UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

BelfegAR
Plataforma para el despliegue gráfico 3D y delegación de tareas en gestión documental con realidad aumentada

Santiago Sánchez Sobrino

Septiembre, 2014

BelfegAR

Plataforma para el despliegue gráfico 3D y delegación de tareas en gestión documental con realidad aumentada

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Tecnologías y Sistemas de Información

TECNOLOGÍA ESPECÍFICA DE
TECNOLOGÍAS DE LA INFORMACIÓN

TRABAJO FIN DE GRADO

BelfegAR
Plataforma para el despliegue gráfico 3D y delegación de tareas en gestión documental con realidad aumentada

Autor: Santiago Sánchez Sobrino

Director: Dr. Carlos González Morcillo

Septiembre, 2014

Santiago Sánchez Sobrino

Ciudad Real – Spain

E-mail universidad: Santiago.Sanchez3@alu.uclm.es
E-mail personal: sanchezsobrino@gmail.com

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL	SECRETARIO
Fdo.:	Fdo.:	Fdo.:

Resumen

La informática gráfica es una de las áreas de las tecnologías específicas de la rama informática que mayores posibilidades ofrece. Desde las lucrativas industrias del cine y los videojuegos, pasando por la visualización científica hasta el ámbito médico y sanitario, son indudables las mejoras que ofrece el uso de este área tecnológica.

Entre las diversas ramas de la informática gráfica y los entornos avanzados de interacción, la Realidad Aumentada es una línea de investigación en constante evolución y de las que mayor interés despierta. Son numerosas las aplicaciones de Realidad Aumentada existentes a día de hoy, que cubren un amplio abanico de posibilidades; desde guías interactivas en museos, sistemas de visualización GIS o videojuegos hacen uso de estos novedosos paradigmas.

El presente Trabajo Fin de Grado surge como parte del proyecto ARgos, cuyo objetivo es el uso de técnicas de Realidad Aumentada como ayuda a la gestión documental en usuarios con necesidades especiales. En este proyecto se plantea la construcción de un dispositivo embebido que de soporte a la detección de documentos y su posterior tratamiento empleando técnicas de realidad aumentada.

En este ámbito, BelfegAR se encarga de la gestión del despliegue multimodal en dispositivos de bajo coste y delegación de las tareas computacionalmente más costosas en servidores externos. En BelfegAR se desarrollará una plataforma completa y genérica que pueda ser usada para la construcción de este tipo de aplicaciones. En BelfegAR además se diseñará un demostrador tecnológico específico que muestre su funcionamiento en diferentes escenarios reales, definidos en el ámbito de la asociación Asprona (Asociación para la Atención a Personas con Discapacidad Intelectual y sus Familias de la Provincia de Albacete).

Abstract

Computer graphics is one of the areas of specific technologies of computer branch that offers more possibilities. From the lucrative industries of film and videogames, ranging from scientific visualisation to medical and healthcare, are undoubted improvements offered by the use of this technology field.

Among the various branches of computer graphics and advanced interaction environments, Augmented Reality is a research line constantly evolving and that bigger interest arouses. Numerous Augmented Reality applications exist nowadays, covering a wide range of possibilities; from interactive museum guides, GIS visualization systems or videogames that make use of these new paradigms.

This Final Degree Project comes as part of the ARgos project whose objective is the use of Augmented Reality techniques to support managing documents by users with special needs. In this project, the construction of an embedded device that supports detection and further processing documents using augmented reality techniques is proposed.

In this field, BelfegAR handles multimodal management deployment in low-cost devices and delegation of the most computationally expensive tasks on external servers. In BelfegAR a complete and generic platform that can be used to build such applications will be developed. Moreover, in BelfegAR will be designed a specific technology demonstrator to show its real functioning in different scenarios defined in the field of Asprona (Association for the Care of Persons with Intellectual Disabilities and their Families in the province of Albacete).

Agradecimientos

Me gustaría dedicar algunas líneas a todas las personas que han hecho posible el presente trabajo.

A mis padres, por su infinito apoyo en la vida personal y académica; sin ellos no habría llegado a donde he llegado hoy en día, ni probablemente cuando lea este documento dentro de algunos años.

A Sully, por demostrarme muchas veces ser mejor amigo que algunas personas.

A Verónica, por aguantarme y por su apoyo en los momentos difíciles que han surgido durante el desarrollo de este trabajo.

A Manuel, por su compañía, apoyo y amistad como compañero en Oreto.

A mis amigos de la infancia y de la carrera, los cuales saben quienes son, por su amistad desde que nos conocemos.

A Juan Carlos López López, por brindarme la oportunidad de formar parte del proyecto ARgos de la Cátedra Indra-UCLM.

Finalmente, a Carlos González Morcillo, por su apoyo y ayuda en el desarrollo de este Trabajo Fin de Grado y por ser uno de los mejores profesores y profesionales que me he encontrado a lo largo de la carrera.

Santiago

Al futuro.

Índice general

Índice de cuadros

Índice de figuras

Índice de listados

Listado de acrónimos

TFG	Trabajo Fin de Grado
TFC	Trabajo Fin de Curso
GNU	GNU is Not Unix
TTS	Text-To-Speech
API	Application Programming Interface
GPU	Graphics Processing Unit
CLI	Command Line Interface
GUI	Graphical User Interface
NUI	Natural User Interface
IPO	Interacción Persona-Ordenador
KISS	Keep It Simple, Stupid
SDL	Simple DirectMedia Layer
RPC	Remote Procedure Call
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IP	Internet Protocol
REST	REpresentational State Transfer
SOAP	Simple Object Access Protocol
HTTP	HyperText Transfer Protocol
XML	Extensible Markup Language
JSON	JavaScript Object Notation
GLM	OpenGL Mathematics
CLM	Configurable Math Library
CSI	Camera Serial Interface
ECB	Emacs Code Browser
SSH	Secure SHell

SOIL	Simple OpenGL Image Library
PNG	Portable Network Graphics
JPEG	Joint Photographic Experts Group
BMP	Bitmap
MVP	Model-View-Projection
FoV	Field of View
GLSL	OpenGL Shading Language
FBO	FrameBuffer Object
RAII	Resource Acquisition Is Initialization
BGE	Blender Game Engine
ARS	ARgos Script
GLUT	OpenGL Utility Toolkit
WYSIWYG	What You See Is What You Get
RAII	Resource Acquisition Is Initialization

Capítulo 1

Introducción

La evolución en la informática gráfica permitió el desarrollo de nuevas posibilidades a la hora de interaccionar de forma directa con un computador y recibir un *feedback* visual de este. Las Interfaces Gráficas de Usuario (GUI de su término en Inglés Graphical User Interfaces) mantienen una estrecha relación con los gráficos por computador porque uno de los principales mecanismos de interacción es el despliegue visual de información al usuario.

El uso de interfaces gráficas de usuario supone una mejora significativa frente a su predecesora, las Interfaces de Linea de Comandos (CLI de su término en Inglés Command Line Interface). De esta forma, el desarrollador idealmente debe desarrollar su software considerando de qué forma va a intervenir o interaccionar el usuario con este. Existe una rama específica de la informática que describe las técnicas de Interacción Persona-Ordenador (IPO), la cual define los siguientes principios para realizar una interfaz de usuario amigable para el usuario [Gre08]:

- **Reconocer los usuarios y tareas objetivos:** Establecer cuántos usuarios son necesarios para realizar las tareas y del mismo modo determinar qué personas serían las indicadas.
- **Medidas empíricas:** Realizar pruebas con usuarios finales de la interfaz.
- **Diseño iterativo:** Después de las etapas anteriores, realizar los siguientes pasos de forma iterativa:
    1. Diseñar la interfaz de usuario.
    2. Pruebas.
    3. Analizar resultados.
    4. Repetir.

Como evolución de las interfaces gráficas de usuario se introdujo el modelo de Interfaz Natural para el Usuario (NUI de su término en Inglés Natural User Interface). Dicho modelo se presenta al usuario de forma invisible para él, es decir, la interacción con la interfaz se realiza de manera natural para el usuario. Algunos ejemplos de interacción natural son el uso de la voz para controlar las diversas funcionalidades de un sistema, los dispositivos que

reconocen los gestos del usuario (como Xbox Kinect [1]), y los dispositivos que reciben su entrada a través de una interacción táctil (Tablets Android [2] o Microsoft Surface [3]), entre otros.

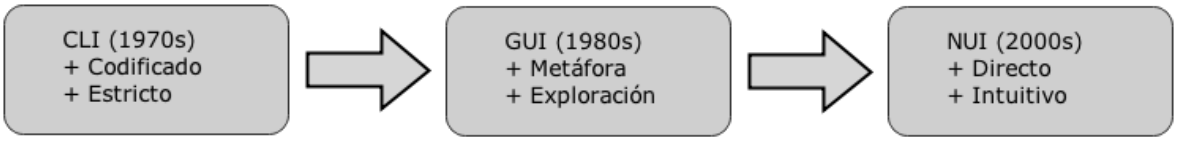

Figura 1.1: Evolución en los paradigmas de desarrollo de interfaces de usuario.

