuario, consiste en varios pasos:

- **Configuración de Cámara**: Se fijan los parámetros intrínsecos y extrínsecos de la cámara así como el tamaño de imagen.
- **Setup general**: Se definen las rutas a los archivos del modelo y script de Blender así como las rutas a las carpetas para las imágenes de salida y se genera esta estructura de carpetas en caso de ser necesario. Se establece el identificador sobre el juego de poses a renderizar.
- **Generar Poses**: Se generara el juego de poses correspondiente al identificador del Setup. Las definiciones sobre los juegos de poses se realizan previamente. Por ejemplo un juego de poses puede consistir en variar el ángulo de rotación respecto del eje x manteniendo todos las otras rotaciones fijas y la traslación en en un $z = 1000mm$. Existe la posibilidad de agregar ruido al juego de poses.
- **Ejecución de Blender**: En base a la Cámara, el Setup y las Poses se ejecuta para cada pose una llamada al sistema con el comando de Blender de línea de comandos con script de Python

y argumentos de línea. El valor de los argumentos se ajusta para cada ejecución de forma de lograr el resultado esperado. Se genera en la estructura de carpetas un juego de imágenes de salida que corresponden a una imagen sintética del marcador.

- **Registro**: Se guarda un archivo de texto con las poses utilizadas y otro con las configuraciones de las cámara.

1.1.3. Resumen

En esta sección se describió el proceso general para la generación de imágenes sintéticas para la validación de los algoritmos desarrollados y utilizados en el proyecto. Se explicó qué ajustes son de interés a realizar en Blender mediante su interfaz gráfica así como a través de la consola de comandos interactiva de Python. Se describió el proceso para realizar renders desde consola de comandos mediante la utilización de un script en Python y adicionalmente mediante la inclusión de argumentos de línea de comandos de forma de parametrizar el proceso. Finalmente se describió el entorno desarrollado en Matlab para automatizar este proceso.

1.2. Herramientas de Hardware y Software de Apple Inc.

1.2.1. *SoC*, CPU y GPU de plataformas Apple Inc.

Uno de los puntos a evaluar de las plataformas de Apple Inc. es el **SoC**, que refiere a *System on Chip* por sus siglas en inglés. *System on Chip* es un concepto de los sistemas embebidos que refiere a la integración de todos los componentes de una computadora u otros sistemas electrónicos, en un solo circuito integrado. Básicamente, un *SoC* y un microcontrolador cumplen funciones similares pero el *SoC* forma parte de una evolución de los microcontroladores, siendo de una complejidad mayor e integrado en un tamaño muy reducido buscando poco consumo y eficiencia de costos. Así entonces un *SoC* puede estar conformado por un microcontrolador y hardware adicional como procesadores de señal y bloques de memoria [**?**].

En la Figura 1.5 se ilustran los dos tipos de *SoC* de los dispositivos de la Tabla **??**: Apple A4 y Apple A5. Algunos dispositivos que no figuran en la tabla como el *iPhone 5* o el *iPad 4* usan *SoCs* más recientes como el Apple A6 o Apple A6x respectivamente. La familia de *SoCs* Apple Ax, es la que la mencionada firma utiliza en todas sus plataformas, inclusive en el *Apple TV* y es fabricada por Samsung. Estos *SoCs* se caracterizan por utilizar CPUs de arquitectura ARM, en su mayoría ARMv7 y GPUs de PowerVR de la línea SGX.

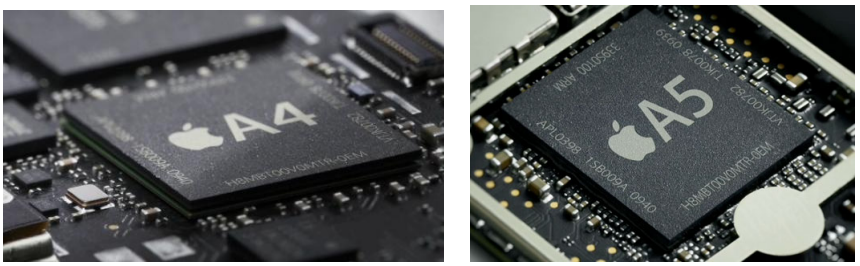


Figura 1.5: *System on Chip*: SoC.

La arquitectura ARM incorpora algunas características de la arquitectura RISC (*Reduced Instruction Set Computer*) como el hecho que las operaciones sean llevadas a cabo sobre un conjunto de registros a tales efectos y no sobre la memoria directamente. Otra característica que tiene de RISC es que tiene un modo simple de direccionamiento donde las direcciones son también guardadas sobre registros destinados a tales efectos [**?**]. La mayoría de los procesadores están hechos con un ancho de palabra de 32-bit salvo el reciente ARMv8 que incorpora la posibilidad de utilizar ambos anchos de palabra: 32-bit o 64-bit. Por más información sobre esta arquitectura referirse a la web de la empresa [**?**].

