ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

PROYECTO FIN DE CARRERA

Simulador de ondas tridimensionales para iPhone y iPad

Autor: Antonio José Martínez Sánchez

Directores: Dr. José Luis Gómez Tornero
Dr. Simon Pomeroy

Febrero 2014

Autor	Antonio José Martínez Sánchez
E-mail del autor	martinez.sanchez.aj@gmail.com
Directores	Dr. José Luis Gómez Tornero
E-mail del director	josel.gomez@upct.es; s.c.pomeroy@lboro.ac.uk
Codirector	Dr. Simon Pomeroy

Título del PFC	Simulador de Ondas Tridimensionales para iPhone y iPad
Descriptores	Ondas electromagnéticas, simulador, TLM

Resumen

Este proyecto consiste en el desarrollo de una aplicación iOS educacional para iPhone y iPad para visualizar y entender como se propagan diferentes ondas electromagnéticas tridimensionales, construida sobre la aplicación Loughborough Wave Lab iPhone e incorporando un solucionador 3D usando el método "Transmission Line Matrix", entre otras prestaciones. Se investigarán maneras para representar la resultante salida en 3D así como también la posibilidad de la adición de una realidad aumentada usando los sensores del dispositivo.

Titulación	Ingeniería de Telecomunicaciones, esp. Telemática
Departamento	Tecnología de la Información y las Comunicaciones
Fecha de presentación	2014

Agradecimientos

Un largo viaje. Este es el fin de un largo y difícil, pero magnífico viaje. Un viaje que empezaría por septiembre de 2008, y en el que han habido momentos difíciles, de tristeza y decepción, pero en el que tampoco han faltado momentos felices, llenos alegría y cargados de orgullo. Es por ello, creo, no hay mejor momento para mirar atrás, y valorar los resultados obtenidos, y por supuesto, agradecer a todas las personas que lo han hecho posible y que han ayudado a que se alcanzara.

Y es que un viaje como es una ingeniería técnica de telecomunicaciones, especializada en telemática, no es para menos. Es sin dudas, un viaje en el que no solo se aprenden y adquieren conocimientos técnicos y académicos, si no también un viaje que te ayuda a definirte como persona. Un viaje que te ayuda a crecer personalmente, y un viaje que te enseña a aprender, porque si hay algo que ha aprendido, esto es aprender. Aprender a pensar y a decidir. Y es por ello, que me gustaría agradecer a todos los profesores y profesoras que he tenido, porque cada uno de ellos han aportado ese diferente granito de arena a seguir en el camino.

Es en ese camino, entre cada grano de arena, donde han salido y aparecido los problemas y dificultades. Afrontarlos y superarlos. Es otro punto a destacar entre los aprendidos. Afrontar y superar los problemas y dificultades que salen a tu paso, nunca detenerse antes ellos, siempre avanzar. No solo he aprendido a afrontar y superar asignaturas y exámenes complicados, sino a superar esos pequeños baches y obstáculos que la vida nos pone día a día . Por todo ello, también tengo que agradecer tanto a compañeros de clase, como amigos, como a mi familia, por todos los ánimos de apoyo recibidos. Porque sin ellos tampoco hubiera sido posible. Por sus inmensas fuerzas transmitidas en cada uno de esos momentos, gracias.

En este punto, tengo que agradecer a Jose Luís Gómez Tornero, por todos sus consejos y todas las ayudas ofrecidas para la realización del proyecto de fin de carrera en la Universidad de Loughborough (Reino Unido). Y cómo no, agradecer también a mi director de proyecto en Loughborough, Simon Pomeroy, por darme la posibilidad de realizar un proyecto como este, por su ayuda a lo largo del mismo, y por adentrarme en conocimientos hasta ahora desconocidos para mí.

Esa parte, es, sin dudas, una de las partes más duras de este trayecto. Por ello, no hay suficientes palabras para agradecer a mi familia por hacer posible la realización del proyecto en una universidad inglesa, una oportunidad única en la vida, así como el enorme y necesitado apoyo transmitido en la distancia en el desarrollo del mismo. Gracias por cada videollamada que habéis realizado conmigo, gracias por cada palabra de apoyo que me habéis dado y transmitido. Para todos vosotros, muchas gracias.

En esta parte final del trayecto, tengo que agradecer especialmente a dos personas, quiénes han sido imprescindibles para que se haya hecho posible uno de los sueños de mi vida, mis padres. Gracias por haber hecho posible este viaje, gracias por ayudarme y acompañarme a lo largo del camino, gracias por estar siempre atentos y preocupados por mí, tanto en los buenos como en los malos momentos, gracias por vuestro apoyo, y sobretodo, gracias por vuestro esfuerzo. Porque aún que ha sido un gran esfuerzo para mí, se que también lo ha significado para vosotros. Por todo ello, muchas gracias papá, muchas gracias mamá.

Y como todo viaje, este también tiene un fin. Un fin, contrario a un titubeante inicio. Un fin, tras una larga distancia recorrida con paso firme, y en el que sólo queda agradecéroslo a todos.

ÍNDICE

Capítulo 1

1. Introducción y planteamiento del proyecto

Este capítulo introductorio iniciará al lector en los objetivos que se pretenden conseguir con este proyecto. Se detallarán las fases de desarrollo del proyecto así como los medios con los que se ha contado para su realización y su fiabilidad. En último lugar se dará una descripción de la estructura de este documento, aportándose una breve descripción de cada capítulo.

1.1 Introducción

Hoy en día vivimos en una sociedad donde la tecnología ha cambiado el modo de vida de las personas. Dispositivos electrónicos, sistemas electrónicos y automatizados, instrumentos y herramientas electrónicas... Todo tipo de tecnología existente con la que trabajamos, no solo son herramientas absolutamente necesarias en la vida cotidiana de nuestros días, si no que la sociedad depende totalmente de ellas. Medicina, transporte, educación, industria, telecomunicaciones.. La tecnología es usada a diario, continuamente.

Pero si hay alguna de ellas, la cuál está en pleno desarrollo, esa es la industria de las telecomunicaciones. Radio, televisión, telefonía móvil, infraestructuras inalámbricas.. Hay muchos ejemplos nacidos con esta tecnología, pero cabe hacer una mención especial a tres de ellos: Internet, cloud computing [1] y los teléfonos inteligentes o también llamados smartphones [2]. Actualmente, todo tipo de tecnología está relacionada con alguno o algunos de estos tres términos. Todo en todo dispositivo electrónico puede ser distribuido y sincronizado gracias al cloud computing en internet, instantáneamente. Los dispositivos móviles que usan internet se han convertido en una esencial herramienta para la sociedad en el día a día, de forma que vivimos enganchados a ellos por las numerosas aplicaciones, y por tanto facilidades, que nos presentan.

Por otro lado, también ha habido un fuerte y profundo auge de desarrollo en los últimos años: la tecnología 3D. Pantallas o televisiones 3D capaces de reproducir videos y juegos en tres dimensiones, la realidad aumentada o gafas 3D usando esta realidad aumentada son unos claros ejemplos de ello [3]. El mundo en tres dimensiones está acercándose poco a poco.

Por lo tanto, con esta gran dependencia de los dispositivos móviles inteligentes, así como de la invasión del mundo tridimensional, ninguna otra idea podría ser mejor para la realización del proyecto sino algo que trate acerca de todo lo anterior: una aplicación para teléfonos inteligentes que involucre estos conceptos.

Es importante también para entender la razón del proyecto la inexistencia de algún tipo de aplicación similar en estas mismas condiciones para sistemas operativos de dispositivos móviles, por lo que esta aplicación también podrá ser usada para futuras implementaciones en otras aplicaciones.

Por los motivos expresados con anterioridad, el principal objetivo del proyecto es el de desarrollar una educacional aplicación para la plataforma iOS, para visualizar y entender como se propagan diferentes tipos de ondas electromagnéticas a través de un material en tres dimensiones.

Este proyecto hace uso del método Transmission Line Matrix o método de la línea de transmisión (TLM), el cual será explicado posteriormente, como principal base teórica para la aplicación iOS . El principal objetivo del proyecto es visualizar y entender como se mueven diferentes ondas electromagnéticas tridimensionales en un material, incluyendo distintos modos de simulación, entre otras características. Está basado en la aplicación iOS Loughborough Wave Lab, modificando el modelo transmisión bidimensional a uno tridimensional, e incorporando otras características y funcionalidades. Loughborough Wave Lab es una aplicación creada por Daniel Browne en la universidad de Loughborough (Reino Unido) [4]. Ésta es una aplicación educacional para la plataforma iOS [5], incluyendo dispositivos como el iPhone, el iPad o el iPod, diseñada para ayudar a los estudiantes al entendimiento y aprendizaje sobre conceptos de ondas electromagnéticas, de forma entretenida y divertida.

De esta forma, este documento describe la teoría , tecnologías e implementación existente detrás del desarrollo de una aplicación con el método TLM en tres dimensiones para la plataforma iOS, además de una introducción a esta última. También es deliberado el planteamiento del proyecto, así como descrita una introducción al método TLM, tanto en dos dimensiones como en tres dimensiones. Además, la aplicación ha sido analizada y testeada en diferentes dispositivos, comprobando su funcionamiento. Finalmente, se han considerado mejoras y futuras optimizaciones para lograr el mejor rendimiento posible así como para mejorar, ampliar y optimizar la aplicación final, las cuales también serán detalladas.

1.2 Objetivos

Como se ha dicho en el apartado anterior, el objetivo fundamental del proyecto es el de diseñar, implementar y producir una aplicación educacional para la plataforma iOS, con el objeto de ayudar a entender y comprender la transmisión y propagación de ondas electromagnéticas en tres dimensiones de una manera gráfica, intuitiva e interactiva.

Para llevar a cabo esta tarea, se ven implícitos otra serie de objetivos a lograr, los cuales van a ser detallados a continuación:

- Aprendizaje del lenguaje y entorno de programación, así como diferentes API's y frameworks de la plataforma iOS.

En primer lugar, es necesario familiarizarse con el lenguaje de programación Objective-C, una derivación de C orientado a objetos, y usado para el desarrollo en este tipo de plataforma. Además, también se ha de conocer profundamente Xcode, la única herramienta oficial para desarrollo iOS soportada por Apple Inc [6]. Así también comprender y saber utilizar diferentes API's y frameworks que serán necesarios en la aplicación.

- Comprensión y entendimiento del código de la aplicación *Loughborough Wave Lab.*

  Como se ha dicho anteriormente, la aplicación estará basada en la aplicación mencionada antes. Por ello, se necesita estudiar la aplicación, entendiendo su funcionamiento general así como cada una de las partes que la forman para su posterior uso.

- Aprendizaje de los fundamentos teóricos usados en la anterior aplicación y los necesarios para el desarrollo del proyecto.

  Se deberán estudiar los conceptos teóricos necesarios para la implementación de la aplicación, tanto para la implementación de las ondas y sus características, así como para su propagación. Con ello, se estudiará el método TLM mencionado en el apartado anterior, así como los diferentes tipo de ondas, en los que se incluyen por ejemplo la sinosuidal y la gaussiana.

- Implementación del modelo tridimensional.

  El primer cambio obligado en la aplicación es la de implementar un modelo del TLM tridimensional para ser capaces de trabajar con ondas tridimensionales, así como crear, modificar o eliminar las diferentes partes del código de la aplicación para la correcta propagación en tres dimensiones.

- Estudio e implementación de una salida para el resultado de ondas tridimensionales.

  Otros de los objetivos necesarios para la creación de la aplicación es el estudio e investigación de métodos para visualizar la propagación de las ondas tridimensionales. Además, tras elegir el adecuado, se realizará su implementación.

1.3 Fiabilidad del Proyecto

En este punto se pretende presentar la viabilidad de la aplicación con el por qué de las tecnologías elegidas para su realización.

Aunque la aplicación Loughborough Wave Lab ha sido portada a Android [7], otro sistema operativo móvil, fue inicialmente diseñada, confeccionada y desarrollada para la plataforma iOS por su creador.

Por otro lado, y aunque existen algunas aplicaciones 3D similares a la desarrollada y con el mismo fundamento para ordenadores, se ha podido observar que no hay ninguna aplicación similar para dispositivos móviles, aparte de la antes mencionada en las casi 800.000 aplicaciones disponibles

para descargar en ambas plataformas, tanto en iOS como en Android. Es decir, todavía no se ha implementado ninguna aplicación parecida, debido a la poca potencia del hardware en estos dispositivos frente a computadores.

Según un informe de la firma Centrix System , sobre el uso por las empresas de todo el mundo de los dispositivos móviles pertenecientes a las tres grandes plataformas actuales, iOS, Android y Windows Mobile, claramente determina que el 58% de las empresas prefieren utilizar productos de la plataforma iOS, tal y como se ve en la siguiente estadística [8].

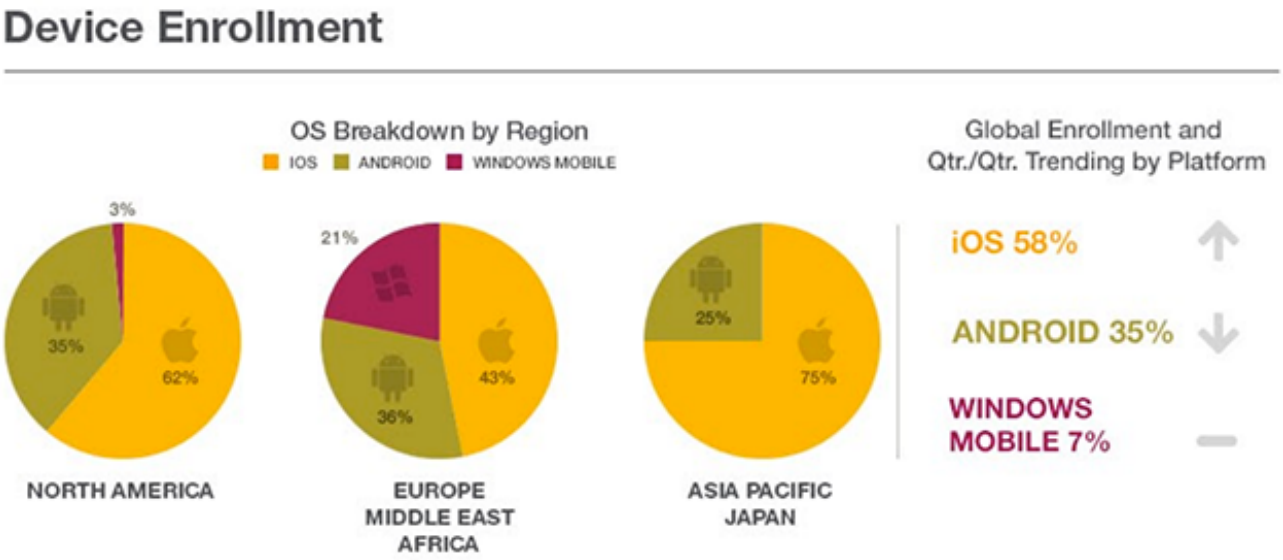

Figura 1: Uso de las plataformas móviles por las empresas de todo el mundo.