1.1 Interfaces de usuario naturales

Las NUI han evolucionado a lo largo de la historia hacia sistemas menos intrusivos y más amigables al usuario. La forma de interaccionar con un computador no se ha basado siempre en el uso de periféricos como teclado y ratón y sistemas de ventanas, sino que para alcanzar tal punto de la evolución, se han ideado desde los años 60 nuevos mecanismos de interacción y despliegue de la información más innovadores y sencillos de utilizar.

Las primeras interacciones entre el usuario y el computador se basaban en la manipulación directa de objetos gráficos, es decir, los gráficos de la pantalla eran manipulados directamente mediante un puntero físico. Este enfoque fue introducido por primera vez por Ivan Sutherland en su *Sketchpad*, producto de su tesis, en 1963. El sistema permitía la manipulación de objetos utilizando un lápiz óptico. Dicha manipulación incluía la posibilidad de mover los objetos y cambiar su tamaño, entre otras.

La problemática asociada al *Sketchpad* radicaba en el coste de los lápices ópticos. Al no resultar especialmente baratos se desarrolló «el ratón» como sustituto, en 1965. Este dispositivo fue popularizado gracias a su distribución junto al *Xerox Star*, en 1981, como dispositivo de entrada.

Por otra parte, y en lo relativo a las interfaces gráficas, se introdujo el concepto de «ventana» en 1968. En aquellos años las ventanas podían ser desplegadas de forma teselada; no fue hasta 1974 cuando se implementaron ventanas superpuestas. Más tarde se introdujeron los gestores de ventanas de la mano de Apple con sus computadores *Lisa* (el primer PC en ofrecer una interfaz gráfica en un computador barato) y *Macintosh*, y finalmente por Microsoft en sus últimas versiones de *Windows*. En 1984, el MIT desarrolló el estándar *X Window System*, implementado en *XFree86* y continuado en *X.Org Server* como una bifurcación del anterior.

Entre las aplicaciones que han hecho historia a lo largo de la evolución de la interacción

[1] http://www.xbox.com/es-ES/kinect
[2] http://www.android.com/
[3] http://www.microsoft.com/surface/es-es

entre persona y computador se distinguen algunas categorías [Mye98], como:

- **Aplicaciones de dibujo:** Como se ha comentado anteriormente, el sistema *Sketchpad* fue pionero en el uso de gráficos manipulables por el usuario, pero no fue hasta el lanzamiento del software *Superpaint* en 1973 cuando se consolidó una aplicación real de dibujo por computador.
- **Editores de texto:** Entre los primeros editores y procesadores de texto, se encontraba en 1962 el propuesto por Douglas Engelbart, el cual permitía búsquedas de palabras, *macros* definidas por el usuario, desplazamiento de texto y comandos para mover, copiar y eliminar caracteres. El primer editor «lo que ves es lo que obtienes» (WYSIWYG de su término en Inglés What You See Is What You Get) llegó de la mano de Xerox PARC con su software *Bravo*, en 1974.
- **Hojas de cálculo:** La primera hoja de cálculo en aparecer fue desarrollada y lanzada para el *Apple II* en 1979 fue *VisiCalc*. Es considerada la aplicación estandarte del *Apple II*, vendiendo más de 700.000 copias en 6 años. Sin embargo, en 1983, y con la salida de la aplicación *Louts 1-2-3*, las ventas se redujeron dramáticamente, provocando la compra de la compañía por parte de *Lotus Development*.
- **Hipertexto:** La idea del hipertexto surgió en 1945 del concepto *MEMEX*, de Vannevar Bush, mientras que la palabra hipertexto fue creada por Ted Nelson en 1965. Bush describió el *MEMEX*, en su artículo «As We May Think», como un dispositivo en el cual los usuarios podrían comprimir y almacenar todos sus libros, grabaciones y comunicaciones, para después ser recuperados de manera fácil y rápida.

Sin embargo, desde la aparición del *Apple Lisa* en la década de los 70 ha cambiado muy poco la forma en que interactuamos con un computador, refiriéndonos al teclado y el ratón. Si avanzamos hasta finales de los años 2000s vemos cómo se ha adoptado progresivamente el uso de dispositivos móviles táctiles hasta el punto de cambiar la forma en que interactuamos con los computadores.

Si vamos más allá vemos cómo el uso de nuevos dispositivos pretende desplazar al ordenador personal como principal medio de computación. El proyecto Google Glass [4] es un claro ejemplo de esto, donde se intenta moldear una capa *informatizada* superpuesta ante nuestra visión. Esto supone aumentar las capacidades del ser humano, de una forma invisible y no intrusiva.

El próximo paso evolutivo de las NUIs, comprende la integración de la informática en el entorno del ser humano. Para ello, mantiene una estrecha relación con la computación ubicua, de tal forma que un entorno basado en NUIs pueda ser establecido mediante pequeños dispositivos heterogéneos distribuidos en el ambiente. Estos pequeños dispositivos, habitualmente de pequeño tamaño y bajo coste, poseen unas capacidades de cómputo reducido,

[4] http://www.google.com/glass/start/

por lo que es necesario contar con mecanismos de delegación de tareas en dispositivos con mayores capacidades de cómputo.

1.2 Contexto

BelfegAR trata el ámbito social que concierne a la informática gráfica y las NUI para ayudar a resolver aquellos problemas relacionados con la representación y la visualización de la información. BelfegAR nace como una parte de un proyecto mayor, **ARgos**, que trata la problemática de la **gestión documental** basada en visión por computador y realidad aumentada. Mediante dichas tecnologías, el sistema identifica el documento con el que está trabajando el usuario, así como la interacción directa que se realiza sobre el mismo. Un sistema de cómputo analiza la entrada y muestra información ampliada multimodal (visual y auditiva) sobre el nodo de información de trabajo actual.

El proyecto de investigación se desarrolla en el ámbito de la **Cátedra Indra de la UCLM** [5], y se ha realizado en conjunción con Manuel Hervás Ortega, quien se ha ocupado del desarrollo del sistema de reconocimiento de documentos y su interacción mediante visión por computador con su proyecto **GrayAR: Técnicas de visión por computador y realidad aumentada aplicadas a la gestión documental**, cuya defensa se realizaŕa en la próxima convocatoria de Trabajos de Fin de Grado de la Escuela Superior de Informática.

1.3 Entorno

BelfegAR resuelve la problemática de despliegue de gráficos 3D y amplificación multimodal sobre la plataforma de producción, ofreciendo una interfaz de usuario lo más clara y sencilla posible. Además, también se encarga de delegar tareas computacionalmente costosas en otras máquinas servidoras que pueden realizar el procesamiento con los requisitos de tiempo real que necesita el sistema en explotación.

El entorno de ejecución se compone de un proyector utilizado como principal soporte de visualización de los componentes gráficos, un sistema de cómputo de bajo coste y tamaño reducido (de ahora en adelante, la unidad de procesamiento) que facilita la portabilidad del sistema, un sistema de computación auxiliar que de soporte en red a la unidad de procesamiento, y una minicámara utilizada para la adquisición de imágenes.

Las tareas que realizan cada uno de los distintos elementos del entorno se encuentran definidas de la manera siguiente (ver Figura 1.2 que muestra un esquema de los principales componentes utilizados en ARgos):

- **La unidad de procesamiento:** Se trata de un dispositivo «Raspberry Pi» utilizado expresamente para el renderizado de gráficos acelerados por hardware gracias a su GPU. Se encarga además de la reproducción de audio y de mantener una comunicación

[5] http://catedraindra.uclm.es/argos

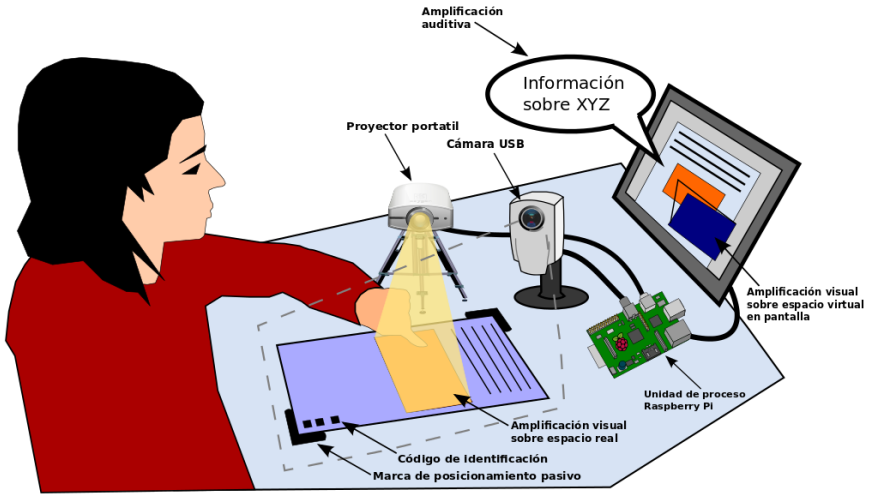

Figura 1.2: Componentes del sistema ARgos.

en red sin pérdidas con el sistema auxiliar.