Por su parte las GPU utilizadas en los *SoCs* de la serie Apple Ax, son GPUs de PowerVR, una división de la firma Imagination Technologies que desarrolla hardware y software para *rendering* 2D y 3D, procesamiento de imágenes y codificación [**?**]. La función que tiene la GPU es asignar a cada pixel de la pantalla su color para cada cuadro. En particular, estas GPU implementan un concepto innovador de *renderizado* que mejora notoriamente la performance de los gráficos: *Tile-Bassed Deferred Rendering* (TBDR) [**?**]. Este concepto aprovecha la independencia de áreas alejadas de la pantalla y de la correlación de píxeles cercanos y divide la pantalla en *tiles* o baldozas. A cada *tile* se le asocia un procesamiento paralelo y con esto se mejora notablemente la performance respecto al método tradicional: *Immediate mode renderers* (IMRs), que procesa la pantalla completa. Las

imágenes *renderizadas* están construidas por triángulos (polígonos), por lo que uno de los indicadores fundamentales para evaluar la performance de una GPU es la cantidad de triángulos (polígonos) que es capaz de procesar por segundo. En [**?**] se puede ver un anásis interesante que compara entre otros, a los dos tipos de GPU que se presentaron en la Tabla **??**: SGX535 y SGX543MP2. En dicho análisis se muestra por ejemplo que en el mejor de los casos la GPU SGX535 fue capaz de procesar 8.69 millones de triángulos por segundo frente a los 29 millones procesados por la GPU SGX543MP2.

1.2.2. Sistemas Operativos

Para poder desarrollar aplicaciones sobre dispositivos móviles de la firma Apple Inc. es necesario contar con computadoras que corran el sistema operativo **Mac OS X**. Esto puede ser llevado a cabo, ya sea adquiriendo plataformas de desarrollo de la mencionada firma o creando máquinas virtuales que corran dicho sistema operativo. Para la segunda opción (la más económica pero con ciertas dificultades de performance), es necesario que la computadora cuente con virtualización de hardware. Se comenzó trabajando de esta manera hasta el momento de adquirir plataformas de desarrollo que contaran con Mac OS X en forma nativa.

Mac OS X refiere a la versión número 10 (en números romanos) de una serie de sistemas operativos que comenzaron a desarrollarse en la década de los 80 (Mac OS 1 data del año 1984). En los últimos 28 años se han ido sucediendo nuevas versiones que han ido mejorando características en la estructura de datos con la incorporación de la jerarquía de archivos en Mac OS 3 por ejemplo, en la búsqueda de archivos, con la simultaneidad de tareas, multiplicidad de usuarios o incluso con el énfasis en la interfaz de usuario por mencionar algunas características importantes en la evolución de esta familia de sistemas operativos. Dentro de Mac OS X existen distintas versiones, siendo la más actual la Versión 10.8: “Mountain Lion” lanzada durante 2012.

Por su parte todas las plataformas móviles de Apple Inc. corren otro sistema operativo de código cerrado: **iOS**. Originalmente llamado así por ser el utilizado por la plataforma *iPhone*, este sistema operativo está también en las plataformas *iPad*, *iPod Touch* y *Apple TV* en todas sus versiones. La versión más reciente de este SO es el iOS 6.1.

Una de las grandes innovaciones de estas plataformas es el hecho de poder desarrollar aplicaciones y correrlas en el propio dispositivo (por supuesto también sucede lo mismo en el mundo Android). Para poder lograr esto, es necesario como se ha dicho, contar con una máquina que corra Mac OS X y contar con el IDE apropiado llamado **Xcode**. Este entorno de desarrollo y su lenguaje se explican en la Sección 1.2.3.

1.2.3. Objective-C

El lenguaje que fue elegido por Apple Inc. para desarrollar sobre plataformas móviles es Objective-C. Este lenguaje fue desarrollado en la década de 1980 como un superconjunto de C orientado a objetos. Se trata entonces de una extensión del standard ANSI C que incorpora un modelo orientado a objetos basado en **Smalltalk** [**?**]. Una de las diferencias sustanciales del modelo orientado a objetos de Objective-C respecto a otros lenguajes como Java o C++, es la manera en que se invocan los métodos (procedimientos) de las instancias de clases. En objective-C esta invocación se da enviando mensajes, algo que se hereda de Smalltalk. Así entonces para invocar un método se procede con el siguiente código por ejemplo:

```
[receiver message];
```

Donde *receiver* es un objeto que recibe un mensaje (acción) *message* a realizar. Esta acción puede tener parámetros asociados, como por ejemplo el siguiente código real:

```
[myRectangle setWidth:20.0];
```

Esta diferencia conceptual de utilizar mensajes se representa en el hecho de que en tiempo de compilación estos mensajes no son más que una etiqueta y no están asociados al bloque de código como es el caso de Java o C++. Entonces es factible que suceda el hecho de que ese mensaje o método no esté implementado por esa clase y recién en tiempo de ejecución es que sucede el error al sustituirse esa etiqueta por un código inexistente, pues un objeto recibe un mensaje para realizar un método que no está dentro de su repertorio. Para esto es que en la documentación de Apple Inc. se recomienda utilizar ciertos trucos para garantizar que el objeto que reciba el mensaje sea capaz de responder correctamente, como por ejemplo consultando primero si es capaz de realizar dicha acción y luego en caso de poder, realizar dicha acción.