Es por todo lo anterior que aún pudiéndose seleccionar Android como base, se ha decidido mantener el sistema operativo original (iOS) pudiéndose así portar la aplicación a otras plataformas diferentes tras su finalización, inclusive Android.

Capítulo 2

2. Herramientas y tecnologías empleadas

En este capítulo se presentan las herramientas utilizadas a lo largo del proyecto, tanto software como hardware, detallando cada una de ellas.

Con este estudio se pretende adquirir una mayor comprensión del entorno tecnológico y valorar otros productos similares con el fin de evaluar si la construcción del sistema planteado es viable en el mercado actual.

2.1 Medios empleados

Una gran serie de elementos, tanto hardware y software, han sido utilizados para el desarrollo del proyecto. Entre ellos figuran los siguientes:

- Portátil Apple MacBook Pro 13” , con Mac OS X Lion 10.7.4 como sistema operativo.
- iPad Mini, con iOS 6.2 como sistema operativo.
- XCode 4.5, como herramienta oficial de desarrollo de la aplicación.
- Adobe Photoshop CS6, como herramienta para diseñar y crear los gráficos de la aplicación.
- Servidor de la Universidad de Loughborough, de donde es posible cargar simulaciones de ondas online.

2.2 Introducción a la plataforma iOS, Objective-C and Xcode

2.2.1 iOS y Objective-C

iOS es un sistema operativo para dispositivos móviles desarrollado y distribuido por la famosa e innovadora compañía americana Apple Inc. [8] Este sistema operativo fue originalmente lanzado en 2007, llamado como iPhone OS, introduciendo el iPod touch y aún más importante, el primer Smartphone multitáctil en el mundo, el iPhone, el cual marcaría y revolucionaría el mundo de las telecomunicaciones desde entonces. Además, Apple ha ampliado su gama de dispositivos móviles introduciendo el iPad y el iPad mini, cambiando también en 2010 el nombre de su sistema operativo utilizado por todos ellos como iOS .

La plataforma iOS es una derivación de Mac OS X [9], el sistema operativo utilizado en los ordenadores de Apple. Por lo tanto, iOS es un sistema operativo basado en UNIX [10], al igual que Mac OS X, siendo Objective-C su lenguaje de programación.

Objective-C es un elegante lenguaje de programación orientado a objetos que fue diseñado al comienzo de los años 80, dando lugar y convirtiéndose en uno de los lenguajes de programación mas usado actualmente y que además, fundamenta todas las aplicaciones iOS, disponibles en la App Store.

La App Store es el mercado de aplicaciones iOS online, que almacena más de 800.000 aplicaciones disponibles para descargar en cualquier momento para cualquier producto de Apple compatible con ellas. Además, es uno de los mercados de aplicaciones donde mas descargas se realizan, y el que mas ingresos y beneficios obtiene.

El kit de desarrollo de software (SDK) para Apple fue anunciado en 2008, dando, junto a Xcode, la posibilidad de desarrollar aplicaciones para iOS para desarrolladores de terceros, e incorporando numerosas herramientas que han sido usadas en el desarrollo del proyecto y que serán explicadas posteriormente.

Tras el lanzamiento del iPhone con iPhone OS, han habido números cambios de dispositivos y varias actualizaciones de su sistema operativo, tal y como se puede apreciar en la siguiente figura.

Esta, recorre desde el primer iPhone hasta actual iPhone 5, del primer iPad al iPad actual, así como el iPad mini, todos estos sin contar su amplia gama de iPod (iPod Touch, iPod Shuffle, iPod Nano…), cuya salida daría una gran fama a la compañía y una gran diferencia frente a la competencia. Todos ellos, poseen actualmente iOS 6.3 como versión final de su sistema operativo. Cabe destacar aquí una gran ventaja con sus competidores. Tal y como se puede ver en la siguiente figura, cuando hablamos de dispositivos iOS, la inmensa mayoría de ellos están actualizados siempre a su última versión, favoreciendo la no desfragmentación del sistema operativo. Esto implica una enorme ayuda a los desarrolladores, ya que estos tan sólo tienen que encargarse de adaptar sus aplicaciones en su última versión, obviando así las versiones anteriores, y reduciendo por tanto la labor para desarrollar y publicar una aplicación.

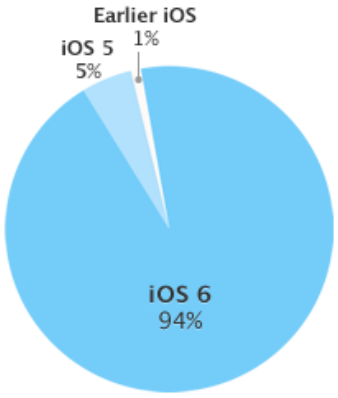

Figura 2: Adopción de las versiones de iOS en sus dispositivos.

Además, recientemente, Apple anunció su última versión de su sistema operativo móvil, iOS 7. Sin embargo, aún no ha sido oficialmente lanzado todavía y se mantiene en estado beta, es decir, iOS 7 está disponible para su descarga para desarrolladores en estado de pruebas y desarrollo hacia la versión final que será lanzada. Ésta actualización significa una de los mas grandes cambios en el sistema, pues además de incorporar numerosas nuevas prestaciones, presenta un cambio total de su interfaz y el modo de interaccionar con él.

Figura 3: Logo de iOS 7.

2.2.2 Xcode

Xcode es la herramienta oficial de desarrollo soportada por Apple para el desarrollo de software para Mac OS X y la plataforma iOS [11]. Xcode es un potente entorno de trabajo para crear aplicaciones para ordenadores Mac, iPhone e iPad. Además, su descarga es totalmente gratuita disponible para descargar directamente desde su tienda de aplicaciones, AppStore.

Figura 4: Logo de Xcode.

Además del entorno de desarrollo, Xcode incluye otra gran serie de herramientas de análisis y estudio para iOS y OS X. Éstas permiten realizar un intensivo estudio de una forma gráfica sobre información tan relevante de una aplicación como son la memoria usada por esta, el uso de CPU a tiempo real y energía consumida, entre otras características, haciendo de este modo las aplicaciones mas rápidas y eficientes . También es destacable iOS Simulator, un simulador de aplicaciones que permite simular aplicaciones desarrolladas para iPhone e iPad directamente desde el ordenador Mac, sin la necesidad de poseer ningún dispositivo conectado.

Entre las grandes características y funcionalidades de Xcode, además de su gran compilador LLVM Compiler, cabe destacar Interface Builder, una herramienta gráfica para diseñar las apliaciones que permite interconectar de una forma gráfica los objetos de la interfaz de la aplicación con el código de esta, insertando de esta forma el código necesario de los objetos de forma automática facilitando así la labor para el desarrollador.

2.2.3 Estructura de iOS y el modelo MVP

Con la introducción del kit de desarrollo de software de iOS (SDK), Apple puso a disposición de los desarrolladores las herramientas necesarias para programar aplicaciones para su sistema operativo. Éste, contiene las suficientes instrumentos e interfaces necesitadas para desarrollar, instalar, lanzar y probar aplicaciones nativas para iOS. Estas aplicaciones corren nativamente en iOS, y están construidas usando los frameworks proporcionados para su desarrollo, todo ello en Objective-C como lenguaje de programación.

Puesto que el SDK de iOS proporciona las fuentes necesarias para el desarrollo de aplicaciones nativas para iOS, cabe detallar alguna de sus funcionalidades.

En primer lugar, la arquitectura de iOS. Como cualquier sistema operativo actual, iOS está dividido en diferentes capas, cada una de las cuales desempeñan una función distinta y siendo en este caso cuatro, tal y como se puede ver en la siguiente figura.

Figura 5: Capas de la arquitectura iOS.

Esta estructura presenta una abstracción en capas, la cual facilita la tarea de programar una aplicación al desarrollador. Es decir, el desarrollador puede centrarse en la capa de mas alto nivel, donde es mucho mas fácil implementar código gracias a su abstracción orientada a objetos, utilizando tan solo las capas inferiores cuando sea estrictamente necesario y ahorrando de esta forma tener que escribir una gran cantidad de código y complejas funciones que ya vienen dados por los frameworks de alto nivel de la capa superior, Cocoa Touch.

La capa Cocoa Touch es una API que contiene los frameworks fundamentales para construir la mayoría de aplicaciones iOS. Esta capa define las estructuras básicas de una aplicación, por lo que se debe tener un gran conocimiento de sus capacidades antes de desarrollar alguna aplicación, con el objetivo de confirmar la posible realización de la aplicación deseada.

Entre estos frameworks figuran algunos tan importantes como los siguientes:

- UIKit Framework
- GameKit Framework
- iAd Framework
- Map Kit Framework
- Message Kit Framework

En este proyecto en concreto, principalmente se ha usado el primero de ellos, UIKit Framework, el cual es imprescindible en cualquier desarrollo de una aplicación. Este framework proporciona las infraestructuras necesarias tanto para la implementación de las interfaces gráficas de las aplicaciones así como para el uso de multi-gestos en las pantallas táctiles, entre otras de las muchas facilidades que ofrece.

Sin embargo, debido a la alta complejidad de la aplicación desarrollada, también se han utilizado otros frameworks pertenecientes a capas mas inferiores, como son….

Por otro lado, además de la estructura de iOS vista anteriormente, las aplicaciones para iOS siguen un diseño de implementación denominado Model-View-Controller (MVC). Según este patrón de diseño se definen tres componentes básicos a la hora de crear una aplicación:

- Modelo: define todos los datos y la información de la aplicación. Se encarga de gestionar y manejar los datos, así como también sus accesos. Por ello, se comporta como un almacén de datos de la aplicación, pero sin definir como se comporta la aplicación ni como son mostrados en ella.
- Vista: configura la interfaz de usuario y por tanto la apariencia física de la aplicación. Es el objeto con el que usuario interactúa, siempre a través de los diferentes elementos que el objeto vista implementa.
- Controlador: es el cerebro del diseño del patrón. Este se encarga de sincronizar los dos objetos anteriores. Responde a los eventos recibidos del usuario a través de la vista invocando peticiones al modelo para utilizar la información, así como también mandar a la vista los cambios que realiza el modelo.

El patrón MVC es fácilmente entendible a través de la siguiente figura:

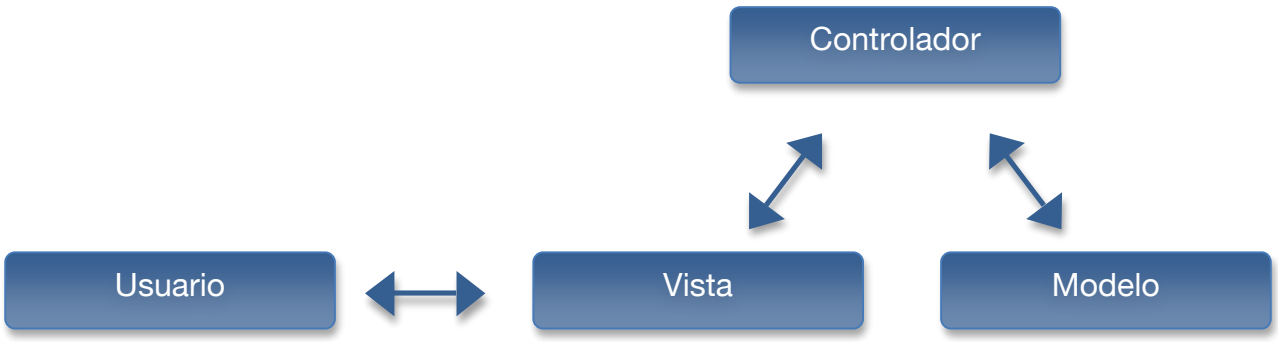

Figura 6: Diseño del patrón MVC.

De esta forma, el usuario interactúa con la aplicación a través de la interfaz de usuario, y por tanto de la vista. La vista le envía un mensaje al controlador, denominado action, informándole del acto realizado por el usuario. El controlador recibe el mensaje y contacta con el modelo para realizar la acción y actualizar la información pertinente. El controlador recoge la información requerida por la vista pero actualizada por el modelo y por último actualiza la vista a través de evento denominado outlet, con los cambios que se han hecho en el modelo.

Capítulo 3

3. Transmission Line Matrix (TLM)

En este capítulo se introducirá y detallará el principal fundamento físico existente detrás de la aplicación, el método TLM, en todas sus dimensiones. Con esto se pretende iniciar al lector a entender el funcionamiento básico de la aplicación, así como comprender los conceptos físicos que se aplican en ella.

3.1 Introducción

El método de la matriz de línea de transmisión o método TLM (Transmission Line Matrix) es una técnica numérica de discretización del espacio y el tiempo para la resolución de cálculos en campos electromagnéticos y acústicos con geometrías complejas. El método TLM fue desarrollado y publicado por Peter B. Johns and R.I. Beurle en 1971, como método de solución para problemas bidimensionales. Además, en 1975, el método sería extendido y ampliado, dando cabida también a cálculos tridimensionales con la introducción del método Nodo Condensado Simétrico denominado como Symmetrical Condensed Node (SCN) por Peter. B Johns y S. Akhtarzad.

Se trata de un método simple, intuitivo e incondicionadamente estable para la modelización de la propagación de ondas, debido a que el modelo está fuertemente relacionado con el proceso físico de la propagación.

Por ello, el método TLM es ampliamente usado en la computación de complejas estructuras electromagnéticas, tanto bidimensionales como tridimensionales, como también en problemas mecánicos y acústicos, siendo uno de los métodos mas potentes en el dominio del tiempo.

3.2 Principios de TLM

Para poder explicar de una manera profunda los métodos correspondientes a dos y tres dimensiones que se detallarán en los siguientes apartados, conviene tener, no obstante, una noción más concreta acerca de qué es el método TLM.