- **El proyector:** Se emplea como dispositivo de salida de vídeo por parte de la unidad.
- **La minicámara:** Se emplea para realizar la adquisición de imágenes, que son utilizadas de dos maneras distintas: durante las sesiones de vídeollamada por BelfegAR (1) y para su posterior procesado por GrayAR (2).
- **El sistema de cómputo auxiliar:** Se encarga de las tareas que supongan una elevada carga para la unidad. Dichas tareas abarcan desde el procesado de las imágenes captadas por la minicámara, hasta la lectura y procesado de los distintos *scripts*, pasando por mantener y desplegar una interfaz de usuario para la configuración del sistema.

1.4 Impacto socio-económico

Los nuevos paradigmas de interacción han revolucionado diversos sectores. El uso de paneles táctiles ha cambiado totalmente la forma en que interaccionamos con los dispositivos móviles como tabletas y *smartphones*. En España el 66 % de la población tenía acceso a un *smartphone* en 2013 encabezando así la lista de países de la Unión Europea en uso de *smartphones* [6]. El principal éxito de esto se basa en el uso de pantallas táctiles que facilitan la interacción del usuario con el dispositivo, simulando tener un ratón en un dispositivo de tamaño reducido pero con una potencia cercana a la de un computador.

En la industria de los videojuegos, se pueden diferenciar varios modos de interacción natural:

- Videojuegos que hacen uso de realidad aumentada, como *Invizimals* para PSP, que mediante el uso de una cámara y tarjetas impresas, se puede interaccionar con el mundo

[6] `http://www.comscore.com/esl/Insights/Presentations-and-Whitepapers/2013/2013-Spain-Digital-Future-in-Focus`

físico desde el propio videojuego. Desde su lanzamiento se ha vendido más de medio millón de unidades [7]. Además, el videojuego cuenta con su propia serie de televisión y *merchandising* como álbumes de cromos y juguetes.

- Videojuegos que emplean nuevos dispositivos de interacción, como *Guitar Hero*, que mediante el uso de un periférico que simula ser una guitarra eléctrica, permite jugar como si de un controlador estándar se tratara. Supuso un éxito en ventas, situándose en más de 2 millones de unidades vendidas [8].
- Videojuegos para la videoconsola Wii, que supuso un éxito de ventas (casi 100 millones de unidades vendidas [9]) y supo abarcar mayor mercado que la competencia gracias a los nuevos paradigmas de interacción gestual que utiliza.
- Videojuegos que hacen uso de *Kinect*, como *Kinect Adventures!*, que mediante el uso de este dispositivo ofrecía una innovadora experiencia de juego. Esto supuso más de 20 millones de unidades vendidas [10].

Como se puede observar de la lista anterior, todos los éxitos fueron en mayor o menor medida gracias al uso de nuevos paradigmas de interacción entre el usuario y el sistema.

El presente trabajo, dentro del proyecto del cual forma parte, supondría un impacto social y económico directamente proporcionales entre si. Gracias a la naturaleza del sistema se permitiría la integración en empresas de personas con necesidades especiales, entre las que se reconocen de ámbitos visuales, auditivas e intelectuales:

- En el caso de algún tipo de discapacidad visual, el sistema puede emitir retroalimentación auditiva al usuario, que le ayude y guíe durante su trabajo con el documento.
- Para el caso de discapacidad auditiva, el sistema puede funcionar prácticamente en su totalidad empleando renderizado de gráficos y texto sobre el documento mediante el proyector.
- Finalmente, y en caso de que se produzca un caso de discapacidad intelectual, el usuario puede simplemente iniciar una videollamada a través del propio sistema con el centro de soporte para que le ayude en su desempeño. Si esta opción no se encuentra disponible en ese momento, se indicará mediante guías visuales y/o auditivas la secuencia de instrucciones que debe de seguir el usuario del sistema para completar su trabajo.

Como vemos, el proyecto plantea algunas ayudas y soluciones a multitud de tipos de necesidades especiales en el ámbito de la inserción laboral. De esta forma, personas con ciertas necesidades especiales pueden llegar a pertenecer a la población activa y realizar un

[7] `http://www.vgchartz.com/game/35139/invizimals/`
[8] `http://www.vgchartz.com/game/924/guitar-hero/`
[9] `http://www.vgchartz.com/weekly/41315/Global/`
[10] `http://www.vgchartz.com/game/45608/kinect-adventures/`

trabajo que de otra forma resultaría mucho más complicado. Por otro lado, los resultados del proyecto pueden ser igualmente interesantes para el resto de la población, que puede igualmente beneficiarse del uso de estas técnicas de amplificación de la inteligencia.

1.5 Problemática

El desarrollo de este Trabajo Fin de Grado supone el despliegue de información gráfica que debe ser localizada a partir de la posición de un documento existente en el mundo físico. Este requisito puede llegar a suponer la aparición de problemas de diversa índole, como los relacionados con las capacidades de la plataforma de despliegue (como pueden ser problemas de rendimiento o de funcionalidad), facilidad de uso por parte del usuario final, latencia de red, y conexiones con el resto de dispositivos (minicámara, proyector y sistema auxiliar), entre otros.

La solución a desarrollar se ha llevado a cabo mediante una división del sistema en **cinco subsistemas** cuyas tareas son completamente diferenciables:

- **Subsistema de representación gráfica:** La representación visual o gráfica de la información debe ser tratada de forma adecuada. Para ello, este subsistema se encarga de dicha tarea empleando las tecnologías y herramientas necesarias para el correcto despliegue.
- **Subsistema de delegación de tareas:** Debido a la elevada carga que supone para el sistema reconocer el documento actual mediante técnicas de visión por computador, se proporciona al proyecto GrayAR soporte por red, que derive su ejecución en un sistema auxiliar y devuelva los resultados a la unidad principal para su posterior procesado.
- **Subsistema de comunicación:** El subsistema anterior solo podría ser llevado a cabo estableciendo un contrato entre la unidad de despliegue y el sistema auxiliar. Dicho contrato debe mantener una relación equivalente entre las tareas que pueden ser ejecutadas de forma auxiliar y los resultados obtenidos de estas.
- **Subsistema de *scripts*:** De cara a la usabilidad del sistema por parte del usuario final, se proporciona un sencillo lenguaje de *script* que le permita definir secuencias de eventos asociadas a documentos específicos.
- **Subsistema de audio:** Cubriendo los casos de usos relacionados con los usuarios que padezcan problemas visuales, este subsistema trata de ofrecer retroalimentación sonora y descriptiva mediante texto a voz (TTS de su término en Inglés Text-To-Speech) y archivos de sonido.

La Figura 1.3 muestra además una visión general de los subsistemas y las relaciones existentes entre ellos.

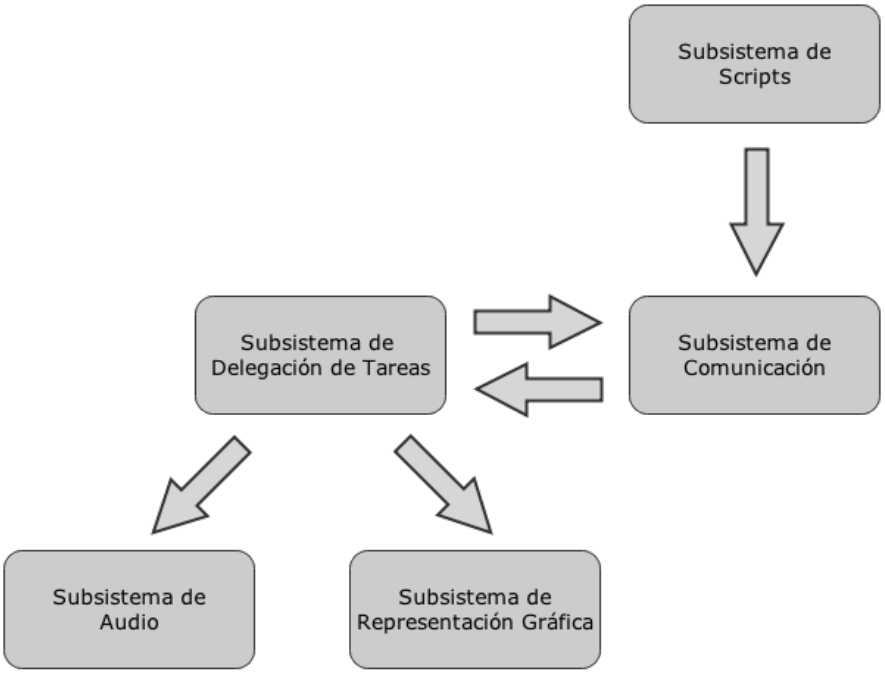

Figura 1.3: Subsistemas de BelfegAR.

1.6 Estructura del documento

El presente documento se ha estructurado según las indicaciones de la normativa de trabajos de fin de grado de la Escuela Superior de Informática de la Universidad de Castilla-La Mancha, empleando los siguientes capítulos:

Capítulo 2: Objetivos

En este capítulo se desglosa y se describen la lista de objetivos y subobjetivos planteados para este TFG.