Otro detalle a destacar es que este lenguaje, al igual que Java, también soporta la herencia múltiple. Esto es, dado un conjunto de métodos que son comunes a un conjunto de clases pero que no llegan a tener un lazo tan fuerte como para estar jerárquicamente relacionadas con una superclase común, se puede generar una clase abstracta cuyos métodos sean implementados por más de una clase sin necesidad de generar ese vínculo fuerte que es la herencia. Así como en Java existen las interfaces, que hacen esto posible, en Objective-C existen los protocolos. Existen protocolos formales e informales y con métodos obligatorios de implementar y otros opcionales. Una clase que implemente un protocolo dado tiene que tener dentro de su encabezado declarado el nombre del protocolo. Esto es:

```
@interface ClassName : ItsSuperclass < protocol list >
```

Hay otras particularidades del lenguaje pero que no van más allá de la sintaxis como los métodos de clase y los métodos de instancia, como los métodos *get* y *set* para acceder y configurar atributos (propiedades) de los objetos, como la notación de *import* en lugar de *include* para quienes están acostumbrados a C, y así varios detalles más. Sin embargo más allá de estas y otras diferencias y particularidades resulta un lenguaje relativamente ágil y dentro de todo sencillo de aprender para quien tiene ya un conocimiento de otros lenguajes orientados a objetos.

1.2.4. Xcode: Herramientas y Librerías

Como se dijo anteriormente el entorno de desarrollo de aplicaciones típico es Xcode, el cual es gratuito y permite compilar código C, C++, Objective-C, Objective-C++, Java y AppleScript. Xcode integra en una sola interfaz todo lo que involucra código, diseño de interfaz de usuario (**Interface Builder**) y *debugging*. También viene con un conjunto herramientas útiles para evaluar la *performance* de la aplicación en distintos aspectos que se llama **Instruments**. Por otra se tiene el iOS SDK que cuenta con un conjunto importante de *Frameworks* entre los cuales se encuentran **Cocoa** y **Cocoa Touch** que proveen las herramientas necesarias para desarrollar aplicaciones para Mac OS X e iOS respectivamente [**?**].
El SDK de iOS se puede clasificar en cuatro grandes capas según el nivel de abstracción: Cocoa Touch, Media, Core Services y Core OS como se muestra en la Figura 1.6. Así entonces, dentro de cada capa existen distintos *Frameworks* según la funcionalidad. A continuación se explica el rol que cumple cada capa, los distintos *Frameworks* que tiene cada una y para qué sirven.

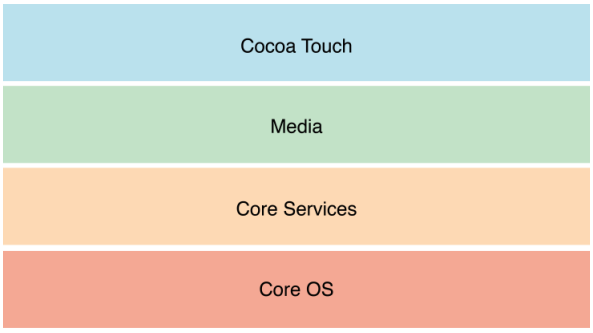


Figura 1.6: Capas del SDK de iOS

1.2.4.1. Cocoa Touch Layer

Cocoa Touch es la capa de más alto nivel del SDK de iOS y es la encargada de proveer al desarrollador de ciertos *Frameworks* que permitan lograr distintas tecnologías como la posibilidad de multitarea, el ingreso de órdenes a la aplicación a través de la pantalla táctil, notificaciones y alertas, preservación del estado de la aplicación al salir de la misma, reconocimiento de gestos en la pantalla y otro tipo de funcionalidades de alto nivel. Permiten al desarrollador, sin tener que involucrarse demasiado a bajo nivel, el acceso a determinados servicios que ya están resueltos en forma bastante modular. Cocoa Touch está basado en la arquitectura **Modelo-Vista-Controlador**, en el que se separa en tres áreas distintas el modelo de la información, la interfaz de usuario y el conjunto de reglas que negocian la presentación de la información en base a la interacción con el usuario. Así pues, el usuario y una aplicación se podrían considerar dos sistemas que interaccionan. Por su parte el usuario tiene como entrada la vista de la aplicación y como salida tiene su respuesta a esta entrada, generando efectos sobre el control de la aplicación. Por otro lado la aplicación tiene como entrada las órdenes dadas por el usuario que tienen efectos sobre el modelo de la información y este sobre la vista, quien resulta ser la salida de la aplicación. Esta interacción se puede ilustrar con la Figura 1.7. Como se dijo, dentro de Cocoa Touch, existen distintos *Framewokrs* enfocados en