El método TLM está basado en el principio de Huygens, el cual es el principal fundamento en la propagación y dispersión de ondas, así como también en la analogía entre la propagación sobre el campo y las líneas de transmisión.

Una simulación normal del método TLM está caracterizada por una red de interconexión de líneas de transmisión, que representa el espacio de propagación de las ondas. Cada nodo de la red representa un nodo eléctrico, donde deben cumplirse las ecuaciones del circuito. De esta forma, el campo eléctrico y magnético son equivalentes al voltaje y corriente de la red, respectivamente.

Este modelo de propagación es exacto, ya que se trata de una red pasiva. En ella, la simulación empieza excitando la malla con impulsos de tensión en los específicos nodos a las condiciones de contorno de la señal inicial, como muestra la figura 1 (a). Después, en sucesivos intervalos de tiempo, la propagación de esos impulsos sobre la malla son parcialmente dispersos y transmitidos a través de las líneas de transmisión a los nodos adyacentes, repitiéndose el proceso de nuevo en estos nodos adyacentes convirtiéndose así en los nuevos impulsos, además de ser reflejados en las fronteras de cada línea de transmisión en cada intervalo de tiempo. De esta forma, las ondas de todos los nodos que se transmiten a través de las líneas de transmisión forman la onda en general, cumpliendo también la ley de conservación de energía.

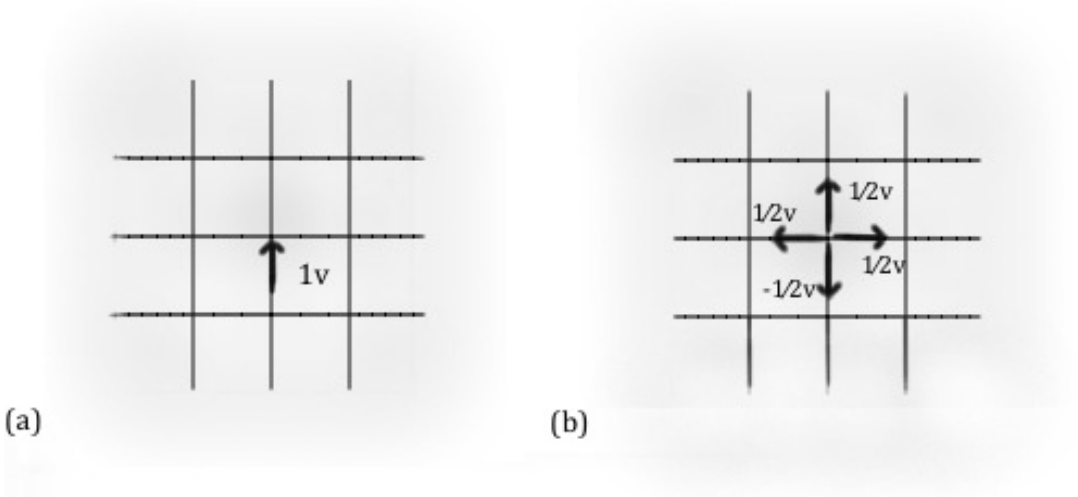

Figura 7: (a) Incidencia de la excitación. (b) Ondas reflectadas resultantes de la onda incidente.

Un ejemplo de esto es el esquema que se muestra en la figura anterior. Ésta muestra los primeros intervalos de una simple simulación sobre una malla homogénea. En ella, tal y como se puede ver

en la figura a), se inyecta un impulso de un voltio sobre un nodo de la malla a través de una línea de transmisión al intervalo k = 0. Al siguiente intervalo, k = 1, el pulso incidente es parcialmente propagado por las líneas del nodo hacia los nodos adyacentes siguiendo la teoría de la línea de transmisión. De acuerdo a esta, cada línea tiene una impedancia característica Z_0 . Por lo tanto, el pulso incidente ve al llegar al nodo una impedancia $Z_0/3$ resultado del paralelo de las tres líneas que convergen al nodo implicando además un coeficiente de reflexión y transmisión. Siguiendo el proceso, este pulso llegan a los cuatro nodos vecinos, propagándose de nuevo desde los cuatro nodos hacia sus correspondientes nodos adyacentes, y repitiendo así el procedimiento sucesivamente.

Este ejemplo teórico es bastante sencillo, ya que en la práctica las mallas se complican y existen mas nodos donde se inician la propagación. Además, en la realidad, existen fronteras donde las ondas son reflejadas. Estas fronteras se modelizan colocando adecuadas impedancias de cargar en los nodos. Así un cortocircuito indica una condición de anulación del campo eléctrico, creado por un conductor (pared eléctrica) que lleva a una reflexión total. De esta forma, un circuito abierto indica una condición de anulación del campo magnético (pared magnética), que también lleva a reflexión total. Para modelizar recintos infinitos, se definen fronteras ideales donde la propagación continua sin reflexión, lo que implica colocar impedancias de cargar adaptadas (pared absorbente). Además, otras variantes de cargas permiten modelizar otras condiciones de contorno.

El modelo TLM puede ser aplicado a un espacio con una, dos o tres dimensiones. En esta sección, y con lo que respecta al proyecto, se explicarán el modelado para dos y tres dimensiones, detallando el proceso de dispersión y conexión correspondientes.

3.3 Algoritmo TLM para dos dimensiones

El algoritmo del método TLM para dos dimensiones sigue el procedimiento descrito en el apartado anterior. De acuerdo a este, se pueden definir dos tipos de procesos, el proceso de dispersión de los pulsos desde el nodo, y el proceso de conexión de los pulsos hacia los nodos adyacentes.

3.3.1 Proceso de dispersión

El proceso de dispersión es la parte del procedimiento donde se calculan los voltajes reflejados por cada una de las ramas pertenecientes a un nodo, en el intervalo t+1, siendo obtenidos los valores desde los voltajes incidentes en la iteración anterior en el intervalo t. Por lo tanto, el voltaje de cada rama viene dado por la siguiente expresión:

$$ {}_{t+1}V_{rama}{}^{r} = ½ \left({}_{t}V_N{}^{i} + {}_{t}V_E{}^{i} + {}_{t}V_S{}^{i} + {}_{t}V_W{}^{i} \right) - {}_{t}V_{rama}{}^{i} \qquad (1) $$

Donde rama es la rama en la cual se calcula el proceso, y donde ${}_{t}V_N{}^{i}$,${}_{t}V_E{}^{i}$,${}_{t}V_S{}^{i}$,${}_{t}V_W{}^{i}$ son los diferentes voltajes pertenecientes a las cuatro ramas de un nodo, siendo estos north, east, south y west,

correspondientemente. De esta forma, es posible calcular el voltaje total de un nodo en particular como:

$$V_t = ½ ({}_tV_N^i + {}_tV_E^i + {}_tV_S^i + {}_tV_W^i) \quad (2)$$

Esto se puede entender de una mejor forma en la siguiente figura 2, donde se pueden ver las correspondientes ramas para un nodo en dos dimensiones.

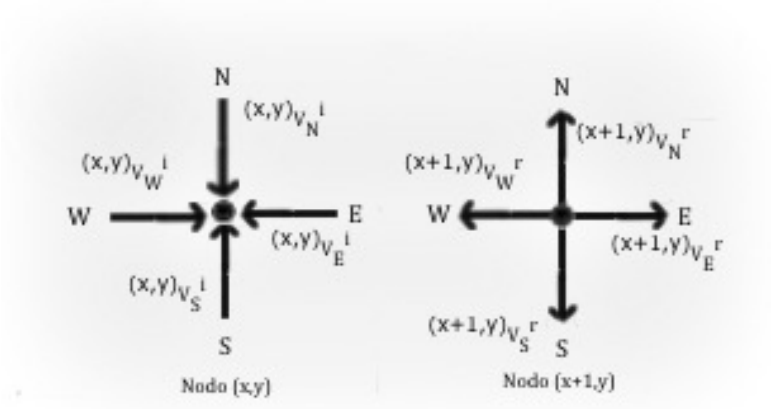

Figura 8: Dos nodos bidimensionales adyacentes.

En ella, se ven los dos diferentes tipos de procesos correspondientes al método TLM. En el esquema de la derecha se puede ver el proceso de dispersión, en el cual salen los pulsos por cada uno de las ramas del nodo en cuestión hacia los nodos vecinos. Estos pulsos se originan en el impulso incidente en el nodo en el intervalo anterior, proceso denominado como proceso de conexión, que se puede ver en el esquema de la izquierda, y que será explicado en el apartado siguiente.

3.3.2 Proceso de conexión

Después de que el proceso de dispersión sea aplicado a todos los nodos, el proceso de conexión convierte todos los voltajes reflejados y calculados en el proceso anterior en los nuevos voltajes incidentes para cada correspondiente rama del nodo adyacente, como se puede ver en la figuras 2 y 3.

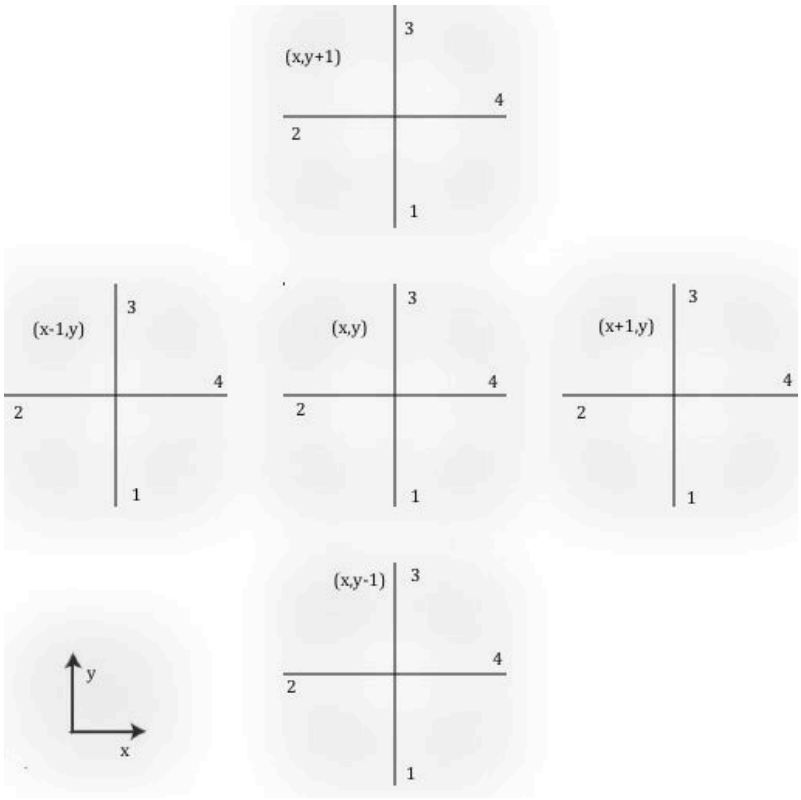

Figura 9: Proceso de conexión bidimensional.

Por lo tanto, el proceso de conexión viene caracterizado por las siguientes expresiones:

$$
\begin{aligned}
{}_{t+1}V_1^i(x,y) &= {}_kV_3^r(x,y-1)\\
{}_{t+1}V_2^i(x,y) &= {}_kV_4^r(x-1,y)\\
{}_{t+1}V_3^i(x,y) &= {}_kV_1^r(x,y+1)\\
{}_{t+1}V_4^i(x,y) &= {}_kV_2^r(x+1,y)
\end{aligned}
\qquad (3)
$$

Además, se debe hacer una apunte en este punto. Las anteriores expresiones son seguidas siempre y cuando se le aplique el proceso a un nodo cuyo nodo adyacente no sea el borde de la malla o un obstáculo dentro de ella. Es decir, para casos donde el nodo vecino sea parte de las fronteras de la malla o parte de algún un obstáculo dentro de la malla, el proceso de conexión se comporta de manera diferente como ya se ha mencionado con anterioridad, y el cual será explicado y detallado en la sección de implementación.

3.4 Algoritmo TLM para tres dimensiones

Como ha sido dicho anteriormente, además de el algoritmo del método TLM para dos dimensiones, también fue introducido un método distinto para tres dimensiones. Al tratarse de ondas de campos, en general las propiedades de propagación dependen de la polarización, de manera que se colocan dos líneas independientes en cada rama de la malla, una vinculada con la propagación del campo polarizado verticalmente y otra vinculada con la propagación del campo polarizado horizontalmente. Para tener en cuenta todas las posibilidades en 3D, habitualmente se utiliza el método llamado Symmetrical Condensed Node (SCN) o nodo condensado simétrico [12]. De acuerdo a este, cada nodo está definido por doce magnitudes, dos por cada sentido y dirección del espacio, para acomodar la propagación de ambas polarizaciones.

Sin embargo, debido a su gran complejidad, y para el desarrollo del proyecto, el algoritmo para dos dimensiones ha sido extendido, siguiendo la analogía y añadiendo la coordinada z para crear el algoritmo del método TLM para tres dimensiones.

3.4.1 Proceso de dispersión

El proceso de dispersión para tres dimensiones sigue la analogía del proceso para dos dimensiones. Por lo tanto, las expresiones seguidas para el anterior algoritmo, pueden ser extendidas de la siguiente manera:

$$ {}_{t+1}V_{branch}^{r} = 1/3\ (\ {}_tV_N^i + {}_tV_E^i + {}_tV_S^i + {}_tV_W^i + {}_tV_T^i + {}_tV_B^i) - {}_tV_{branch}^i \qquad (4) $$

Siendo ${}_tV_T^i$ y ${}_tV_B^i$ las ramas correspondientes al plano z.