Capítulo 4: Antecedentes

En este capítulo se hace un repaso a los conocimientos y áreas que han sido necesarias estudiar para el desarrollo de BelfegAR. En definitiva, ofrece un *estado del arte* de las tecnologías ya existentes y que abarcan los distintos subsistemas de este proyecto.

Capítulo 5: Método de trabajo

En este capítulo se explica y se justifica la metodología escogida para el desarrollo de este sistema. Además se describen los recursos empleados, tanto hardware como software.

Capítulo 6: Resultados

En este capítulo se enumeran y explican los resultados que se han obtenido del desarrollo del presente sistema.

Capítulo 7: Conclusiones

En este capítulo se hace un resumen a modo de conclusión del desarrollo y las metas

alcanzadas. También se enumera una serie de líneas de trabajo futuro planteadas para continuar su desarrollo, y una conclusión personal.

Capítulo 2

Objetivos

COMO se comentó anteriormente, el presente TFG forma parte de un proyecto de investigación en el ámbito de la Cátedra Indra-UCLM llamado ARgos, cuyo principal objetivo resulta en el tratamiento de documentos físicos de diversa índole mediante técnicas de Visión por Computador y Realidad Aumentada.

Un caso particular de dichos documentos son aquellos burocráticos empleados en multitud de instituciones públicas y empresas. La administración ha intentado informatizar parte de dichos documentos a lo largo de los años con cierto éxito. Algunos ejemplos de esto son el uso de la plataforma SARA [dAPGdE08], para realizar transferencias de forma segura entre las distintas administraciones públicas; la implantación del DNI electrónico [1], permitiendo que cada ciudadano posea su propio certificado digital; la red 060 [2], la cual ofrece un punto de acceso común a los ciudadanos para acceder a las diferentes sedes electrónicas, entre otros.

Sin embargo, y pese a los constantes esfuerzos por automatizar el tratamiento de la información, aún queda mucho por hacer, y durante este proceso, todavía se necesita lidiar con documentos físicos; ARgos supone una transición sencilla, que aúna documentos físicos con posibilidades digitales.

2.1 Objetivo general

El objetivo principal del proyecto ARgos es la construcción de un sistema de ayuda a la gestión de documentos mediante el uso de técnicas de visión por computador y síntesis visual y auditiva en el espacio físico, empleando técnicas de realidad aumentada.

Dentro de ARgos, el objetivo principal de BelfegAR puede definirse como «*la construcción de una plataforma para facilitar el despliegue de gráficos 3D en dispositivos empotrados de bajo coste para el soporte de gestión documental con realidad aumentada, así como el soporte para la delegación de tareas complejas en computadores de altas prestaciones*».

[1] http://www.dnielectronico.es/
[2] http://www.060.es/

2.2 Objetivos específicos

A partir del objetivo principal descrito en la sección anterior se determinan los subobjetivos de los apartados siguientes.

2.2.1 Desarrollo de componentes gráficos

En BelfegAR se desarrollará un *framework* que de soporte al despliegue de diferentes componentes gráficos.

El sistema deberá ser capaz de:

- Utilizar fuentes *TrueType* para la representación de textos.
- Emplear componentes comprensibles para el usuario.
- Proporcionar un abanico variado de componentes gráficos (cajas, botones, etc.).
- Construir componentes gráficos más complejos a partir de otros más simples (definición jerárquica).
- Ofrecer componentes gráficos modulares y fácilmente escalables.
- Soportar el despliegue de vídeo. En concreto este componente deberá instanciarse para el desarrollo de un módulo de videollamadas en tiempo real.

2.2.2 Delegación de tareas y comunicación

Se implementará una arquitectura cliente-servidor que permita ejecutar tareas en un computador con mayores capacidades de procesamiento. Dicha arquitectura será compartida entre los subsistemas de delegación de tareas y comunicación.

Dichos subsistemas deberán:

- Enviar por red la información crítica para su procesado en el servidor.
- Recibir los resultados del servidor para su interpretación en el cliente.
- Mantener la comunicación durante una videollamada.
- Mantener la comunicación entre el cliente y el servidor.

2.2.3 Reproducción de audio

Se implementará un gestor de audio que permita la carga y decodificación de archivos de audio en diversos formatos.

Este gestor deberá ser capaz de:

- Cargar archivos de audio.
- Soportar flujos de audio en lugar de la carga total del archivo.
- Soportar los formatos multimedia más populares.

- Mantener una lista de archivos de audio para su posterior reproducción.
- Autocargar un archivo de audio cuya reproducción haya sido solicitada y no se encuentre actualmente en memoria.
- Proporcionar métodos generales de control de audio como reproducir, pausar o detener.
- Permitir controlar el volumen tanto de los sonidos individuales como del sistema completo.

2.2.4 Sistema programable de forma externa

Se desarrollará un lenguaje de *scripting* que permita la programación de algunas partes del sistema, mediante el subsistema de *scripts*.

Dicho subsistema deberá cumplir con los siguientes requisitos:

- Mantener una sintaxis lo más clara y simple posible para el usuario.
- Definir un conjunto de funciones predefinidas a realizar en los *scripts*.
- Mantener la carga de los *scripts* en el servidor del sistema.
- Facilitar al usuario la creación de nuevos *scripts*.
- Mantener el subsistema lo suficientemente modular como para poder añadir nuevas funciones sin afectar al resto de módulos.

2.2.5 Estabilidad del sistema y recuperación ante errores

Tanto el cliente como el servidor deben de poder recuperarse de errores.

De esta forma el sistema deberá:

- Recuperarse ante errores no críticos.
- Mantener un registro con los eventos del cliente y del servidor.
- Reconectar el cliente en caso de que se pierda la conexión con el servidor.

2.2.6 Orientación hacia usuarios con necesidades especiales

Dada la naturaleza de su proyecto padre, ARgos, y al ser la rama principal que ofrezca retroalimentación al usuario, BelfegAR debe de tener en cuenta todas las necesidades de sus usuarios.

De esta forma, el sistema deberá:

- Tener en cuenta las necesidades de sus usuarios y proporcionar las funciones necesarias para cumplirlas.
- Facilitar el soporte del usuario.
- Facilitar el desempeño del usuario con el sistema.

2.2.7 Bases, principios y filosofía

De acuerdo al objetivo principal, el sistema debe sostenerse sobre ciertas bases priorizando sobre todo en componentes de bajo coste. Además, el proyecto será llevado a cabo utilizando software libre, tanto para la creación de los recursos necesarios como para la implementación empleando distintas bibliotecas.

Por lo anterior, el sistema deberá:

- Implementarse mediante buenas prácticas de programación.
- Implementarse atendiendo a las optimizaciones necesarias.
- Basarse en tecnologías que sirvan para solventar algunas partes de la implementación.
- Implementarse empleando herramientas con licencias libres.
- Desarrollarse siguiendo los principios Keep It Simple, Stupid (KISS).
- Utilizar bibliotecas de código abierto y con licencias libres para la implementación.
- Emplear bibliotecas multiplataforma que faciliten la transición a otros sistemas y arquitecturas.

2.3 Limitaciones físicas

En los apartados anteriores se han nombrado algunas de las limitaciones más importantes, como es el reducido rendimiento de la unidad de detección y despliegue.

La unidad de procesamiento elegida para el desarrollo del proyecto no dispone de un altavoz integrado, por lo que requerirá la instalación de uno propio para ser usado por el subsistema de audio.

Otra limitación física que atiende el sistema trata sobre las videollamadas. La unidad de procesamiento no cuenta con entrada de micrófono; solo posee una salida de audio. Esto supondrá un problema a la hora de enviar audio a través de las videollamadas.

En el ámbito de red, el sistema requiere una conexión cableada con la red del servidor para funcionar correctamente.

2.4 Limitaciones de cómputo

Pese a que la carga del sistema es delegada en un servidor con mayores capacidades, hay que tener en cuenta las cualidades de la conexión de red entre el cliente y el servidor. Dicha conexión puede plantear un *cuello de botella* insalvable si no se es consciente de la información que se debe enviar y recibir por ella.

Será necesario por tanto prestar especial atención a las limitaciones relativas a la unidad de procesamiento y a los diversos cuellos de botella que puedan surgir (conexiones de red, despliegue gráfico, tiempo de procesamiento, etc.).

Capítulo 3

Objectives

As it was stated before, the current Final Degree Project belongs to a research project in the field of the Cátedra Indra-UCLM named ARgos, which main objective results in treating various kinds of physical documents through Computer Vision and Augmented Reality techniques.

A particular case of these documents are those used in many bureaucratic public institutions and businesses. The administration has attempted to computerize part of such documents over the years with some success. Some examples are the use of SARA [dAPGdE08] platform for secure transfers between different public administrations; the introduction of electronic ID [1] allowing each citizen possesses its own digital certificate; 060 network [2], which provides a common point of access for citizens to access the different virtual offices, among others.