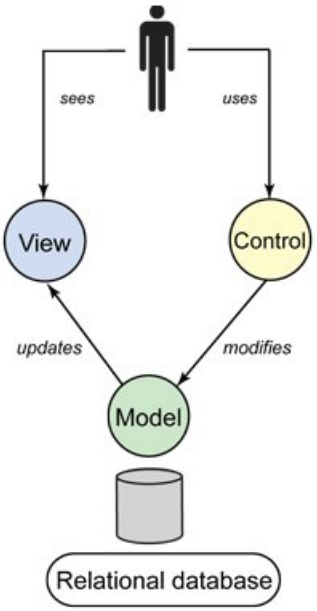


Figura 1.7: Interacción entre las tres partes del MVC

permitirle al desarrollador resolver en alto nivel distintos aspectos. Los mismos son los siguientes:

(1) Address Book UI Framework

(2) Event Kit UI Framework

(3) Game Kit Framework

(4) iAd Framework

(5) Map Kit Framework

(6) Message UI Framework

(7) Twitter Framework

(8) UIKit Framework

Varias de estas librerías no fueron utilizadas en el presente proyecto dada su función específica. Sin embargo hay una en particular que tiene bastante importancia y que permite la mayoría de las funcionalidades básicas que toda aplicación tiene. Se trata del **UIKit**, encargado de gestionar la aplicación, su interfaz de usuario y gráficos, encargado soportar eventos frente al toque de la pantalla, de manejar sensores como el acelerómetro y giroscopio, y de tener acceso a la cámara y galería de fotos entre lo más importante a destacar. El soporte de la multitarea y de **Storyboards** también está a cargo de este *Framework*.

Hay funcionalidades que han ido cambiando con las distintas versiones de iOS. Una de ellas y quizá una que ha generado bastantes diferencias respecto a versiones anteriores a iOS 5, es esta última, el Storyboard, una herramienta muy útil de programación gráfica, que permite generar instancias de clases y vínculos entre las mismas en forma visual a la vez de ser una contraparte de interfaz de usuario. Con una biblioteca de objetos disponibles, listos para ser usados, mediante el uso de Storyboard se hace accesible con algunas horas de dedicación implementar aplicaciones sencillas. Esta herramienta vino para sustituir los archivos *.nib* que permitían diseñar la interfaz pero no tantas funcionalidades programáticas como el Storyboard. En particular éste último permite agregar la funcionalidad de *segues*, encargados de vincular un *ViewController* con otro. Este tipo de diferencias vinieron con la idea de evitar la necesidad de implementar ciertos bloques de código en forma repetitiva. Un Storyboard luce como en la Figura 1.8.

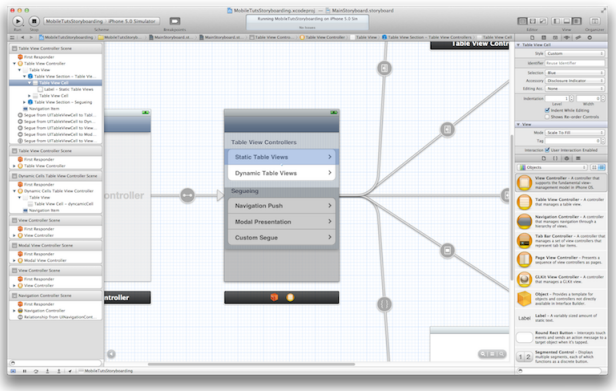


Figura 1.8: Ejemplo de Storyboard.

Si bien se podría extender bastante más la explicación sobre los detalles de Cocoa Touch, a los efectos del presente proyecto, no es de tanta relevancia excederse en este punto.

1.2.4.2. Media Layer

Media Layer es la capa encargada de gestionar correctamente elementos multimedia y es posible distinguir tres grandes grupos que engloban distintos *Frameworks*: Gráficos, Audio y Video.

Dentro de las tecnologías más destacadas está todo lo vinculado a **gráficos** 2D y 3D, dentro de lo que se puede incluir algunos *Frameworks* bastante utilizados en el presente proyecto, tales como: **Core Graphics**, muy utilizado para dibujos 2D, **Quartz Core**, quien contiene las herramientas necesarias para interactuar con otro *Framework* para animación de vistas, de una capa de más bajo nivel como *Core Animation*, que es comentado en la Sección 1.2.4.3. También es parte de lo vinculado a gráficos, el *Framework* **Core Image**, conteniendo lo vinculado a procesamiento de imágenes a través filtros que utilizan directamente la lu unidad de procesamiento de gráficos (GPU) y otros dos *Frameworks* bastante importantes en lo que respecta a *rendering* como **Open GL ES** y **GLKit** (utilizado por el motor de juegos *Isgl3d* entre otros).