En adición, el voltaje total para un nodo en particular es también extendida como:

$$ V_t = 1/3\ (\ {}_tV_N^i + {}_tV_E^i + {}_tV_S^i + {}_tV_W^i + {}_tV_T^i + {}_tV_B^i) \qquad (5) $$

3.4.2 Proceso de conexión

El proceso de conexión para tres dimensiones también sigue la analogía del proceso para dos dimensiones, quedando las expresiones para este proceso como:

$$
\begin{aligned}
{}_{t+1}V_1^{i}(x,y,z) &= {}_{k}V_3^{r}(x,y-1,z)\\
{}_{t+1}V_2^{i}(x,y,z) &= {}_{k}V_4^{r}(x-1,y,z)\\
{}_{t+1}V_3^{i}(x,y,z) &= {}_{k}V_1^{r}(x,y+1,z)\\
{}_{t+1}V_4^{i}(x,y,z) &= {}_{k}V_2^{r}(x+1,y,z)\\
{}_{t+1}V_5^{i}(x,y,z) &= {}_{k}V_6^{r}(x,y,z-1)\\
{}_{t+1}V_6^{i}(x,y,z) &= {}_{k}V_5^{r}(x,y,z+1)
\end{aligned}
\qquad (6)
$$

Capítulo 4

4. Descripción, Implementación y pruebas

En este tercer capítulo se tratará de describir la aplicación y su funcionamiento en general, además de detallar todo aquello relacionado con la implementación del proyecto, y por tanto de la aplicación. Adicionalmente se especificarán las principales partes de esta, explicando la estructura de la aplicación así como las metodologías de las partes mas importantes del código, facilitando el buen entendimiento de este. Por otro lado, se especifican las pruebas y test realizadas a lo largo del proyecto, acompañados de todos los resultados obtenidos.

4.1 Descripción general de la aplicación

Una vez presentadas las tecnologías que han sido y son usadas por la aplicación, y estudiados los objetivos y el principal fundamento teórico en el que se basa la propagación de las ondas que hace posible la aplicación, se proseguirá a realizar una descripción general de la aplicación y su funcionamiento.

Como se ha dicho en los primeros capítulos, el principal objetivo del proyecto es el de crear una aplicación para visualizar como se propagan diferentes ondas tridimensionales en distintas condiciones, usando el método TLM para tres dimensiones visto en el capítulo anterior. De esta forma, se visualizará un tipo de onda sobre la pantalla, pudiendo cambiar entre las desarrolladas por defecto y las cuales se encuentran dentro de la aplicación: Gaussiana, Sinusoidal, Ripple Tank o Tanque de ondas, Waveguide o Guía de ondas y Double Slit o Doble Agujero, y las que se puede cargar a través de un servidor. Además de poder elegir entre los diferentes modos de excitación, entre los distintos tipos de ondas, también es posible configurar otras características de las ondas directamente desde la aplicación, entre las que destacan poder cambiar la longitud de onda de la onda en ejecución, cambiar el modo de simulación o modificar el tipo de borde de la malla y sus condiciones. Sin embargo, todas las características de la aplicación serán detalladas con mas profundidad en secciones posteriores.

4.2 Implementación

Una vez presentadas las tecnologías que han sido y son usadas por la aplicación, y descrita la aplicación de una manera general, a continuación se especifican todos los detallas de la implementación de la aplicación, así como también todas las partes importantes de esta para lograr tener claro el funcionamiento del programa.

Por otro lado, basándose en lo anterior y debido a la gran cantidad de archivos y al amplio contenido que componen la aplicación, sería imposible detallar todas las partes del código de la aplicación. Por ello, solo se mostrarán las partes mas importantes, las cuales tienen una mayor incidencia en el resultado y objetivo de la aplicación.

4.2.1 Estructura general

En primer lugar, y para empezar a comprender el funcionamiento de la aplicación, se muestra la estructura del código del programa en la siguiente figura, con la que es posible hacerse una idea general de la aplicación en una primera vista.

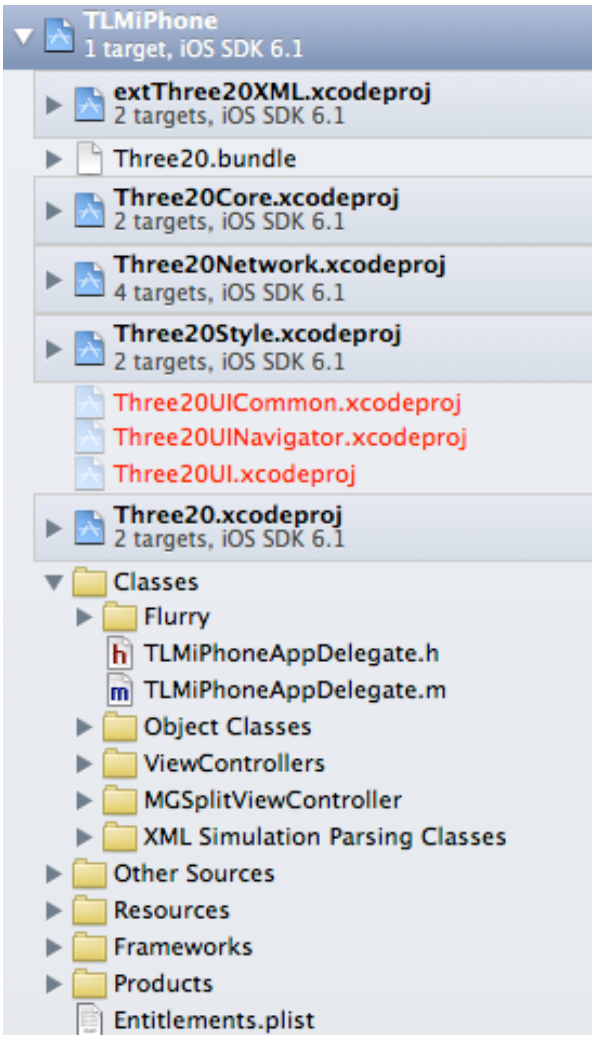

Figura 10: Estructura general de la aplicación.

A continuación se va a describir cada una de estas partes, antes de adentrar y detallar algunas de las mas importantes.

Como vemos, el proyecto en Xcode ha sido llamado *TLMiPhone*. Es decir, *TLMiPhone* es el nombre del programa. A continuación siguen una serie de elementos, exactamente nueve, que forman parte de una librería utilizada y que es llamada three20 [13]. Esta es una potente y muy útil librería que expande la funcionalidad para desarrollar para esta plataforma, agregando elementos de la API privada de Apple , y que de otra forma no es posible usar.

Seguidamente, empiezan todas las clases desarrolladas para el programa. Siguiendo la jerarquía de este, nos encontramos varias carpetas, que contienen subcarpetas donde están definidas todas las clases y elementos. En un primer nivel nos encontramos las carpetas *Classes*, *Other Sources*, *Resources*, *Frameworks y Products*, además del fichero *entitlements.plist.* Por lo tanto, a continuación se explican sus funciones y contenidos:

- *Classes*: contiene la información que define el programa, en cinco carpetas a su vez, además de los archivos *TLMiPhoneAppDelegate.h* y *TLMiPhoneAppDelegate.m*. *Object Classes*, *ViewControllers*, *MGSplitViewController* y *XML Simuation Parsing Classes* componen todas las clases que definen el comportamiento de la aplicación. Algunas de estas serán detalladas posteriormente, puesto que es una de las partes mas importantes de la aplicación. En adición, también se encuentra la *carpeta Flurry*. Esta es la carpeta correspondiente al uso de Flurry Analytics, una herramienta gratuita que analiza y ofrece estadísticas de uso y trackeo de audiencia entre otras características. Sin embargo, esta es una funcionalidad de la aplicación base que se ha inhabilitado en la presente aplicación, pero que se ha mantenido para poder hacer uso de ella cuando la aplicación final se lance en la App Store.
- *Other sources*: contiene los archivos *TLMiPhone_Prefix.pch* y *main.m*. Estos dos archivos se encargan de iniciar y ejecutar la aplicación y en la mayoría de ocasiones no hace falta modificar ninguno de ellos. El primero se encarga de compilar especificadas cabeceras en un primer momento, estando disponible para el resto de los archivos que implementen esas cabeceras durante la ejecución, sin necesidad de volver a compilarlas para cada archivo. La principal tarea de *main.m* es la de llamar al método que inicia la aplicación.
- *Resources*: son todos los elementos gráficos que componen las interfaces gráficas y apariencia de la aplicación. Contiene tres carpetas: *iPhone, iPad* e *Images*. En esta última carpeta se guardan las diferentes imágenes externas que son usadas tanto por las interfaces de iPhone como por las interfaces al iPad. Precisamente, son en las carpetas *iPhone* e *iPad* donde se almacenan sus correspondientes interfaces gráficas.

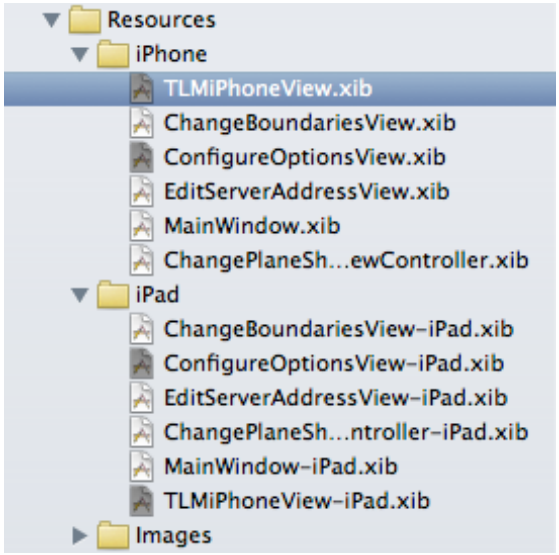

Figura 11: contenido de la carpeta *Resources*.

Como se ve en la figura anterior, las interfaces gráficas de la aplicación son archivos .xib. Con la actualización de Xcode a su versión 4.0, este incluye un sistema, llamado Storyboard, para definir las interfaces gráficas y sus elementos de una manera mucho mas fácil y sencillo, todo de forma gráfica adjuntando el código necesario para su uso automáticamente y ahorrando así esa tarea para el desarrollador. Sin embargo, se ha debido mantener este tipo de desarrollo de interfaces, dejando el inserción de las interfaces mediante storyboards para las futuras mejoras de la aplicación.

- *Frameworks*: contiene todos los frameworks con los que trabaja de la aplicación que forman parte de la API Cocoa Touch, pero que no están disponibles por defecto. Están algunos fundamentales y necesarios para el funcionamiento en iOS como el antes mencionado *UIKit.framework* o *CoreGraphics.framework*, así como otros que dan soporte a la utilización de elementos para la interfaz gráfica o las configuraciones del sistema.

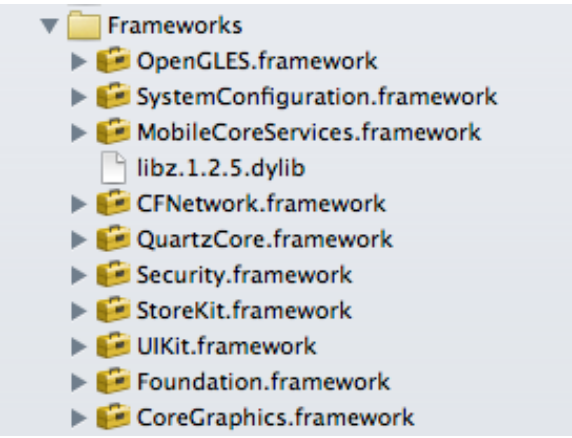

Figura 12: contenido de la carpeta *Frameworks*.

- *Products*: esta última carpeta contiene el paquete funcional de la aplicación.
- *Entitlements.plist*: es un archivo de configuración que proporciona acceso a la aplicación a determinadas funciones de iOS, como por ejemplo las notificaciones push.

4.2.2 Clases

Una vez vista la estructura general de la aplicación, y las funciones que desempeña cada una de sus partes, de una manera simple y generalizada, en esta sección se van a detallar aquellas clases mas importantes del programa, que haciendo memoria de la sección anterior, están contenidas en la carpeta *Classes*.

La carpeta *Classes*, tal y como se comentado en la sección anterior, contiene todas las clases desarrolladas para la aplicación. Sin embargo, antes de pasar a profundizar cada una de las clases, cabe definir el concepto de clase en iOS.

Una clase en iOS viene definida por dos archivos, un archivo de cabecera o interface, terminado con el sufijo .h de “header”, y un archivo de implementación, terminado con el sufijo .m. Es en el archivo interface donde se define la clase, y por norma general sus métodos y propiedades. Si estos son definidos y declarados en este archivo, significa que serán públicos, y por tanto pueden ser usados por el resto de clases. En el archivo de implementación es donde se describen los métodos y propiedades declarados en el interface. Además, se pueden definir métodos y propiedades dentro de la implementación, haciendo a los métodos y propiedades privados para el resto de clases.

Observando la siguiente figura, se puede ver todas las clases que se han desarrollado para la aplicación, y que se encuentran dentro de la carpeta *Classes.*

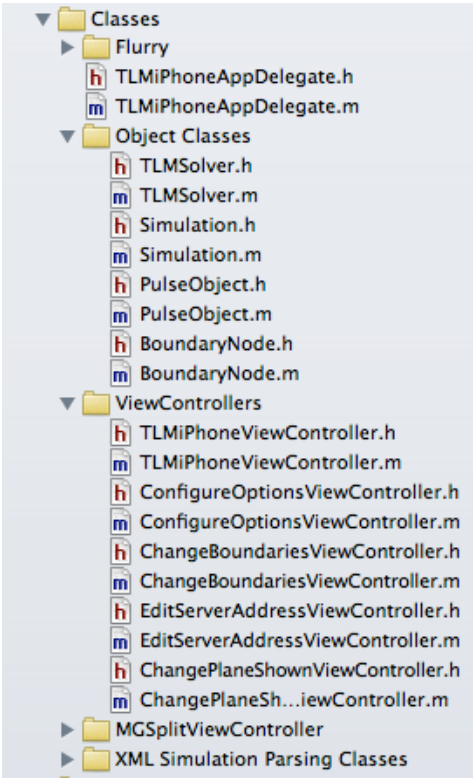

Figura 13: contenido de la carpeta *Classes.*

4.2.2.1 TLMiPhoneAppDelegate

En primer lugar, nos encontramos con la clase TLMiPhoneAppDelegate, con sus dos archivos *TLMiPhoneAppDelegate.h* y *TLMiPhoneAppDelegate.m*. Toda aplicación tiene una clase AppDelegate, que es la superclase y la encargada de mostrar la vista principal del programa y ejecutar las funciones que se le indique antes de comenzar totalmente la aplicación. Es más, el papel del archivo *main.m* antes visto es llamar a esta clase.