However, in spite of the constant efforts to automatize the treatment of information, there is still too much to do, and during this process, it already needed to deal with physical documents; ARgos supposes an easy transition which joins physical documents with digital possibilities.

3.1 General objective

The main objective of the ARgos project is to build a helping system for document management through computer vision techniques, and visual and auditive synthesis in the physical space, using augmented reality techniques.

Within ARgos, the main aim of BelfegAR can be defined as «*the construction of a platform to facilitate the deployment of 3D graphics in low-cost embedded devices to support document management with augmented reality, as well as to support delegation of complex tasks to high performance computers*».

[1] http://www.dnielectronico.es/
[2] http://www.060.es/

3.2 Specific objectives

From the main objective described in the previous section the subgoals of the following sections are determined.

3.2.1 Graphic components development

In BelfegAR it will be developed a framework to support the deployment of different graphic components.

The system will be capable of:

- Using *TrueType* fonts to display texts.
- Employing comprehensive components for the user.
- Providing a diverse range of graphical components (boxes, buttons, etc.).
- Building graphic components more complex from simpler ones (hierarchical definition).
- Offering modular and easily scalable graphic components.
- Supporting video deployment. Specifically, this component will have to be instantiated for the development of a real time videoconference module.

3.2.2 Tasks delegation and communication

It will be implemented a client-server architecture to execute tasks remotely in a computer with high computing capabilities. This architecture will be shared between the tasks delegation and communication subsystems.

These subsystems will:

- Send critical information through the network to be processed in the server.
- Receive the results from the server to be interpreted in the client.
- Keep the communication during a videoconference.
- Keep the communication between the client and the server.

3.2.3 Audio playback

It will be implemented an audio manager that allows the loading and decoding of audio files in different formats.

This manager will be able to:

- Load audio files.
- Support audio streams instead of loading the whole file.
- Support the most popular multimedia formats.

- Keep a list of audio files for later playback.
- Autoload an audio file which playback has been requested and it is not found in memory yet.
- Provide general methods of audio control like playing, pausing or stopping.
- Allow volume control both single audios and the entire system.

3.2.4 External programmable system

It will be developed a scripting language that allows programming some parts of the system, through the scripts subsystem.

This subsystem will have to achieve the next requirements:

- To keep a clear and simple syntax as much as possible for the user.
- To define a set of predefined functions to call them within the scripts.
- To keep the loading of the scripts in the system server.
- To facilitate the making of new scripts to the user.
- To keep the subsystem sufficiently modular to be able to add new functions without affecting other modules.

3.2.5 System stability and error recovery

Both of the client and the server have to be able to recover themselves from errors.

This way the system has to:

- Recover itself from non-critical errors.
- Keep an event log from both the client and server.
- Reconnect the client in case of losing connection with the server.

3.2.6 Orientation towards users with special needs

Given the nature of its father project, ARgos, and being the main branch offering feedback to the user, BelfegAR has to take into account all the needs of its users.

This way, the system will have to:

- Take into account the needs of its users and provide the functions needed to accomplish them.
- Facilitate the user support.
- Facilitate the user performance with the system.

3.2.7 Bases, principles and philosophy

According to the main objective, the system must hold onto certain basis prioritizing above all in low cost components. Moreover, the project will be carried out using free software, both for making the needed resources and for implementing the project using different libraries.

Because of that, the system will have to:

- Implement itself through good programming practices.
- Implement itself attending to the needed optimizations.
- Build itself on technologies useful to solve some parts of the implementation.
- Implement itself using free software tools.
- Develop itself following the Keep It Simple, Stupid (KISS) principles.
- Use free and open source libraries to implement it.
- Use crossplatform libraries which facilitate the transition to other systems and architectures.

3.3 Physical limitations

In the previous sections it have been named some of the most important limitations, such as low efficiency of the detection and deployment unit.

The processing unit chosen for the project does not have a built-in speaker, so it will need to have one's own for use by the audio subsystem.

Another physical limitation of the system is about videoconferences. The processing unit has no microphone input; it only has one audio output. This will be a problem when sending audio through videoconferences.

In the field of network, the system requires a wired connection to the network server to run correctly.

3.4 Computing limitations

Although the system load is delegated to a server with higher capacities, it has to take into account the qualities of the network connection between the client and the server. This connection may present an insurmountable bottleneck if you are not aware of the information to be sent and received by it.

Therefore, it is necessary to pay special attention to limitations of the processing unit and the various bottlenecks that may arise (network connections, graphical display, processing time, etc.).

Capítulo 4

Antecedentes

Los cinco grupos de requisitos funcionales que merecen especial mención dentro del diseño y desarrollo de BelfegAR son el **despliegue gráfico**, la **delegación de tareas**, la **reproducción de audio**, su **sistema de *scripts*** y **la captura de imágenes para videollamada**. Para realizar una implementación satisfactoria de cada uno de ellos se ha realizado una investigación previa para cada tema. Este estudio del estado actual de cada uno de estos aspectos tecnológicos viene acompañado en el documento de una conclusión final que resume las decisiones finalmente adoptadas en la implementación del sistema.

El estudio de cada una de las áreas anteriores se realizará en base a subáreas que nos facilitarán realizar un estudio pormenorizado de los aspectos más relevantes a tener en cuenta. Para la parte de **despliegue gráfico** se analizarán las distintas bibliotecas, APIs y *frameworks* disponibles, siempre teniendo en cuenta los objetivos del proyecto. También se introducirá parte de los conocimientos matemáticos necesarios para aplicar transformaciones sobre el espacio 3D y otra información teórica necesaria para comprender la totalidad del ámbito gráfico. La **delegación de tareas**, por su parte, realizará una introducción de los conceptos adoptados cuando se habla de una arquitectura de red, y se revisará además otros proyectos previos que necesitaron implementar una arquitectura de red similar a la necesaria por BelfegAR. En lo referente a la **reproducción de audio** se revisarán las distintas bibliotecas de audio existentes y que cumplan los requisitos establecidos. El **sistema de *scripts***, introducido por primera vez en el capítulo anterior, será estudiado en este capítulo intentando buscar la solución más simple para nuestra implementación. Finalmente, el tema de **videollamadas** será tratado con especial atención desde el punto de vista del rendimiento, por ser uno de los módulos que más recursos requiere en la plataforma final de despliegue.

La Figura 4.1 contiene una visión general de los conceptos tratados en este capítulo. Como añadido, en el apartado de **diseño software** se estudiarán los fundamentos necesarios para ofrecer una solución multiplataforma del sistema. También se estudiarán algunas **herramientas y bibliotecas externas** que serán utilizadas para la implementación de ciertas partes que no tienen cabida en otros apartados, como la carga de fuentes de texto desde archivos o las herramientas necesarias para modelar algunas aplicaciones externas, como un prototipo diseñado en 3D de todo el entorno que ofrezca una visión general del sistema.

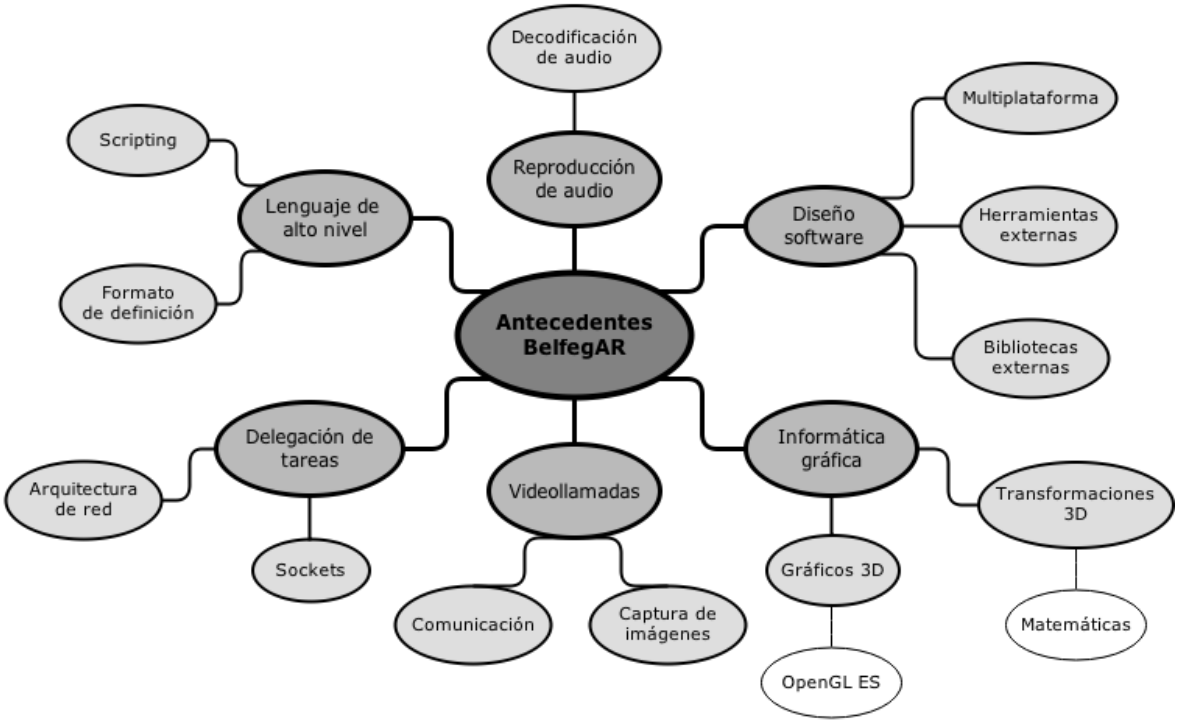

Figura 4.1: Mapa conceptual del capítulo.