Por otra parte hay otra gran familia de *Frameworks* dentro de Media Layer que apunta a resolver todo lo vinculado al manejo de audio, ya sea de grabación como procesamiento y reproducción de alta calidad. Existen algunos SDK como **iSpeech** o **Dragon Mobile** que resuelven de manera similar al proyecto Siri, el procesamiento de la voz humana reconociendo palabras e interpretando, que utilizan algunos de los *Frameworks* de procesamiento de audio de esta familia.

En cuanto a lo vinculado al manejo de video, como parte de esta capa, se tienen dos *Framework* importantes con distintos niveles de abstracción: **MediaPlayer** y **AVFoundation** [**?**]. También existen otros *Frameworks* fuera de esta capa que son capaces de manejar video como la clase UIImagePickerController (muy utilizada en el proyecto) del mencionado UIKit. En la Figura 1.9 se esquematiza el nivel de abstracción de los *Frameworks* de las distintas capas que son capaces de manejar multimedia. Con MediaPlayer es posible reproducir audio y video muy fácilmente en

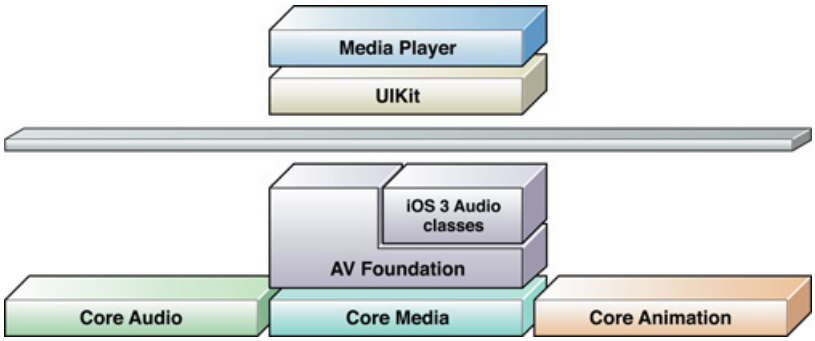


Figura 1.9: Frameworks de las distintas capas para manejo de video

determinada área de la pantalla ya sea desde un URL o de un archivo, es posible mostrar o no los elementos de control del video así como también controlar, volumen y tamaño de la pantalla. Por su parte, con *AVFoundation* es posible capturar con la cámara, reproducir, editar y procesar audio y video. Es posible implementar ciertos protocolos que hace de esto algo relativamente sencillo.

1.2.4.3. Core Services

Core Services es la capa de más bajo nivel de iOS y contiene los elementos fundamentales sobre los que se construyen las capas superiores. Es posible que al comenzar a programar para iOS no se tenga mucha interacción con esta capa pero sin embargo existen algunos conceptos importantes de esta capa que sí vale la pena mencionar dado que en el presente proyecto se tuvieron que entender y discutir. Una de ellas es la *Automatic Reference Counting* o **ARC**. Esta funcionalidad compete a la reserva y liberación de memoria por parte de los objetos. La idea básica es lograr que el uso de

memoria sea el mínimo posible, logrando que los objetos existan en la medida que son necesarios y que su memoria sea liberada ni bien sea posible. Típicamente, al crear una instancia de un objeto se

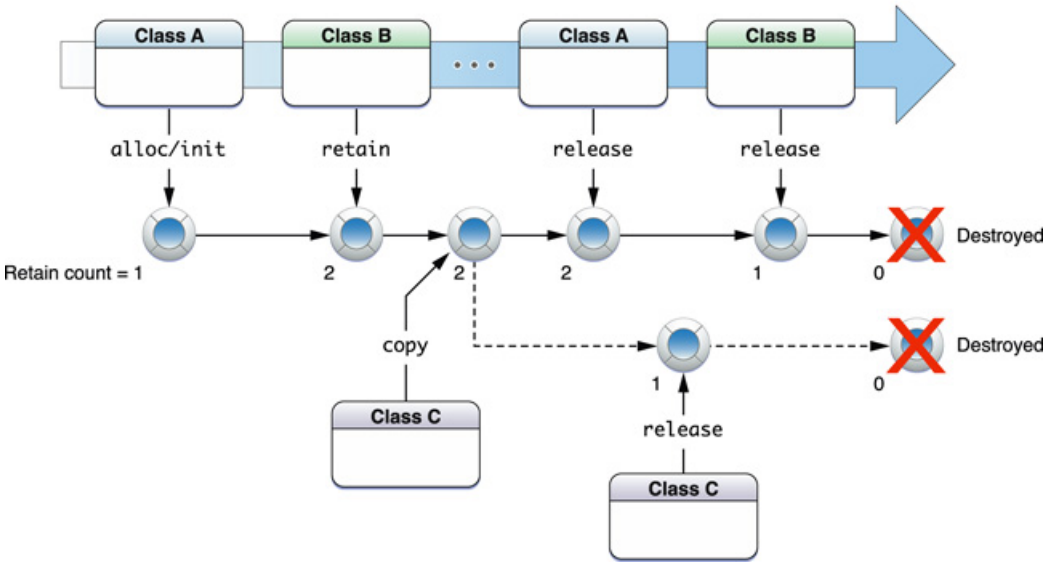


Figura 1.10: Ciclo de vida de objetos, Manual-Retain-Release.