Sin embargo, debido a su larga extensión, es imposible mostrarla directamente, por lo que se van a señalar los puntos mas importantes. Antes de empezar, cabe decir que es donde se declaran todas las constantes y todas las propiedades que afectarán a la aplicación, al tratarse de la superclase. En un primer lugar se crean todos los identificadores que servirán para seleccionar los modos diferentes a través de un nombre en lugar de utilizar números, ya que facilita tanto su desarrollo como su legibilidad. La mayoría de ellos son creados a través de estructuras, al igual que en C, puesto que Objective-C es un derivado de C. Además, es aquí donde se definen algunas constantes del programa mediante el comando *#define*, ya que una vez creadas, son accesibles por todo el resto de clases en cualquier momento. Algunos ejemplos importantes de todo lo anterior son los siguientes:

Tipo	Identificadores	Descripción
Current_Device	IPHONE_ORIGINAL,IPHONE_RETINA IPAD,IPAD_RETINA,IPAD_MINI	Sirven de identificadores para inicializar la aplicación de una forma diferente dependiendo del dispositivo usado.
Mode	GAUSS,SIN,RIPPLE,WAVEG, DBLSLIT,CUSTOM	Sirven de identificadores para los distintos tipos de ondas.
Sim	EZ,HX,HY,HZ	Sirven de identificadores para los distintos modos de visualización.
BType	NOEGDE,TOP,BOTTOM,LEFT RIGHT,UPPER,LOWER	Sirven de identificadores para los distintos tipos de bordes de la malla de la aplicación.

Tabla 1: identificadores pertenecientes a la clase *TLMiPhoneAppDelegate*.

Constantes	Descripción
COMPONENTS_PER_PIXEL	Número de elementos por píxel
BITS_PER_COMPONENT	Número de elementos por bits
PEC_COND	Condición de nodo tipo Perfecto Conductor Eléctrico
PMC_COND	Condición de nodo tipo Perfecto Conductor Magnético
MTC_COND	Condición de nodo tipo Matched Termination Conductor
numberOfBranches	Número de ramas que compone un nodo

Tabla 2: constantes pertenecientes a la clase *TLMiPhoneAppDelegate*.

Además, dentro de la clase, se definen una gran serie de variables y propiedades. Debido a la gran cantidad, se van a destacar los siguientes:

Variables	Descripción
xScreenConstraint e yScreenConstraint	Definen el espacio entre el borde de la pantalla y el borde de la malla (tipo property).
rowLen y columnLen	Definen la altura y anchura del tamaño de la pantalla dependiendo el dispositivo usado (tipo property).
meshDepth	Tamaño de la profundidad de la malla (tipo property).
zPlaneShowed	Número de plano o capa mostrado en la aplicación (tipo property).
nodesX y nodesY	Definen el número de nodos de la malla (tipo property).
currentExcitationMode y previousExcitationMode	Sirven para controlar la transición entre los tipos de excitaciones (tipo property).

Tabla 3: variables pertenecientes a la clase *TLMiPhoneAppDelegate*.

Los métodos definidos por esta clase se explican a continuación:

- *(void) savePreviousGlobalIterationCounterValue;* -> guarda el tipo de excitación de onda visualizada en la iteración anterior para poder guardar el estado de la malla y poder volver a representarla posteriormente, en el momento que se vuelva a ella.
- *(void) fetchCurrentGlobalIterationCounterValue;* -> actualiza el contador con el tipo de excitación de onda actual.
- *(void) resetScreenAndMeshDefaults;* -> reinicia todos los valores de las variables concernientes al tamaño de la pantalla así como el de la malla, ajustándose y distinguiendo entre el dispositivo usado.
- *(void) clearTLMArray;* -> resetea todos los valores de los arrays que contienen la información del tipo de excitación actual, borrando de esta forma la onda.
- *(void) clearBuffers;* -> llama al método clearBuffers de la clase objeto TLMSolver, el cual borra todos los datos de todos los arrays.
- *(void) checkMeshBuffersForMalloc;* -> reserva espacio de memoria para los buffers dependiendo del tipo de la excitación de onda actual.

Además, cabe decir que se han modificado algunos métodos que configuran el estado inicial de la aplicación, ya que como se ha visto anteriormente, la clase AppDelegate es la encargada de ello. De este modo, se destaca la implementación del método - *(BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions:.*

Este método es el encargado de ejecutar todas aquellas acciones deseadas antes de lanzar totalmente la aplicación. Por ello, se han hecho algunas modificaciones como la visualización de una imagen de entrada a la aplicación, la detección del dispositivo que usa la aplicación o el inicio de los valores por defecto para todas las variables de las que depende de la aplicación, de acuerdo al dispositivo usado.

En adición, hay que realizar un apunte aquí para distinguir los tipos de métodos que existen en iOS. Estos se diferencian en el primer carácter, pudiendo ser "-" o "+", correspondiendo respectivamente a los métodos estancia, los cuales solo se pueden usar sobre un objeto creado a partir de la clase, y los métodos de clase o estáticos, que pueden ser usados sin crear esa instancia del objeto. Por consiguiente, todos los métodos anteriores son métodos de estancia, y a lo largo de la sección se verá que el resto también lo son.

4.2.2.2 Object Classes

A continuación se encuentran las carpetas *Object Classes* y *ViewControllers*. Estas están destinadas a almacenar las clases que trabajan y manejan los datos de la aplicación, y asocian los datos con la interfaz gráfica, respectivamente.

Uno de los principales objetivos del proyecto era poder modificar el algoritmo de propagación de ondas para dos dimensiones en uno para tres dimensiones. Es decir, el modelo 2D de la aplicación base se ha modificado y se ha creado un modelo 3D para poder trabajar con ondas tridimensionales. La clase fundamental y responsable de realizar esta tarea es *TLMSolver,* donde se aplica la teoría acerca del método TLM mostrada en las anteriores secciones. Por ello, se va a profundizar especificando cada una de sus partes y su funcionalidad: constantes, variables y métodos utilizados mas importantes.

Identificadores	Descripción
VXN,VYN,VXP,VYP,VZN,VZP	Son los identificadores de las ramas de un nodo, que serán usados como punteros en los arrays que contienen la información de la malla.
R,G,B	Identificadores de los elementos de un pixel, que sirven para facilitar la tarea de componer los colores a través de los arrays correspondientes.

Tabla 4: identificadores de la clase *TLMSolver*.

Variables	Descripción
rowLen y columnLen	Tamaño de la anchura y altura de la pantalla del dispositivo usado (tipo NSInteger).
meshHeight, meshWidth, meshDepth	Tamaño de la anchura, altura y profundidad de la malla (tipo NSInteger).
pixels	Array que almacena todos los valores de los píxels que se expondrán por pantalla.(tipo puntero a signed char) .
TLMArray, kPlus1Array, swapper, kPlus1ArrayCopy	Punteros a float que sirven para trabajar con todos los datos de la malla (tipo puntero a float).
pixelLoopCount	Sirve de puntero para llevar la cuenta de los pixels en el algoritmo TLM (tipo int).

specmapArray	Array que contiene una serie de colores pre-calculados para hacer mas eficiente el algoritmo TLM (tipo array de enteros).

Tabla 5: variables de la clase *TLMSolver*.

Los métodos de la clase se muestra en la siguiente figura:

```
- (id)init;
- (void)seedNewImpulseWithXCoordinate:(int)touchXPos YCoordinate:(int)touchYPos withPhaseOffset:(int)phase;
- (void)computeBoundaries;
- (void)computePulses;
- (void)implementBoundaryNodeWithCondition:(float)cond atX:(NSInteger)x andY:(NSInteger)y andZ:(NSInteger)z ;
- (void)setupEdgeBoundaries;
- (void)removeEdgeBoundaries;
- (void)createStraightLineBoundaryWithCondition:(BOOL)cond andStartX:(NSInteger)stX andStartY:(NSInteger)stY andEndX:
    (NSInteger)eX andEndY:(NSInteger)eY isEdge:(NSInteger)meshEdge;
-(void)createAreaBoundaryWithCondition:(BOOL)cond andStartX:(NSInteger)stX andStartY:(NSInteger)stY andEndX:(NSInteger)
    eX andEndY:(NSInteger)eY andZ:(int)z isEdge:(NSInteger) meshEdge;
- (void)createRippleTankWave;
- (void)createWaveguide;
- (void)createDoubleSlitWave;
- (void)injectEnergyValue:(float)energy atX:(int)x andY:(int)y;
- (void)updateLocalScatterAndConnectVariables;
- (void)scatterAndConnect;
- (void)scatterAndConnectNEON;
- (void)configureCurrentExcitationMode;
- (signed char *)getPixelArray;
- (void)clearPixelArray;
- (void)initSpecMapArrayWithCustomContrast:(BOOL)customContrast;
- (void)meshCopyBack;
- (void)meshSave;
- (void)clearBuffers;
- (void)scatterAndConnectAlgorithmFromX:(int)xInitial Y:(int)yInitial toX:(int)xFinal toY:(int)yFinal
    withSimulationMode:(int)mode;
- (void)printTLMValues;
- (float)normalDistWithX:(float)x mean:(float)mean stdDev:(float)stdDev;
@end
```

Figura 14: métodos implementados por la clase *TLMSolver*.

La tarea fundamental de la clase *TLMSolver* es el algoritmo TLM. Este se encarga de calcular los voltajes de los nodos de la malla a través del método TLM y almacenarlos en un array (*TLMArray* y *kPlus1Array*), con lo que se hace posible la propagación de las ondas. El método dentro del código responsable de esto es *scatterAndConnect* el cual, a su vez, llama a *scatterAndConnectAlgorithmFromX(int)xInital Y:(int)yInitial toX:(int)xFinal toy:(int)yFinal withSimulationMode:(int)mode*, los cuales han sido creados y modificados para implementar adecuadamente la propagación tridimensional, siempre aplicando el método de la línea de transmisión. Además, conforme se calculan los voltajes de los nodos, estos son transformados a su correspondiente intensidad de pixel a través del array pre-configurado de intensidades RGB, que son las que forman el color que se muestra por pantalla. Este array, *specmapArray*, guarda los correspondientes valores de un pixel RGB, de acuerdo al voltaje de un nodo. Para no calcular continuamente los valores del pixel dentro de la iteración del algoritmo TLM, el método -*(void)initSpecMapArrayWithCustomContrast:(BOOL)customContrast* pre-calcula todos los valores posibles almacenándolos en el array antes mencionado, ahorrando de esta forma un gran número de operaciones y haciendo la aplicación mas eficiente.

De esta forma, primero se calculan los voltajes de cada nodo, transformándose en su correspondiente valor para el pixel RGB, y guardándose todos estos valores calculados en el array pixels, que es el array que contiene todos los valores que se imprimirán por pantalla.

Además, cabe destaca la creación de arrays unidimensionales, para almacenar la información correspondiente a nodos tridimensionales, para optimizar la velocidad de acceso a memoria. Es

decir, los datos utilizados para la propagación de las ondas en tres dimensiones es almacenada en arrays de una dimensión, permitiendo un acceso más rápido a la información de los arrays, mejorando y optimizando el rendimiento y la velocidad de la aplicación.

Por otro lado, se ha de explicar algunos otros métodos relevantes, que influyen también en el cálculo de los voltajes de los nodos. Los métodos anteriores que calculan el voltaje de todos los nodos, en realidad no calcula el de todos los nodos, debido a los bordes de la malla. Es decir, estos métodos se encargan de calcular el voltaje de todos los nodos dentro de la malla, pero para los contiguos al borde de la malla se le aplica otro, y parecido, algoritmo. De esto se encarga el método - *(void) implementBoundaryNodeWithCondition:(float)cond atX:(NSInteger)x andY:(NSInteger)y andZ:(NSInteger)z ,* el cual calcula ,de forma similar al algoritmo antes explicado, el voltaje de los nodos contiguos a los bordes de la malla, aplicándoles el tipo de condición correspondiente. Esto hace posible la diferencia en reflexión o absorción por parte del borde, de acuerdo al tipo de borde del que se trate.

Llegado a este punto, y debido a la diferencia de los bordes existentes en una malla bidimensional y en una tridimensional, a continuación se realizará un análisis de la implementación de los bordes de la malla, con el fin de obtener una aclaración en el desarrollo de los bordes en el modelo 3D.

La función de los límites es llevada a cabo también mediante el método TLM, pero de una manera especial. Como se ha detallado anteriormente, el principal principio del método TLM es obtener el valor de cada rama de las ramas de los nodos vecinos. Por lo tanto, se ha implementado un tipo de nodo especial, vacío, de tal forma que la onda reflectada es la misma que la incidente, que es la forma en las que las ondas se propagan al chocar con un obstáculo. Esto significa que cuando una onda incide sobre un borde, este desconecta la onda usando el nodo vecino (vacío) mientras que también se le aplica la condición de límite, devolviendo los valores de los nodos adyacentes, y por tanto reflectando toda onda incidente.

Para una mejor compresión, la siguiente figura muestra la reflexión existente cuando una onda incide sobre un nodo perteneciente al límite de la malla.

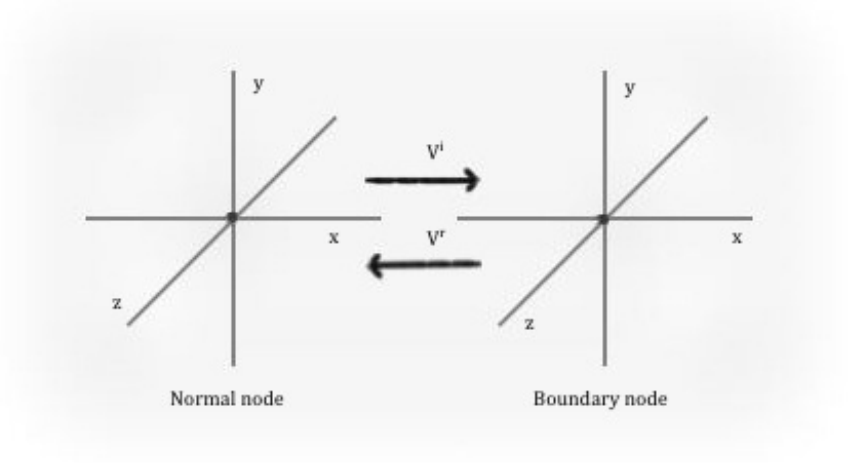

Figura 15: reflexiones de las ondas en los límites y obstáculos de la malla, donde $V^i = V^r$, siendo V^i el voltaje incidente y V^r, el voltaje reflectado.

Por todo ello, los nodos pertenecientes a los bordes, o nodos frontera, han tenido que ser modificados y extendidos para poder adaptarse a la propagación tridimensional.