4.1 Despliegue gráfico

En esta sección se revisarán las áreas abarcadas por la informática gráfica que involucra gráficos por computador necesarios para el total funcionamiento de BelfegAR.

Las áreas que trataremos son aquellas que hemos visto anteriormente en la Figura 4.1:

- Estado actual de bibliotecas gráficas
- Gráficos por computador y OpenGL ES 2.0
- Transformaciones 3D empleando matemáticas

4.1.1 Estado actual de bibliotecas gráficas

Aunque el número de bibliotecas gráficas existentes es bastante grande, hay que tener en cuenta las limitaciones de la unidad de detección y despliegue. Necesitamos conocer de antemano qué interfaz emplea la unidad para comunicarse con la GPU, para de esta forma conseguir aceleración gráfica por hardware y mejores resultados. En caso de que el procesado gráfico se realizara por software, obtendríamos un rendimiento muy reducido que haría imposible el desarrollo del proyecto.

Como dijimos en capítulos anteriores, la unidad de detección y despliegue es un minicomputador «Raspberry Pi». En el Anexo A, si nos fijamos en la fila *GPU* de la tabla de características, vemos que el dispositivo soporta gráficos acelerados por hardware mediante OpenGL ES 2.0.

...	...
GPU:	Broadcom VideoCore IV, **OpenGL ES 2.0**, [...]
...	...

Conocido esto y debido al bajo rendimiento de la unidad, solo podíamos utilizar bibliotecas que operaran sobre OpenGL ES 2.0 y que abstrajeran el proceso de desarrollo, o realizar la implementación directamente sobre OpenGL ES 2.0.

A fecha de la implementación no había ninguna biblioteca conocida que operara sobre OpenGL ES 2.0. Aun así se realizaron algunas pruebas de rendimiento sobre la biblioteca Simple DirectMedia Layer (SDL), por ser lo suficientemente madura y haber sido empleada en la industria en numerosas ocasiones, para comprobar como se desempeñaba.

SDL

Se trata de una biblioteca de desarrollo multiplataforma diseñada para proporcionar un acceso de bajo nivel a sistemas de sonido, teclado, ratón, joystick y hardware gráfico a través de OpenGL y Direct3D [SDLb].

SDL puede ser extendida empleando módulos externos que añaden funcionalidades de red, mezclador de audio y carga de diferentes formatos de imagen, entre otros.

El principal problema de la versión utilizada de SDL (1.2), a fecha de este documento, era que no soportaba aceleración gráfica, ya que no permitía la interacción con la versión empotrada de OpenGL incluida en la Raspberry Pi.

Aun así, se realizaron algunas pruebas de rendimiento sobre SDL, las cuales se detallarán en el Capítulo 6. El reducido rendimiento se hizo visible al aplicar dichas transformaciones en tiempo real, ya que se estaba trabajando sobre mapas de bits y no sobre polígonos 3D, lo que resultaba en algoritmos más complejos a la hora de tratar con los primeros.

La nueva versión de SDL (2.0) sí que permitía el uso de OpenGL ES 2.0 de forma nativa [SDLa], sin embargo no se encontraba totalmente implementada para la plataforma, presentando de esta forma algunos problemas de funcionamiento en los sistemas de sonido y eventos. Debido a esto y a los posibles futuros problemas que pudiera presentar una vez el sistema se encontrara implementado se decidió realizar una apuesta más segura descartando su uso.

Aunque descartada, volveremos a ver SDL en este mismo capítulo usándola en otro ámbito, gracias a las numerosas posibilidades que ofrece.

OpenGL ES 2.0

OpenGL for **E**mbedded **S**ystems es un subconjunto de la API de renderizado gráfico 2D y 3D por computador, OpenGL, normalmente acelerado por hardware usando el procesador

gráfico (GPU). Está diseñada principalmente para sistemas como smartphones, tablets, video consolas y PDAs.

La principal diferencia entre OpenGL ES y su análogo de escritorio es la ausencia de un *pipeline* de renderizado fijo, convirtiéndolo en programable y forzando al desarrollador a implementar sus propios *shaders*.

Los *shaders* son programas informáticos que se ejecutan sobre la GPU, empleados para *sombrear*, es decir, producir los niveles de color apropiados en una imagen, producir efectos especiales o realizar post-procesado de vídeo; se trata de programas que describen las características de un vértice o de un píxel. Originalmente existían dos tipos de *shaders*: *vertex shaders* y *fragment (o pixel) shaders*, aunque recientemente se han introducido nuevos tipos denominados *geometry shader* y *tessellation shaders*. A efectos prácticos, OpenGL ES 2.0 solo soporta los dos primeros, por lo cual son los que se describen a continuación:

- **Vertex shader [Ver13]:** Describe las características relacionadas con la posición, coordenadas de textura, color, ... de un vértice. Su propósito es transformar las posiciones 3D de cada vértice en coordenadas del espacio 2D, tal y como aparecerían en la pantalla. Se ejecuta una vez por cada vértice.
- **Fragment shader [Fra13]:** Describen las características relacionadas con el color, la profundidad y la transparencia de un píxel. Para cada grupo de píxeles cubiertos por una primitiva se genera un nuevo *fragmento*, el cual contiene todos los valores de salida de los vértices anteriores interpolados. Al igual que el *vertex shader*, se ejecuta una vez por cada fragmento existente.

OpenGL define un conjunto de funciones, las cuales suelen seguir un convenio de nombrado para informar al programador sobre su funcionamiento. Por ejemplo, la función para especificar el valor de una variable de tipo *uniform* para el programa *shader* actual:

```
void glUniform3i(GLint location, GLint v0, GLint v1, GLint v2);
```

- Comienza con el prefijo `gl`.
- Se llama `Uniform`.
- Admite como parámetros manipuladores para la variable *uniform* indicada 3 enteros (`3i`).

OpenGL funciona como una máquina de estados; mientras que no se cambie el estado todas las funciones que sean invocadas operarán sobre el estado actual en que se encuentre OpenGL.

Como es de suponer, finalmente se optó por emplear OpenGL ES 2.0 para el desarrollo del sistema por la facilidad y rendimiento producido a la hora de tratar con polígonos directamente en lugar de mapas de bits.

4.1.2 Gráficos por computador y OpenGL ES 2.0

En esta sección se introducirán los conceptos principales empleados en computación gráfica. Pese a haber utilizado un reducido grupo de técnicas gráficas en el desarrollo de BelfegAR, se procederá a ofrecer una visión general del funcionamiento de una aplicación que utilice gráficos 3D.

Pipeline de renderizado gráfico de OpenGL ES 2.0

El *pipeline* o cauce de renderizado, es la secuencia de pasos que realiza OpenGL cuando renderiza escenas tridimensionales [MGS08]. Como ventaja, dicho cauce permite la paralelización de etapas de diferentes iteraciones, incrementando de este modo el rendimiento.

Entre las diferentes etapas que se explicarán a continuación sobre el cauce de renderizado se ofrece en la Figura 4.2 una visión general. Se han omitido algunas de las etapas por no ser consideradas relevantes para la completa compresión del cauce.

Las cajas sombreadas indican las etapas del cauce que son programables.

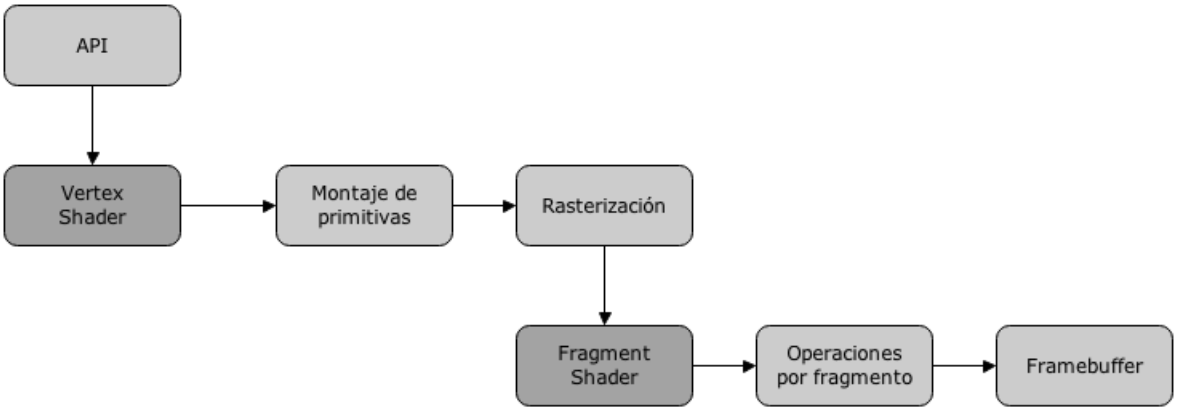

Figura 4.2: *Pipeline* gráfico de OpenGL ES 2.0 (simplificado).