incrementa un contador y al liberar se decrementa y entonces se tiene cierto control sobre la reserva y liberación de memoria en base al contador. Sin embargo, la liberación de memoria reservada por objetos queda bajo la responsabilidad del desarrollador y en casos de un código complejo puede llegar a ser habitual olvidarse de la liberación de memoria. Lo anterior refiere a una gestión manual de la reserva y liberación conocido como *manual retain-release* se puede ver esquemáticamente en la Figura 1.10. Para no tener que enfrentar este tema y poder instanciar clases sin tener presente la posterior liberación de memoria se utiliza ARC. Esta funcionalidad evalúa el ciclo de vida de los objetos y agrega código en tiempo de compilación en caso de considerarlo necesario. Es bueno aclarar que esto refiere a memoria reservada pura y exclusivamente por objetos, es decir mediante *alloc*. En caso de tratarse de memoria reservada para variables de lenguaje C (*malloc*), es necesario proceder de igual manera que en dicho lenguaje, liberando la memoria mediante un *free* [**?**] [**?**].

Además del ARC, Core Services permite el manejo de archivos XML y manejo de base de datos SQL así como también la protección de datos cuando el dispositivo está bloqueado entre otros servicios importantes. Tiene varios *Frameworks* como **Core Media** que logran un nivel aún más bajo que AVFoundation para el manejo de multimedia, **Quick Look** para las vistas previas de archivos, **Social** que viene a suplantar el *Framework* para la utilización de Twitter que existe en iOS 5 y extiende la gestión para otras redes sociales, **Core Motion** para el manejo de sensores como el acelerómetro y el giroscopio, **Core Telephony** para el manejo de la información de red del dispositivo como elemento de la red de telefonía, **CFNetwork** para el manejo de protocolos de red como http, https, ftp y resolución de servidores DNS, entre otros *Frameworks* importantes dentro de la capa.

1.2.4.4. Core OS

Con esta capa de iOS en general es difícil que el desarrollador tenga que involucrarse directamente dado que es la de más bajo nivel. Salvo que se esté frente a una aplicación que requiera aspectos de seguridad o comunicación con HW externo, esta capa solamente existe para ser la base sobre la cual se desarrollan los *Frameworks* de las capas de más alto nivel. Los distintos *Frameworks*

que tiene están enfocados en resolver temas de procesamiento basados en el hardware de iOS, en comunicarse con dispositivos externos basados en iOS y de garantizar la seguridad de los datos de una aplicación.

1.2.4.5. Simulador

Uno de los detalles más importantes del entorno de desarrollo es la capacidad de simular lo que se programa antes de probarlo en un dispositivo. Esto es útil por cuestiones de seguridad e incluso permite programar sin la necesidad de contar con una plataforma. Esto existe para *Xcode* y es necesario decir que funciona muy bien, generando una representación bastante fiel de lo que sucede en el dispositivo real. La única crítica que se le podría hacer es el hecho de no contar con cámara y para el caso de aplicaciones de realidad aumentada esto es algo bastante importante. Sin embargo, sin ser eso, el simulador cuenta con conexión a internet, pantalla multitáctil, con información de GPS ingresada por el programador, acceso a la galería de fotos, capacidad de procesamiento y todas las funcionalidades que un dispositivo real tiene.

1.2.4.6. Instruments

Dentro de las herramientas que vienen con el entorno de desarrollo viene *Instruments*, un conjunto de herramientas que permiten analizar la *performance* de una aplicación para iOS o para Mac OS X desde distintos puntos de vista. Resulta muy útil pues muchas veces sucede que una aplicación compila y se ejecuta correctamente y sin embargo puede que el desarrollador no esté conforme en cuanto a los tiempos de procesamiento o el uso de memoria consumido.

Para poder hacer uso del *Instruments*, es necesario correr la aplicación en modo *Profile*. Eso despliega una ventana como la de la Figura 1.11. Allí es posible elegir dentro de cada una de las posibilidades que ofrece *Instruments*, si se quiere analizar tiempos, memoria, recursos de CPU, *multithreading* entre otros tipos de datos de interés que es posible recoger de la aplicación. También es posible elegir la plataforma, ya sea iOS, simulador iOS o Mac OS X [**?**].