Por otro lado, y teniendo en cuenta todo lo anterior, cabe mencionar otra modificación que ha de hacerse para el correcto funcionamiento de la aplicación. Como se puede ver en la figura 16, existen

diferencias en cuanto a los límites de una malla entre propagaciones de ondas bidimensionales y propagaciones de ondas tridimensionales. Respecto al primer tipo, y puesto que sólo hay dos dimensiones, solamente han de tratarse cuatro bordes o fronteras de la malla. Sin embargo, tratando con ondas tridimensionales, hay que tener en cuenta otros dos límites de propagación, que corresponden con la introducción del plano z, dedicado a la interconexión entre capas. Por lo tanto, debe haber una capa superior y una capa inferior, que se comporten como bordes, reflectando la propagación de ondas entre capas.

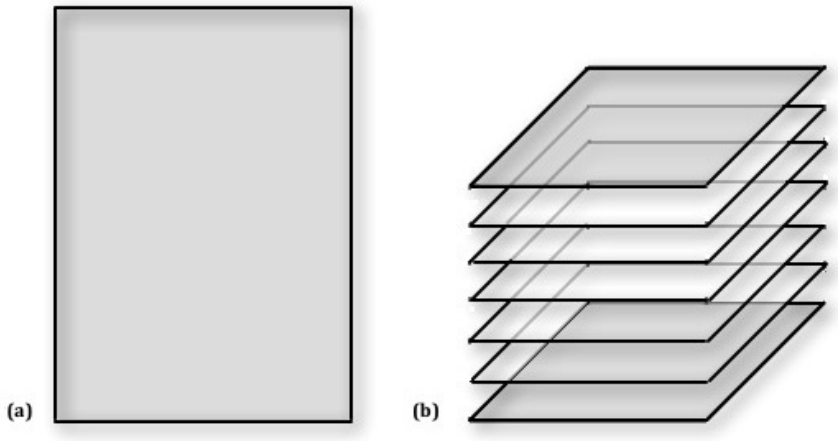

Figura 16: (a) Vista de una malla bidimensional. (b) Vista de una malla tridimensional.

Una vez vista la clase *TLMSolver*, la siguiente es *Simulation*. Sin embargo, su única función es la liberar memoria reservada para la simulación en ejecución, siendo esto lo único destacable de la clase.

Seguidamente se encuentra la clase *PulseObject*, y cuyo fichero de cabecera se muestra a continuación:

```
#import <Foundation/Foundation.h>
@class TLMiPhoneAppDelegate;

@interface PulseObject : NSObject {
    TLMiPhoneAppDelegate *appDelegate;
    NSInteger iterationCountAtSeed;
    int pulseXCoord;
    int pulseYCoord;
    int pulseZCoord;
    float zNormalDistributionValue;
    float sinusoidValue;
    float scalingValue;

    NSInteger pulseExcitationMode;
    NSInteger pulseBandwidth;

    BOOL pulseCompleted;
}

- (id)initWithXCoord:(int)pulseXPos YCoordinate:(int)pulseYPos iterationCount:
    (NSInteger)count excitationMode:(NSInteger)excitation;
- (id)initWithXCoord:(int)pulseXPos YCoordinate:(int)pulseYPos iterationCount:
    (NSInteger)count excitationMode:(NSInteger)excitation wavelength:(NSInteger)
    wavelength;
- (float)normalDistWithX:(float)x mean:(float)mean stdDev:(float)stdDev;
- (void)repeatPulse;

@end
```

Figura 17: archivo de cabecera de la clase *PulseObject*.

Esta clase implementa la función de crear el pulso de onda para insertarlo sobre la malla como un objeto, siempre distinguiendo de acuerdo al tipo de excitación que esté en ejecución. Es decir, según el tipo de excitación que se este ejecutando sobre la aplicación, esta clase crea un pulso del tipo de onda de la excitación que se insertará sobre la malla, creando así la onda y su propagación.

Como se ve en la anterior figura, dispone de varias variables: *TLMiPhoneAppDelegate *appDelegate, NSInteger iterationCountAtSeed, int pulseXCoord, int pulseYCoord, int pulseZCoord, float zNormalDistributionValue, float sinusoidValue, float scalingValue, NSInteger pulseExcitationMode;, NSInteger pulseBandwidth y BOOL pulseCompleted*. Entre ellas cabe destacar los tres enteros (int) *pulseXCoord, pulseYCoord* y *pulseZCoord*, que son las coordenadas donde ira el pulso creado. Las demás variables sirven principalmente de valores para la formación de los pulsos.

Además de las variables antes comentadas, también se encuentra los métodos:

- *(id)initWithXCoord:(int)pulseXPos YCoordinate:(int)pulseYPos iterationCount: (NSInteger) count excitationMode:(NSInteger)excitation;* -> se encarga de iniciar el impulso en un determinado punto que se le pasa como argumento, con el correspondiente tipo de excitación.
- *(id)initWithXCoord:(int)pulseXPos YCoordinate:(int)pulseYPos iterationCount: (NSInteger) count excitationMode:(NSInteger)excitation wavelength:(NSInteger)wavelength;* -> es una modificación del método anterior en el que inicia el impulso con una determinada longitud de ondas que se le pasa como argumento.
- *(float)normalDistWithX:(float)x mean:(float)mean stdDev:(float)stdDev;* -> método encargado de obtener la función de distribución Gaussiana que da forma a la onda de tipo de excitación Gaussiana, de acuerdo a los parámetros que se le pasan como argumento. En este caso son la media (mean), y la desviación típica (stdDev).
- *(void)repeatPulse;* -> este usa los métodos anteriores para formar el correcto pulso, pasando los adecuados parámetros que son calculados por *repeatPulse.* Es decir, forma el impulso de acuerdo al tipo de excitación, usando los métodos anteriores y llama al método que inyecta el impulso sobre la malla.

Como última clase de la carpeta *Classes* está *BoundaryNode*. Esta clase es la encargada de configurar e iniciar los bordes de la malla aplicándoles además el distinto tipo de condición del borde. Como se ha visto en variables de clases anteriores, existen tres tipos de nodos en los bordes de acuerdo a su condición: Perfecto Conductor Magnético (PMC), Perfecto Conductor Eléctrico PEC y Matched Termination (Conductor MTC). La diferencia entre ellos es la capacidad que tienen para reflejar o absorber las ondas que le llegan.

```
#import <Foundation/Foundation.h>
#import "TLMiPhoneViewController.h"

@class TLMiPhoneAppDelegate;

@interface BoundaryNode : NSObject {
    TLMiPhoneAppDelegate *appDelegate;
    NSInteger boundaryXCoord;
    NSInteger boundaryYCoord;
    NSInteger boundaryZCoord;
    NSInteger meshEdgeType;
    float boundaryCondition;
}

@property (nonatomic) NSInteger boundaryXCoord;
@property (nonatomic) NSInteger boundaryYCoord;
@property (nonatomic) NSInteger boundaryZCoord;
@property (nonatomic) float boundaryCondition;
- (id)initBoundaryAtXCoord:(NSInteger)x YCoordinate:(NSInteger)y
  ZCoordinate:(NSInteger)z withCondition:(BOOL)cond isEdge:(NSInteger)meshEdge;
- (void)resetNodeBoundaryCondition;

@end
```

Figura 18: contenido del fichero de cabera de la clase *BoundaryNode*.

Como se ve en la tabla siguiente, dispone de varias variables y propiedades cuya funcionalidad se describe a continuación:

Variables	Descripción
boundaryXCoord, boundaryYCoord y boundaryZCoord	Indican el punto en el que se inicia el borde (tipo NSInteger).
meshEdgeType	Indica de qué tipo de borde se trata, esto es TOP,BOTTOM,LEFT RIGHT,UPPER o LOWER, que corresponden con los seis tipos de bordes que existen en una malla tridimensional (tipo NSInteger).
boundaryCondition	Indica la condición del borde, pudiendo ser PMC, PEC o MTC (tipo float) .

Tabla 6 : variables de la clase BoundaryNode.

Por otro lado, implementa los dos siguientes métodos:

- *(id)initBoundaryAtXCoord:(NSInteger)x YCoordinate: (NSInteger)y ZCoordinate: (NSInteger)z withCondition: (BOOL)cond isEdge:(NSInteger)meshEdge; ->* inicia el nodo del borde del punto que se le pasa como argumento, aplicándole además el tipo de condición deseado.
- *(void)resetNodeBoundaryCondition; ->* reinicia el tipo de condición del nodo del borde desde el que es llamado.

Si las clases vistas hasta ahora son las clases encomendadas a la tarea de trabajar con la información de la aplicación, a continuación se encuentran las clases contenidas en la carpeta *ViewControllers*. Esta carpeta está dedica a los controladores de la vista, es decir, aquellas clases que enlazan los datos trabajados de las anteriores clases con la interfaz gráfica. A grandes rasgos, y recordando el patrón de diseño MVC presentado en capítulo 2, *ViewControllers* contiene las clases que pertenecen al controlador, mientras que las anteriores corresponderían con el modelo. Por lo tanto, estas clases se encargan de modificar el código de las clases modelo de acuerdo a lo que se selección por la interfaz gráfica, o viceversa, es decir, de modificar la interfaz gráfica de acuerdo a alguna modificación en el código.

4.2.2.3 ViewControllers

Como se ve en la figura ,existen 6 clases controladoras, una por cada una de las interfaces gráficas de la aplicación, ya que en iOS, por cada una de sus interfaces, debe haber un controlador correspondiente. Sin embargo, como causa de la extensión de las clases, se va a especificar sus funciones, y a destacar los puntos mas significativos.

- *TLMiPhoneViewController:* es la clase controladora de la interfaz principal con la que aparece la aplicación. Además de controlar la serie de elementos que aparecen sobre la pantalla como botones, etiquetas o imágenes, cabe destacar la tarea que desempeña un método en especial*, - (void)drawToScreen:(NSTimer*)timer.* Su principal, y fundamental, labor es la de obtener los datos calculados con el algoritmo TLM de la clase *TLMSolver*, creando la imagen y exponiéndola sobre la pantalla, continuamente. De esta forma, mostrando la imagen de una forma iterativa, se sucede a la propagación de la onda sobre la pantalla. Por otro lado, otra

destacable función es la de poder introducir varios pulsos de onda a la vez, realizando un doble pulso sobre la pantalla, o introducir nuevos bordes dentro de la malla, manteniendo el del pulsado una vez estando dentro del estado de ingresar varios pulsos de ondas. De esto se encargan métodos como *-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event, -(void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event* o *-(void)handleLongPress:(UILongPressGestureRecognizer *)gesture.*

- *ConfigureOptionsViewController:* es la clase controladora de las opciones de configuración del simulador, así como de la elección entre los distintos modos de simulación y excitación de ondas. Controla la selección de modos y ofrece la posibilidad de pasar a la configuración de los bordes de la malla o cargar algún tipo de excitación pre-cargado vía servidor.
- *ChangeBoundariesViewController:* se encarga de ofrecer la posibilidad de cambiar el tipo de condición de los diferentes bordes de la malla. Además, aquí también se proporciona la opción para poder insertar nuevos bordes dentro de la malla estando dentro del modo de inserción múltiple.
- *EditServerAddressViewController:* es la controladora de la interfaz de la opción para cargar simulaciones de ondas pre-cargadas en el servidor, mostrando los distintos tipos a elegir.
- *ChangePlaneShownViewController:* aunque ha habido modificaciones en las clases anteriores, esta ha sido creada ex profeso para la actual aplicación. Como se ha visto en capítulos anteriores, la propagación tridimensional se basa en la propagación sobre una misma capa, y la propagación entre capas. Por ello, y para poder ver el efecto de las ondas sobre cada capa, y por tanto el efecto tridimensional, se ha implementado un botón sobre la pantalla principal, el cuál permite al usuario seleccionar la capa que se desea visualizar en la pantalla, directamente desde la aplicación. Este simple botón da una interesante e interactiva opción para poder ver los diferentes tipo de propagación de los distintos tipos de ondas, dando lugar a una mejor comprensión y entendimiento de los conceptos de propagación de las ondas tridimensionales. El contenido del archivo de cabecera de la clase es mostrado en la siguiente figura:

```
#import <UIKit/UIKit.h>

@class TLMiPhoneAppDelegate;
@protocol ChangeZViewDelegate;
@interface ChangePlaneShownViewController : UITableViewController <UIPopoverControllerDelegate>{
    TLMiPhoneAppDelegate *appDelegate;
    id <ChangeZViewDelegate> delegate;
    UIPopoverController *popOver;

}
@property (nonatomic, assign) id <ChangeZViewDelegate> delegate;
@property (nonatomic, retain) UIPopoverController *popOver;

@end

@protocol ChangeZViewDelegate <NSObject>

- (void)dismissPopOverViewPlane;
- (void)changePlaneShownButtonTitle:(NSString *)title;

@end
```

Figura 19: contenido del archivo de cabecera de la clase *ChangePlaneShownViewController*.

La implementación del botón muestra una tabla entre todas las capas disponibles a elegir, dependiendo del tamaño de la malla, cuando es pulsado. Una vez se selecciona la capa deseada, esta se muestra por pantalla, y además se notifica sobre una etiqueta sobre el botón, la cual muestra en cada instante la capa mostrada en ese momento.

4.3 Forma básica de funcionamiento

Una vez vistas las clases de implementación del programa, en esta sección se pretende realizar un resumen de la forma básica en la que trabaja la aplicación, o mas concretamente, el método de funcionamiento general que sigue el código de la aplicación, a grandes rasgos y sin entrar en todo tipo de detalles.

En primer lugar se selecciona un tipo de excitación de ondas entre las disponibles. Depende de la elegida, los impulsos de las ondas son creados automáticamente o creados por el usuario. De una forma u otra, los pulsos son transmitidos a las ondas con una cierta intensidad. Una vez se inserta en la malla, el algoritmo TLM es el encargado de calcular los voltajes de todos los nodos de una forma iterativa. En cada iteración, esos voltajes son transformados en valores de intensidades, a la que cada una, mediante una rutina de transformación, le corresponde un determinado color del píxel en la pantalla. De esta forma, y durante cada iteración, se presentan los voltajes sobre la pantalla, creando así la propagación de la onda.

4.4 Pruebas y resultados

En esta sección se describen en un primer lugar las diferentes pruebas con las que se ha testeado la aplicación a lo largo de su desarrollo y tras su finalización. Estas pruebas han consistido principalmente en el aprovechamiento de todas las herramientas que ofrece Xcode para el testeo de las aplicaciones en su desarrollo, así como también en la realización de distintos tests de la aplicación tanto sobre el dispositivo. Además de las pruebas a describir, también se presentan los resultados obtenidos tras realizarlas tratando de conseguir una conclusión sobre su funcionamiento en los distintos dispositivos.