API

Se trata del punto de entrada al cauce de OpenGL ES 2.0. El programador tiene control total para definir las listas de vértices que serán enviadas al cauce para su posterior procesado. Como resultado, esta etapa genera la información geométrica que se proporcionará a la siguiente etapa, tales como puntos, líneas o triángulos.

Vertex Shader

Esta etapa implementa un método programable de propósito general para operar sobre los vértices. La información de entrada de esta etapa puede ser [1]:

- **Attributes**: Información para cada vértice suministrada a través de listas de vértices.

[1] No se traducen los nombres del inglés por mantener la consistencia más adelante con el lenguaje de definición de *shaders*

- **Uniforms**: Información constante usada por esta etapa.
- **Samplers**: Un tipo concreto de *uniform* que representa texturas usadas en esta etapa; opcionales, sin embargo.
- **Shader program**: El código fuente o binario de un programa de esta etapa que describa las operaciones que se realizarán a nivel de vértice.

La salida de esta etapa resulta en las denominadas variables variantes o *varying variables* en inglés. Estas variables son calculadas para cada fragmento existente en la etapa de rasterización mediante interpolación y pasadas como entradas a la etapa de *fragment shader*.

Esta etapa puede ser programada para atender los requisitos del desarrollador, tales como transformar las posiciones de los vértices, generar color a nivel de vértice o generar/transformar coordenadas de textura.

El Listado 4.1 muestra un sencillo ejemplo de programa *vertex shader*, que lo único que hace es calcular las nuevas posiciones de los vértices y sus colores a partir de las entradas proporcionadas al programa (*attributes*) y genera un nuevo color para dicho vértice que se pasará a la etapa de *fragment shader* (*varying*). Dicho ejemplo será continuado con su segunda parte en la etapa de *fragment shader*.

```
1  attribute vec4 a_position;
2  attribute vec4 a_color;
3
4  varying vec4 v_color;
5
6  void main() {
7    v_color = a_color;
8    gl_Position = a_position;
9  }
```

Listado 4.1: Ejemplo de programa *Vertex Shader*.

Montaje de primitivas

En esta etapa los vértices ya sombreados (por la etapa anterior) son montados en primitivas geométricas individuales que pueden ser dibujadas como triángulos o líneas. Las primitivas son objetos geométricos que pueden ser dibujados empleando para ello comandos concretos de OpenGL ES, los cuales especifican un conjunto de atributos para cada vértice, tales como su posición o color.

Para cada primitiva se comprueba si entra dentro del *frustum* de visión o no. El *frustum* es la región del espacio 3D que es visible en la pantalla (ver Figura 4.3). Si la primitiva no entra dentro del *frustum*, esta es recortada hasta que se adapte. Este último proceso se conoce como recorte o *clipping*. Si la primitiva está completamente fuera del *frustum* se descarta directamente. Después de la fase de *clipping* las posiciones de

los vértices son convertidas a coordenadas de pantalla. Opcionalmente otras primitivas pueden ser descartadas en función de si están mirando hacia la cámara (no se descarta) o no (se descarta); este proceso se conoce como *culling*. Finalmente, la primitiva está preparada para ser pasada a la etapa de rasterización.

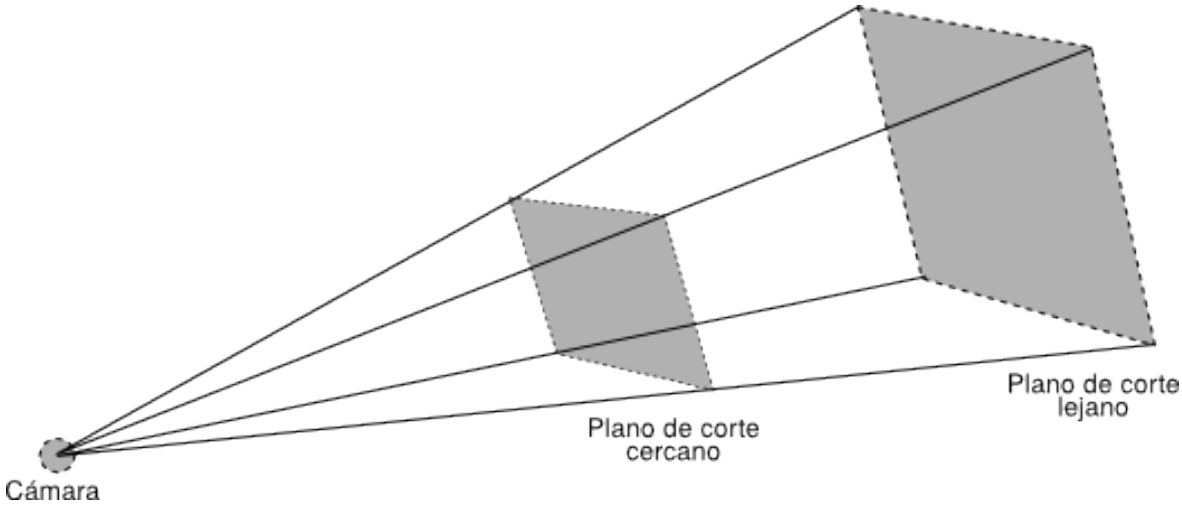

Figura 4.3: Representación gráfica del *frustum* de visión.

Rasterización

En esta etapa la primitiva es dibujada. Rasterización es el proceso que convierte las primitivas en un conjunto de fragmentos 2D, procesados posteriormente en la etapa de *fragment shader*. Dichos fragmentos son simplemente píxeles que serán dibujados en pantalla.

Fragment Shader

La siguiente etapa programable del cauce es el *fragment shader*. A diferencia del *vertex shader* que opera sobre los vértices de la primitiva, el *fragment shader* opera directamente sobre los fragmentos producidos por la etapa anterior.

La información de entrada de esta etapa puede ser:

- **Varying variables**: La salida del *vertex shader* generada durante la etapa de rasterización para cada fragmento mediante interpolación.
- **Uniforms**: Información constante usada por esta etapa.
- **Samplers**: Un tipo concreto de *uniform* que representa texturas usadas en esta etapa.
- **Shader program**: El código fuente o binario de un programa de esta etapa que describa las operaciones que se realizarán a nivel de fragmento.

El Listado 4.2 muestra la segunda parte del programa *shader* que empezamos en el *vertex shader* (ver Listado 4.1). Ahora, mediante la *varying variable* se recibe como

entrada el color que se asignó a cada vértice en la etapa de *vertex shader* y a partir de este se interpolan los colores para todos los fragmentos definidos entre los vértices.

```
varying vec4 v_color;

void main() {
  gl_FragColor = v_color;
}
```

Listado 4.2: Ejemplo de programa *Fragment Shader*.

Operaciones por fragmento

En esta etapa se aplican las distintas operaciones a los fragmentos producidos en la etapa anterior antes de ser enviados al *framebuffer* para su visualización o post-procesado.

La Figura 4.4 muestra la secuencia de operaciones que se llevan a cabo para cada fragmento obtenido.

También se explican a continuación las diferentes operaciones en detalle:

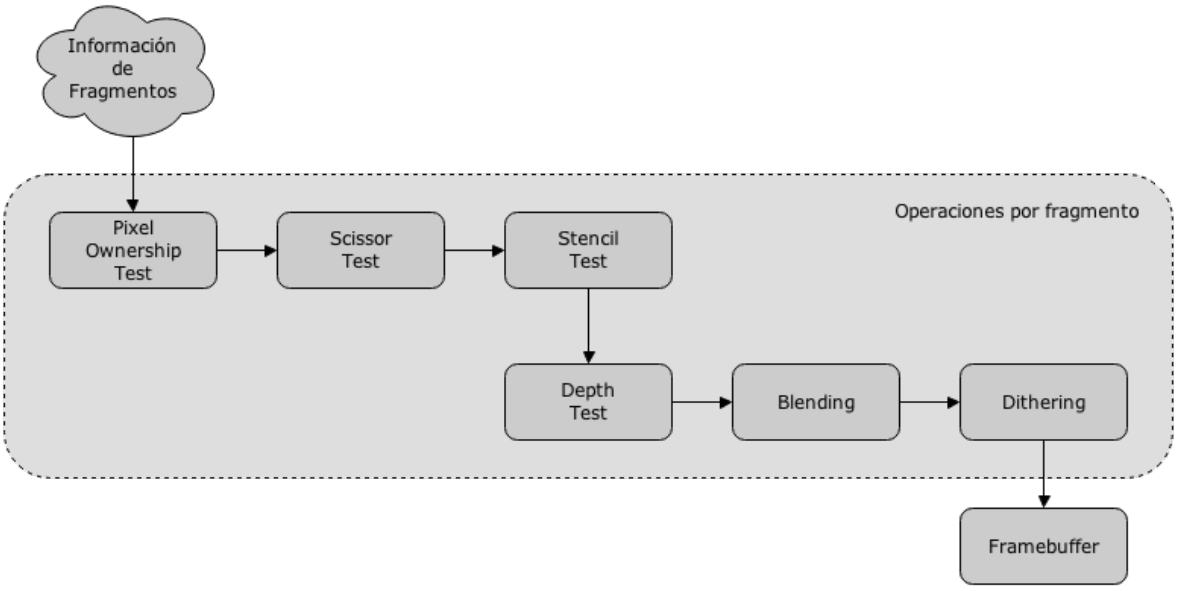

Figura 4.4: Operaciones por fragmento en OpenGL ES 2.0.