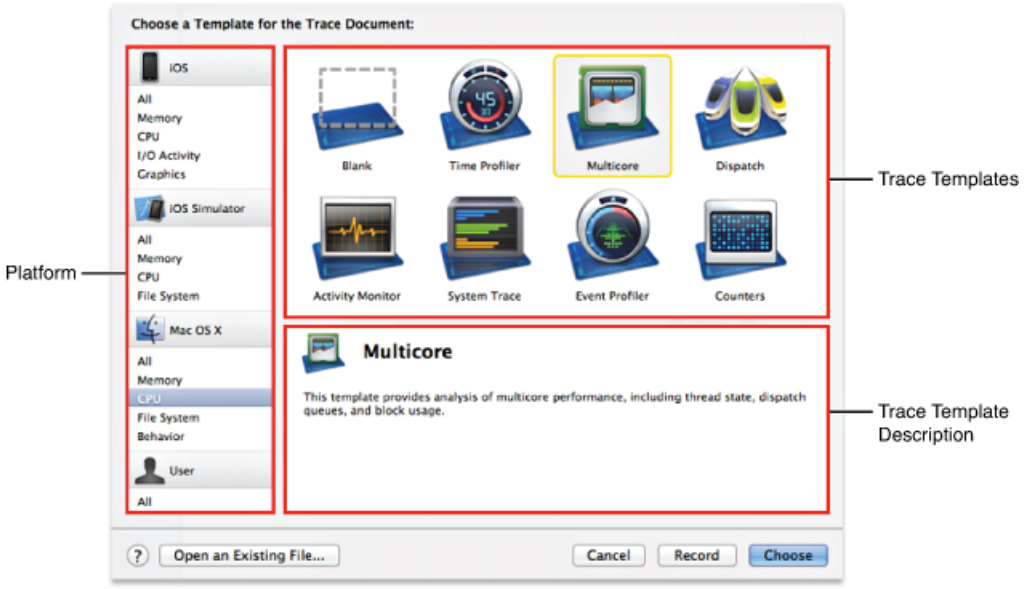


Figura 1.11: Distintas opciones de *Instruments*.