En primer lugar, el proyecto ha sido desarrollado usando el software oficial de Apple para el desarrollo de aplicaciones iOS, Xcode, en un MacBook Pro. Xcode también ha sido utilizada para testear la aplicación en cada momento a lo largo de la realización del proyecto para asegurar el apropiado funcionamiento, debido a la complejidad del modelado de la aplicación. Para ello, se han creado diferentes métodos y técnicas de comprobación y testeo, como pueden ser la representación de los valores obtenidos con el cálculo de la propagación de un tipo de onda en una pequeña malla, para comprobar la correcta propagación, o métodos que testean el adecuado acceso a la información guardada en los arrays.

Además, Apple proporciona una inmensa cantidad de herramientas, junto a Xcode, para realizar un exhaustivo test y control de las aplicaciones desarrolladas. Como ha sido mencionado en los primeros capítulos, una de ellas y la cual ha sido amplia y continuamente utilizada en el proyecto es *iOS Simulator*, un simulador que permite ejecutar, lanzar y probar cada aplicación directamente sobre el ordenador, pudiendo elegir entre emular en un iPhone o iPad, sin la necesidad de ningún otro dispositivo. Esto ha hecho posible observar diferentes problemas y fallos encontrados a lo largo del desarrollo de la aplicación. Sin embargo, y como será explicado posteriormente, pueden haber considerables diferencias en el comportamiento entre el simulador y los dispositivos reales, debido a la diferencia de potencia del hardware entre los ordenadores y los dispositivos móviles.

Otra, y muy útil herramienta es *Instruments tools*. Este software da la posibilidad de realizar un profundo análisis y estudio sobre el comportamiento de la aplicación. Por este motivo, ha sido usado

para comprobar el uso de la memoria de la aplicación, así como el rendimiento gráfico, siendo posible determinar el origen de algunos fallos relacionados con la reserva de memoria en los arrays.

En adición, de acuerdo a la posible diferencia de rendimiento de la aplicación entre el simulador y los dispositivos, la aplicación ha sido testeada en un iPad mini, obteniendo algunos puntos a analizar.

Como era esperado, debido a la inmensa cantidad de datos procesados en la aplicación, y debido a la tridimensionalidad de las ondas, el rendimiento sobre el iPad mini es bastante menor que en el simulador, siendo mucho mas lento en el dispositivo, por su menor potencia en hardware. El iPad mini posee una memoria RAM de 512 Mb y un procesador A4, mientras que el MacBook Pro tiene 4 Gb de memoria RAM, y un procesador mucho mas rápido y potente. Esto hace que la gran cantidad de datos se procesen rápidamente y sin problemas sobre el simulador, pero no tanto sobre el iPad mini.

Asimismo, gracias a las pruebas realizadas sobre el iPad mini, y tras comprobar su poca fluidez con una gran malla, se ha podido extraer su origen. Además de la diferencia de potencia en hardware, se ha visto que algunos algoritmos que se han tomado de la aplicación original no son eficientes. En otras palabras, algunas implementaciones aplicadas de la aplicación base han resultado no estar lo suficientemente bien diseñadas, al no seguir el patrón MVC, ni ser lo suficientemente eficientes ya que, y aunque funcionan de una manera perfecta en la aplicación base, reservan memoria innecesaria e inútil. Al trabajar en dos dimensiones, en caso de la aplicación base, es poca la memoria que se reserva innecesariamente. Sin embargo, en este proyecto en particular, y al estar trabajando con información en tres dimensiones, esa innecesaria reserva de memoria se multiplica, siendo demasiada para el procesamiento de la aplicación por el dispositivo. Debido a esto se han pensado formas de solventarlo, y será explicado en capítulos posteriores concernientes a ello.

Todo lo anterior provoca que se deba configurar un diferente tamaño de la malla para cada dispositivo, de acuerdo a su potencia de rendimiento, para lograr un rendimiento adecuado. Por ello, se ha creado una malla menor para el iPad mini, es decir, se han disminuido el número total de nodos de la red. Con esto se consigue disminuir el tamaño de la malla, y con ello reducir la información procesada por la aplicación, mejorando su rendimiento y proporcionándole mas rapidez y fluidez.

Probar la aplicación no solo en el simulador, si no también sobre el iPad mini, ha confirmado lo que se esperaba previamente; la aplicación desarrollada puede funcionar de una manera diferente dependiendo el dispositivo usado, y por lo tanto, dependiendo de su potencia en hardware. Obviamente, cuanto mas potente sea el dispositivo en la que se ejecuta la aplicación, mas rápido y mejor funcionará, ya que manejará y trabajará mejor los datos procesados.

Por ello, y aunque los avances en la tecnología con las actualizaciones de los dispositivos presentan grandes progresos y mejoras en el hardware de estos, lo que proporciona un mayor rendimiento y eficiencia en el sistema operativo y las aplicaciones, se ha pensado en diferentes modificaciones y optimizaciones a desarrollar e implementar en la aplicación, las cuales serán presentadas en capítulos posteriores.

Capítulo 5

5. Guía de usuario

En este sexto capítulo se detallará la forma de uso de la aplicación, explicando cada pantalla de la interfaz y cada elemento de ella, así como las diferentes características que ofrece y presentando la correcta manera de configurarlas a través de un breve tutorial. Con esto se pretende introducir la aplicación al usuario, facilitándole la entrada a la aplicación y ayudándole a la utilización y al total aprovechamiento de la misma.

5.1 Tutorial de uso

Figura 20: pantalla de carga de la app.

Figura 21: pantalla inicial de la app. Modo Gaussiano

En primer lugar, la aplicación carga con el modo de excitación de ondas Gaussianas por defecto, incluyendo además en pequeño tutorial de introducción al uso de la aplicación que mejora la experiencia de usuario. Sobre la pantalla, destacan las tres ventanas donde aparecerán las componentes de las ondas, de tipo gaussiano en este caso, al pulsar sobre alguna de ellas e introduciendo así impulsos de ondas.

Por otro lado, en la esquina superior derecha, se puede ver un botón que indica la capa del modelado 3D que se está viendo sobre la pantalla, y en el cual al pulsar sobre él podemos intercambiar la capa para mostrar otra diferente.

Por último, contiene tres botones en la parte inferior, que sirven para resetear el estado de las ondas que el usuario ha podido introducir, acceder al menú de opciones y pausar o continuar el estado de las ondas, respectivamente.

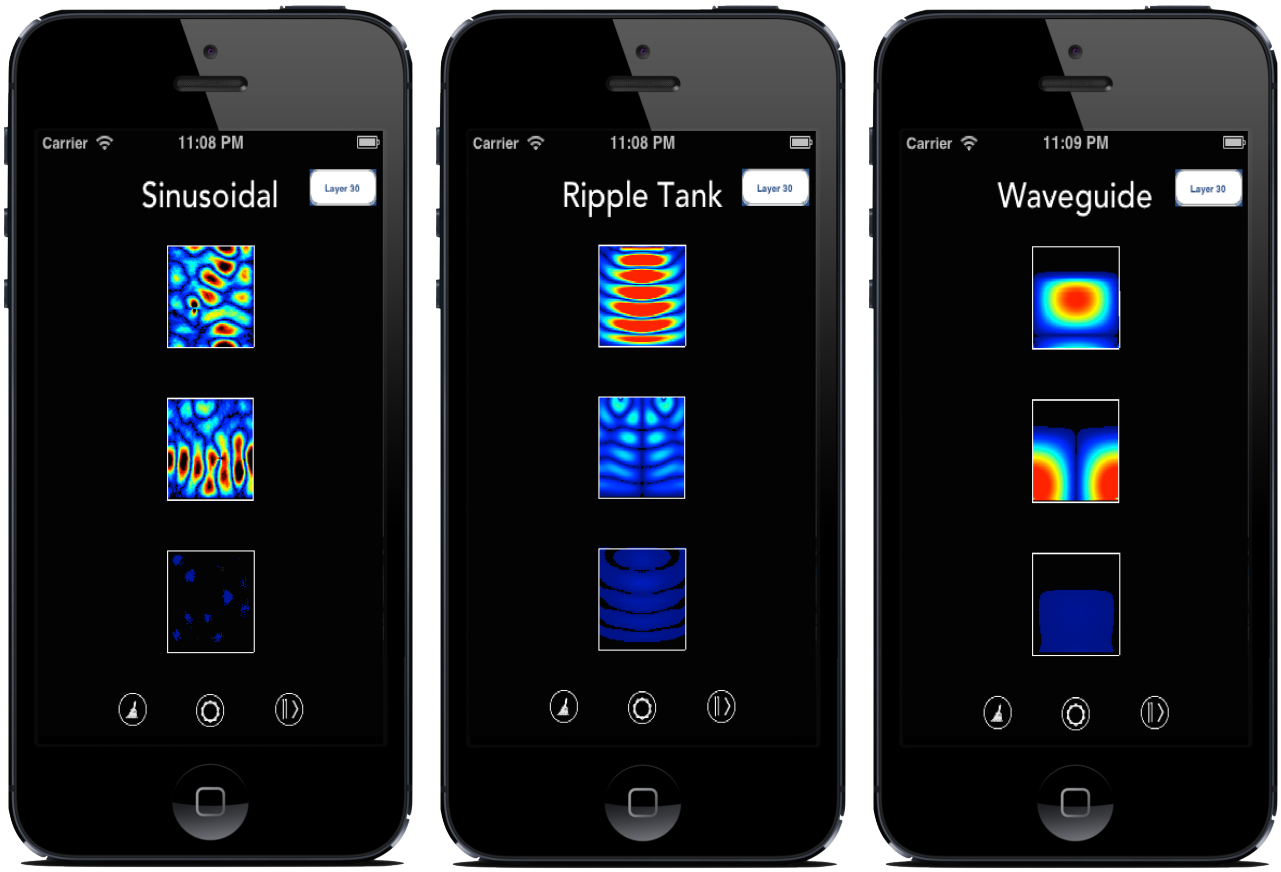

Figura 22: pantallas de los modos de excitación de ondas Sinusoidales, Tanque de Ondas y Ondas Guíadas.

Además del modo Gaussiano cargado por defecto como pantalla inicial, a través de deslizamientos hacia los lados se puede intercambiar entre los distintos tipos de modos de excitación de ondas. Como se puede ver en la figura 22, es posible elegir también entre ondas sinusoidales, ondas en una cubeta u ondas guiadas, además del modo que se puede cargar a través del servidor, que será visto posteriormente.

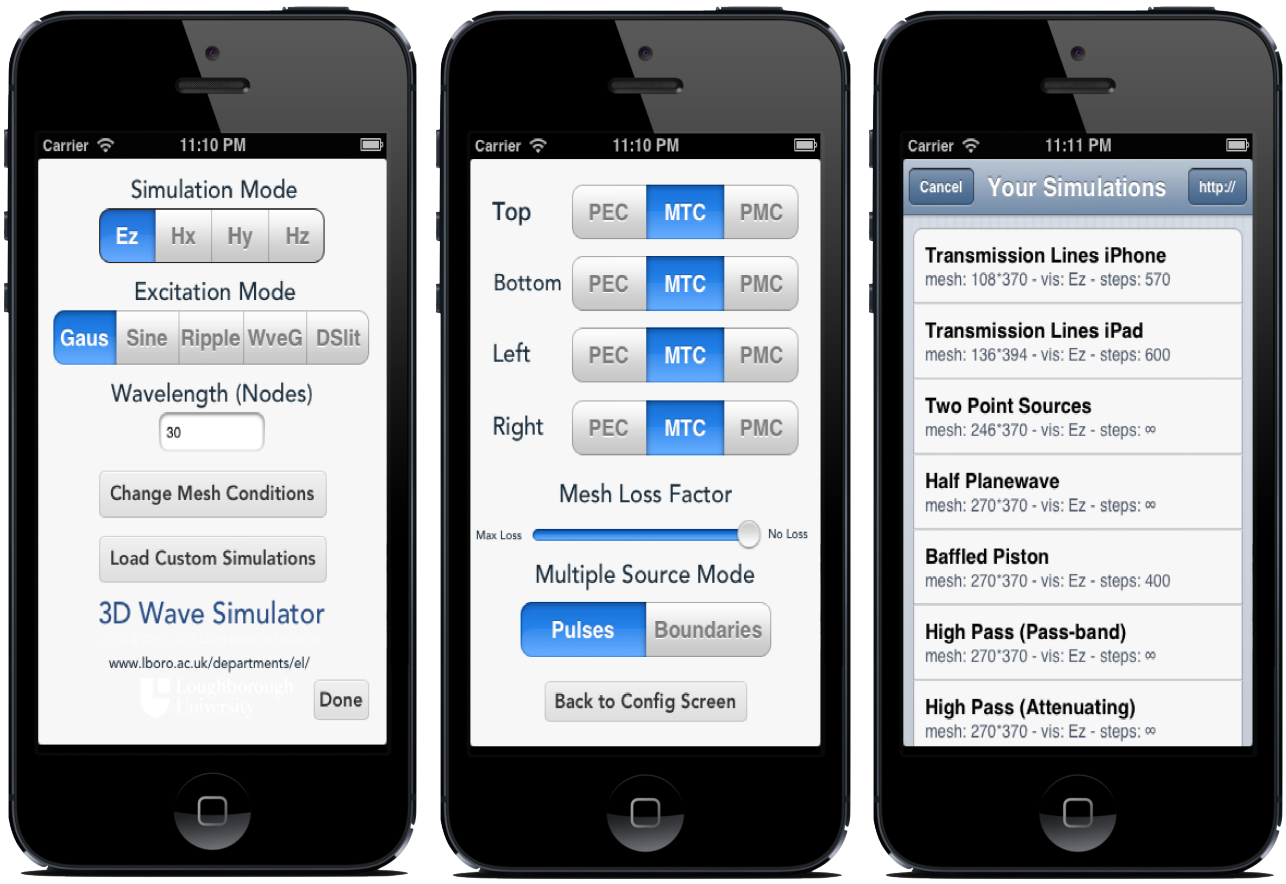

Figure 23: pantallas del menú de opciones, configuración de las ondas y la carga de modos de excitación de ondas.

Si pulsamos en el botón de opciones anteriormente mencionado, se accede al menú de opciones, visualizado a la izquierda en la figura 23. Este menú ofrece distintos menús para cambiar entre modos de excitación de ondas o incluso cambiar distintos parámetros de los mismos. Por ejemplo, desde aquí también es posible cambiar entre los distintos tipos de modo de excitación de ondas además de a través de deslizamientos laterales en la pantalla principal.

Por otro lado, también permite modificar diferentes parámetros como la longitud de la onda del tipo de excitación de onda que esté reproduciéndose en ese momento. Además, podemos enlazar a otro menú que le permite al usuario configurar los distintos parámetros de la malla, como la pérdida de las ondas o los tipos de bordes.

Además, también cabe destacar otra opción presente en el menú de opciones. Éste es la posibilidad de cargar simulaciones ajenas a la aplicación a través de un servidor, que nos vienen descritas en otra pantalla al pulsar el botón correspondiente para cargar las simulaciones personalizadas.

Capítulo 6

6. Conclusión y líneas futuras

Este capítulo está dedicado a la explicación de las conclusiones y resultados que se han obtenido a lo largo y tras la finalización del proyecto, tanto positiva como negativamente. Además, también se detallarán algunos aspectos y mejoras a perfeccionar e introducir en la aplicación como trabajo de futuro en relación al proyecto desarrollado.

6.1 Conclusión

Como conclusiones, y tras un largo desarrollo del proyecto, son muchos los resultados que se pueden extraer.

En primer lugar, la realización del proyecto final de carrera ha sido un duro reto a la vez que un interesante proceso. A pesar de su dificultad, se ha proporcionado una gran oportunidad para aprender a desarrollar una aplicación para la plataforma iOS usando Objective-C como lenguaje de programación. Además, su realización también ha servido para adquirir diferentes conocimientos y aprender sobre ámbitos desconocidos, como pueden ser conceptos técnicos sobre la propagación de ondas electromagnéticas y el método de transmisión de la línea (TLM).

En adición, se han adquirido diversas técnicas para el desarrollo de software teniendo en cuenta las limitaciones de los dispositivos portables, donde el hardware es limitado. Cabe decir también que se ha obtenido una invaluable experiencia a lo largo del desarrollo del proyecto, aprendiendo diferentes métodos y técnicas sobre creación, diseño, testeo y evaluación de una aplicación.

Aunque a priori puede parecer ser beneficioso basarse en otra aplicación, esto también ha demostrado tener sus inconvenientes. Si bien es cierto que ayuda a tener una idea de cómo desarrollar la aplicación, también hay que decir que esa idea llega tras entender totalmente el funcionamiento de la aplicación base, algo que, debido a su gran complejidad, no es tarea fácil. La aplicación original fue desarrollada a lo largo de varios años antes de su versión final, por lo que conocer y entender todas sus partes no se logra en un corto período de tiempo, siendo esto imprescindible para poder usarla como base. Además, esta aplicación hace uso de potentes librerías, y una gran cantidad de frameworks y métodos de iOS, por lo que hace una total explotación del sistema operativo. Por ello, se ha tenido que realizar un profundo y amplio aprendizaje de programación en iOS antes de ser capaz de poder entender el funcionamiento general de la aplicación, así como particularmente cada fragmento de código. Asimismo, tras el desarrollo de la aplicación del proyecto, se han visto fallos en su funcionamiento debido a una ineficiente

implementación en la aplicación original que se ha aplicado a la presente aplicación, como se ha explicado en capítulos posteriores. Esto da lugar a mejoras y modificaciones que se han contemplado, y que serán detalladas posteriormente. Por todo ello, basarse en la aplicación original ha servido de ayuda para obtener ideas acerca de la implementación de la aplicación, pero también ha llevado un duro y fatigoso trabajo a la par que finalmente ha mostrado la ineficiencia que de algunas de las implementaciones que se han aplicado de la aplicación original.

Por otro lado, y centrándose en el proyecto en particular, se han conseguido un gran número de objetivos. Después de aprender sobre programación para la plataforma iOS y todo lo que ello involucra, y además de comprender el funcionamiento de la aplicación base, se ha conseguido modificar el modelo bidimensional desarrollando así el modelo tridimensional. Aún teniendo una aplicación base, este tipo de modelo no tiene ningún tipo de referencia por lo que se ha desarrollado desde cero, teniendo que diseñar y modificar todos los aspectos que afectan a la correcta propagación de las ondas en tres dimensiones. Además, aprovechando la introducción de la tridimensionalidad de las ondas, se han incorporando nuevas características como la posibilidad de mostrar la capa deseada. Esta característica introducida hace posible ver la diferencia que existe en la propagación de las ondas entre capas, es decir, ver como el efecto tridimensional que provocan.

Por otro lado, también se han investigado diferentes formas para visualizar el resultado de la salida tridimensional. Una de las opciones contempladas ha sido la de visualizar en un vista en tres dimensiones usando OpenGL ES 2.0 [14], ya que permite una visualización en 3D y las herramientas necesarias para crear una “vista virtual” pudiendo añadir diferentes escenas a la propagación de las ondas en la aplicación. Además, a esta se le podría añadir una interesante e innovadora característica como es la introducción de Realidad Aumentada. Con ella, esta vista 3D se movería usando los sensores del dispositivo, moviéndose de acuerdo al movimiento descrito por el dispositivo usado. Sin embargo, la plataforma de OpenGL ES 2.0 para dispositivos móviles no es lo suficientemente potente como para desarrollar esta características en su perfección, por lo que se han contemplado otras opciones. Una de ellas, la cual ha sido escogida para la visualización de la aplicación como se ha detallado en capítulos anteriores, consiste en mostrar tres ventanas en la pantalla, mostrando cada una de ellas un plano diferente, dando lugar al resultado en 3D.

En resumen, la realización del proyecto ha sido una gran experiencia con la que se han obtenido numerosos conceptos técnicos, así como metodologías de trabajo. Se ha dado fruto a una aplicación educacional, que aún no estando en su versión final, puede ayudar para el entendimiento sobre propagación de ondas electromagnéticas tridimensionales y también para otros proyectos y aplicaciones futuras.

6.2 Futuras mejoras

Además del desarrollo de la aplicación descrito a lo largo de este documento, se han considerado una gran cantidad de mejoras y optimizaciones a implementar en la aplicación desarrollada para intentar conseguir un perfecto funcionamiento, en conjunción de nuevas características que proporcionen a la aplicación otra forma de interactuar con ella. A continuación se detallarán diferentes aspectos de la aplicación que deben ser modificados y pulidos, ya que la presente aplicación no se trata de la versión final.

En primer lugar, en capítulos anteriores se han explicado los diferentes inconvenientes que presenta la aplicación. Uno de ellos, es la falta de fluidez y la lentitud con la que la aplicación funciona en algunos dispositivos iOS debido al alto consumo de memoria trabajando con información tridimensional. Aunque era lo esperado, y algo que puede solucionarse con la rápida evolución que sufre el hardware de los dispositivos móviles, se ha considerado modificar la metodología de trabajo

de la aplicación, corrigiendo los patrones de diseño de la aplicación y disminuyendo así la cantidad de datos con la que la aplicación tiene que trabajar, y por tanto, ahorrando consumo de memoria y mejorando la rapidez y fluidez de la aplicación. Ejemplos de modificaciones pensadas que entrarían dentro de lo anterior pueden ser cambios en los algoritmos de la propagación de las ondas en las mallas, así como también cambios en la manera con la que se reserva y trabaja su información, evitando así reserva de memoria inútil y por tanto haciendo la aplicación mucho mas eficiente.

Además, recientemente Apple ha anunciado iOS 7, una nueva y revolucionaria versión de su sistema operativo para dispositivos móviles en la que ha realizado una gran serie de cambios, tanto en su comportamiento y prestaciones, como, y en la que destaca sin dudas, su total revolución en la interfaz grafica. Esto significa que para que la aplicación funcione en la nueva versión, iOS 7, ha de realizarse una actualización de la aplicación, utilizando las nuevas funciones que aprovechan y perfeccionan el funcionamiento de las aplicaciones en los dispositivos, así como adaptando y renovando la interfaz para su total correcto funcionamiento dentro del nuevo sistema operativo. Por ello, y aunque puede ser un duro trabajo, se actualizará en la menor brevedad posible, ya que todo está previsto para que Apple lance esta nueva versión en un muy cercano futuro, haciendo obligatoria la actualización de la aplicación para que funcione en todos los dispositivos que actualicen a la nueva versión de iOS.

Asimismo, con la introducción de iOS 7, también se ha incluido la actualización a OpenGL ES 3.0 [12], una multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D, que incluye potentes nuevas funcionalidades gráficas entre las que destacan mejoras en la renderización de texturas y sombras y la inclusión de características que hacen tratar a los elementos de una manera mucho mas eficiente y eficaz, dotando así a la aplicación de nuevas potentes herramientas y también facilitando su desarrollo. Esta nueva actualización facilitará el desarrollo de la vista 3D estudiada y anteriormente expuesta ya que proporciona las herramientas necesarias para su implementación, y de las cuales carecía la anterior versión dificultando su desarrollo hasta ahora. Además, con ello se podría incorporar el efecto de realidad aumentada anteriormente citado, lo que daría otra llamativa forma de interactuar con la aplicación.

Es por todo lo anterior, que esté en desarrollo una total actualización de la aplicación, y con ello, casi un total cambio en ella, para aprovechar todas las nuevas prestaciones que ofrece la nueva versión del sistema operativo de Apple que entre otras mejoras, optimizan el rendimiento de las aplicaciones. Las capacidades que ofrece iOS 7 se usarán tanto para mejorar los algoritmos de propagación de las ondas como para mejorar la reserva de memoria de la información, que servirá para hacer la aplicación mucho menos pesada, y por tanto mas rápida y eficiente, y mucho mas fluida sobre los dispositivos móviles. También su interfaz será renovada para adaptarla a la esencia de iOS 7, con interfaces que sean mas familiares facilitando así también el uso de la aplicación a los usuarios que la usen por primera vez. Además, se desarrollará uno de los métodos de salida estudiados y anteriormente mencionados para visualizar gráficamente la propagación tridimensional, pudiendo visualizar cada uno de los planos en diferentes mallas, o bien obteniendo una vista tridimensional de la propagación en una malla.

Una vez que la actualización en desarrollo se consiga y se obtenga la versión final, incorporando las mejoras y nuevas prestaciones que se han mencionado, está previsto lanzar la aplicación en el mercado de aplicaciones iOS, la AppStore. Con esto se pretende lograr el último objetivo, y por el que está pensada la realización de esta aplicación, que es la de facilitar el aprendizaje y entendimiento sobre la propagación de ondas electromagnéticas.

Por otro lado, la aplicación está totalmente en inglés, puesto que ha sido desarrollada en Inglaterra. Por ello, y aunque los textos existentes en la aplicación es de fácil entendimiento incluso para usuarios que no tenga un nivel de inglés muy avanzado, se ha considerado traducir la aplicación a un total español, facilitando así el aprendizaje de la aplicación a aquellos usuarios con un bajo nivel de inglés. Para ello se tiene en cuenta que Xcode, la aplicación oficial para el desarrollo en la plataforma iOS, dispone de un estupendo sistema para traducir todos los textos de la aplicaciones

desarrolladas a otros idiomas de una manera muy fácil, por lo que no sería un trabajo excesivo, y sin embargo si podría ser de una gran ayuda para los usuarios.

Finalmente, tras el desarrollo de cualquier tipo de software, se ha de llevar un mantenimiento y una continua actualización del software desarrollado. Por ello, se seguirá llevando a cabo un gran análisis para mejorar y optimizar pequeños bugs que puedan surgir, así como mantener las pertinentes cambios y adaptaciones correspondientes a las actualizaciones del sistema operativo, para lograr una total funcionalidad en todos los dispositivos disponibles de la compañía.

En adición, y además de la continuación del proyecto con todas las mejoras y optimizaciones ya expuestas, el proyecto realizado puede ser utilizado como base o complemento para diferentes proyectos que se desarrollen posteriormente en algún ámbito relacionado con la temática del proyecto presentado a lo largo del documento.

Referencias

[1] Cloud Computing: http://en.wikipedia.org/wiki/Cloud_computing.
[2] Smartphone : “ http://es.wikipedia.org/wiki/Teléfono_inteligente “.
[3] Google Inc.: http://es.wikipedia.org/wiki/Project_Glass.
[4] Daniel Browne: “Loughborough Wave Lab App”, “https://itunes.apple.com/us/app/loughborough-wave-lab/402424695?mt=8 “.
[5] Apple Inc. : “ http://www.apple.com/ios ”.
[6] Apple Inc. : “http://www.apple.com/es/ “.
[7] Google Inc. : “ http://www.android.com “.
[8] Centrix System : “ http://www.citrix.com/content/dam/citrix/en_us/documents/products/q4_enterpese_mobility _cloud_report.pdf “:
[9] Apple Inc. : “ http://www.apple.com/osx “.
[10] Unix : “ http://www.unix.org “.
[11] Apple Inc. : “ https://developer:apple.com/xcode “.
[12] Christos Christopoulos: “ The transmission-line modelling method TLM “, IEEE Press and Oxford University Press, 1995.
[13] Three20 : “ http://three20.info “
[14] Khronos Inc.: “ http://www.khronos.org/opengles/ “

Bibliografía

- P.B. Johns and R.I. Beurle: " Numerical solution of 2-dimensional scattering problems using a transmission-line matrix ", Proceedings of the Institution of Electrical Engineers, Vol.118, Nº 9, 1971, pp.1203-1208.
- Christos Christopoulos: " The transmission-line modelling method TLM ", IEEE Press and Oxford University Press, 1995.
- Apple Inc. http://www.developer.apple.com, último acceso en 10 de septiembre 2013.
- Joe Conway and Aaron Hillegass: " iOS Programming: The Big Nerd Ranch Guide ", The Big Nerd Ranch, 2011.
- David Mark, Jack Nutting, Jeff LaMarche, Fredrik Olsson: "Beginning iOS6 Development", Apress 2013.
- Stephen G. Kochan: "Programming in Objective-C, Fourth Edition", Addison-Wesley, 2012.
- Stanford University: " http://www.stanford.edu/class/cs193p/cgi-bin/drupal/ ", último acceso en 5 de septiembre de 2013.

Memoria en Inglés

En las páginas siguientes se muestra la memoria realizada en inglés para la exposición del proyecto en la Universidad de Loughborough.