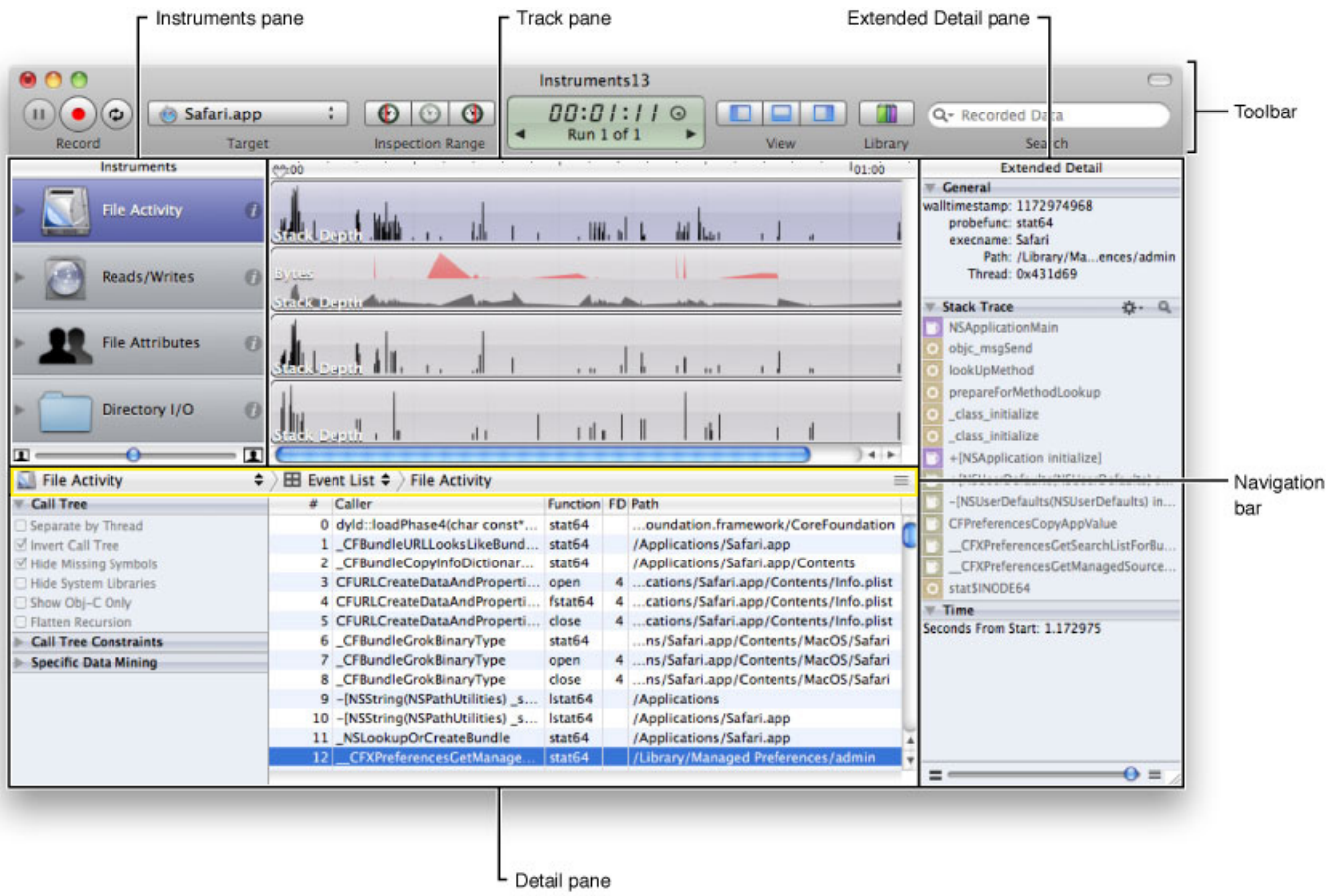


Figura 1.12: Trazado y análisis de datos recogidos.

Luego de elegir el tipo de datos a ser recolectados según lo que se busque analizar, se despliega una ventana como se ve en la Figura 1.12. Allí es necesario registrar durante varios segundos los datos mientras se corre la aplicación y luego de terminado el registro, *Instruments* dedica un tiempo a analizar los datos recolectados. En el detalle inferior, se desglozan los procesos que corre la aplicación en forma de árbol. Cuando se desea medir tiempos por ejemplo, esto resulta muy útil porque entre otras cosas se puede analizar qué porcentaje del tiempo de la aplicación es consumido por un método en particular. Esto es posible simplemente buscando dentro del árbol mencionado, al método y leyendo el valor asignado de tiempo. Para el caso de análisis de memoria también es posible identificar fácilmente en qué parte del código se está dando algún problema de reserva de memoria no liberada.

En el presente proyecto se hizo uso principalmente del **Time Profiler** que permite analizar tiempos y del **Memory Leak** que permite hacer un análisis de la reserva de memoria no liberada. Con los mismos fue posible optimizar tiempos en determinados métodos del procesamiento, así como también eliminar problemas de memoria no liberada que generaban la interrupción abrupta de la aplicación luego de llegado un cierto nivel de reserva. Esta interrupción abrupta es una forma de proteger la memoria del dispositivo y evitar que se vea afectada cierta memoria útil a otros efectos. El resto de las herramientas de *Instruments* fueron probadas pero no utilizadas en detalle para resolver problemas particulares.

1.3. Herramientas de Productividad

Además de las herramientas mencionadas propias del entorno de desarrollo fueron utilizadas algunas herramientas más generales de desarrollo de software en lo que respecta al control de versiones de código. A continuación se describen brevemente ciertos detalles de las mismas.

1.3.1. GIT

GIT es un software para gestionar código que fue desarrollado por Linus Torvalds. Entre algunos de los tantos proyectos que es gestionado por esta herramienta se encuentra el Kernel de Linux. Se basa en la idea de desarrollo colaborativo en el que existen diversas fuentes que aportan al código y es necesario llevar un control de las distintas versiones. Así entonces lo que se hace es generar un repositorio remoto donde se aloja todo el código y generar una copia local donde es posible realizar modificaciones y actualizar ese repositorio remoto. El repositorio tiene una estructura de archivos y mediante distintos comandos en terminal es posible hacer las actualizaciones. Otro aspecto a considerar es que es posible que existan distintas líneas de desarrollo simultáneas que pueden ser independientes o no. Es posible entonces crear, borrar o *converger* (*merge*) distintas líneas de desarrollo. Existen numerosas empresas que hacen uso de esta herramienta en su desarrollo como Facebook, Twitter o RedHat entre otros. Más información sobre este software se puede encontrar en [**?**].

1.3.2. GoogleCode

GoogleCode es un sitio que ofrece Google para *hosting* de proyectos, foros de discusión sobre temas de desarrollo y una API para quienes desarrollan sobre recursos de Google. Una de las características principales es que es *open-source* y que ofrece la opción de utilizar distintos software para la gestión de código como Subversion, Mercurial y Git. También ofrece otras funcionalidades como una página *wiki* y la posibilidad de vincular directamente el código con esta página.

1.3.3. Github

Github es otro sitio que ofrece *hosting* para proyectos *open-source*. Cuenta con muchas herramientas de gran utilidad y se ha vuelto uno de los sitios más populares. También utiliza el software Git y entre otras cosas permite navegar dentro de la estructura de archivos *online*, ofrece un historial para cada archivo, permite identificar qué colaborador realizó determinados cambios e incluso realizar modificaciones de archivos *online*, sin necesidad de hacer actualizaciones desde terminal.

Para el presente proyecto comenzó utilizándose GoogleCode pero luego se decidió migrar todo el respositorio a Github por contar con mayores facilidades de gestión. Todo el código generado en este proyecto, así como la documentación es *open source* y se puede bajar del repositorio git ubicado en: `https://github.com/encuadro/encuadro`.