- **Pixel ownership test:** Esta operación determina si cierto píxel en la posición (x,y), pertenece o no al contexto de OpenGL ES. Gracias a esta operación el sistema de ventanas puede saber qué píxeles del *framebuffer* pertenecen al contexto de OpenGL ES actual.
- **Scissor test:** Esta operación determina si la posición (x,y) del fragmento cae dentro del rectángulo definido como parte del estado de OpenGL ES por el programador. Los fragmentos que caigan fuera de esa región serán descartados.

- **Stencil y depth tests:** Estas operaciones realizan pruebas sobre el valor *stencil* y de profundidad del fragmento entrante para determinar si éste debería de ser rechazado o no.
- **Blending:** Se encarga de realizar un fundido entre el color del fragmento con el color almacenado en el *framebuffer* sobre la posición (x,y).
- **Dithering:** Se trata de una operación que puede ser usada para minimizar los *artefactos* (errores visuales) que ocurren al utilizar una precisión limitada a la hora de almacenar los valores de color en el *framebuffer*.

Framebuffer

Finalmente, y después de todas las etapas anteriores, la información es almacenada en el *framebuffer*. Un *framebuffer* es simplemente un área en la memoria sobre la que se puede renderizar. OpenGL ES permite la creación de distintos *framebuffers* que pueden ser usados como escenas compuestas de multitud de gráficos para ser tratados conjuntamente. Además, también define un *framebuffer* inicial o 0 que corresponde a la salida de vídeo de la GPU, normalmente el monitor.

Siguiendo los programas de *shaders* anteriores y suponiendo la entrada a la API de 3 vértices formando un triángulo con los colores rojo, verde y azul, uno para cada vértice, una posible aproximación del *framebuffer* 0 después de pasar por todas las etapas sería la visible en la Figura 4.5.

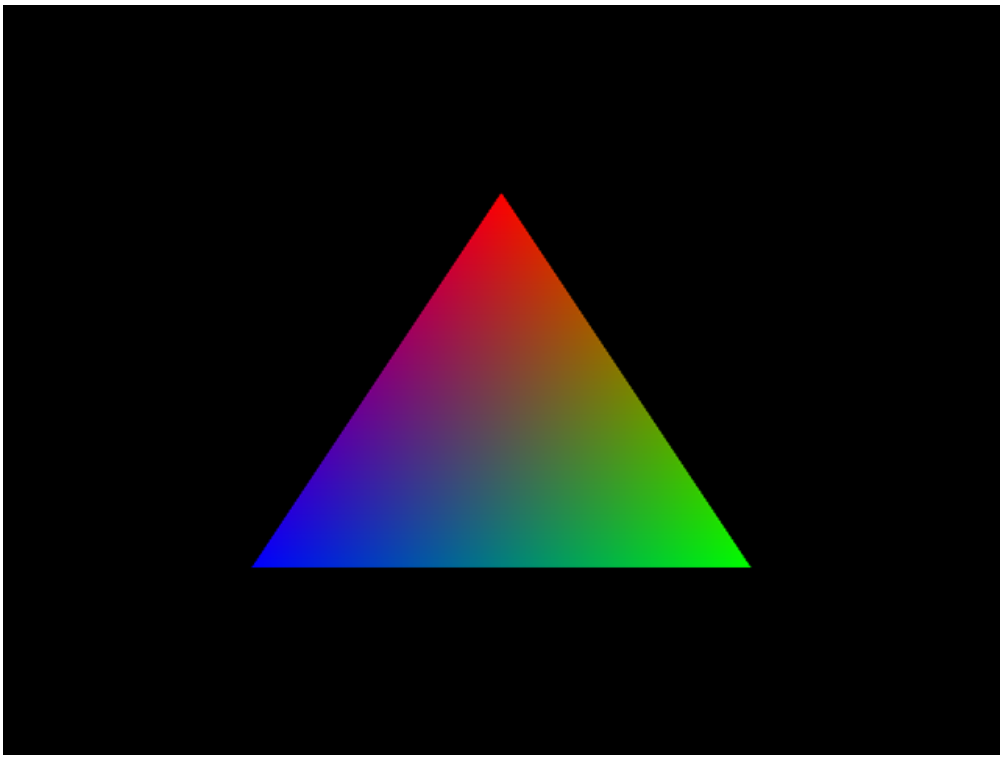

Figura 4.5: Posible resultado final después de recorrer todas las etapas del cauce gráfico.

4.1.3 Transformaciones 3D empleando matemáticas

La informática gráfica se encuentra íntimamente ligada a las matemáticas. Las transformaciones necesarias para escalar, rotar y trasladar una figura geométrica son aplicadas mediante operaciones matemáticas basadas en geometría espacial del espacio euclídeo tridimensional $\mathbb{R}^3$.

En esta sección se introducirán las bases matemáticas y las transformaciones necesarias para operar sobre el espacio 3D de forma correcta; una necesidad fundamental de BelfegAR. Además, se planteará el uso de una biblioteca externa que facilite dichas operaciones matemáticas.

Vectores

Los vectores son magnitudes físicas definidas por un punto de referencia, un módulo (o longitud), una dirección y un sentido [EMVH05].

Entre las operaciones disponibles para trabajar con vectores se encuentran:

Suma de vectores

Sean $\vec{u}$ y $\vec{v}$ vectores definidos en $\mathbb{R}^3$, la suma $\vec{u} + \vec{v}$ viene definida por la suma de sus componentes:

$$\begin{aligned} \vec{u} &= (u_1, u_2, u_3) \\ \vec{v} &= (v_1, v_2, v_3) \\ \vec{u} + \vec{v} &= (u_1 + v_1, u_2 + v_2, u_3 + v_3) \end{aligned} \tag{4.1.3.1}$$

Producto de un vector por un escalar

Sea $\vec{u}$ un vector definido en $\mathbb{R}^3$ y k un número real, el producto de $\vec{u}$ por k viene definida como la multiplicación de k por cada una de las componentes de $\vec{u}$ como $k\vec{u}$:

$$\begin{aligned} \vec{u} &= (u_1, u_2, u_3) \\ k\vec{u} &= (ku_1, ku_2, ku_3) \end{aligned} \tag{4.1.3.2}$$

Producto escalar

Sean $\vec{u}$ y $\vec{v}$ vectores definidos en $\mathbb{R}^3$, el producto escalar $\vec{u} \cdot \vec{v}$ viene definido como la suma de los productos de las componentes de ambos vectores:

$$\begin{aligned} \vec{u} &= (u_1, u_2, u_3) \\ \vec{v} &= (v_1, v_2, v_3) \\ \vec{u} \cdot \vec{v} &= u_1v_1 + u_2v_2 + u_3v_3 \end{aligned} \tag{4.1.3.3}$$

También se define el producto escalar como:

$$\vec{u} \cdot \vec{v} = |\vec{u}||\vec{v}| \cos\theta \tag{4.1.3.4}$$

Propiedad: Si el producto escalar de dos vectores es cero ($\vec{u} \cdot \vec{v} = 0$), ambos vectores son perpendiculares entre sí.

Módulo de un vector

Sea $\vec{u}$ un vector definido en $\mathbb{R}^3$, su módulo o longitud viene definido como la raíz cuadrada del producto escalar $\vec{u} \cdot \vec{u}$:

$$|\vec{u}| = \sqrt{\vec{u} \cdot \vec{u}} = \sqrt{u_1^2 + u_2^2 + u_3^2} \tag{4.1.3.5}$$

Producto vectorial

Sean $\vec{u}$ y $\vec{v}$ vectores definidos en $\mathbb{R}^3$, el producto vectorial $\vec{u} \times \vec{v}$ viene definido como:

$$\vec{u} \times \vec{v} = (u_2v_3 - u_3v_2, u_3v_1 - u_1v_3, u_1v_2 - u_2v_1) \tag{4.1.3.6}$$

O empleando notación matricial calculando su determinante:

$$\vec{u} \times \vec{v} = \begin{bmatrix} i & j & k \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{bmatrix} \tag{4.1.3.7}$$

También se define el producto vectorial como:

$$\vec{u} \times \vec{v} = |\vec{u}||\vec{v}| \sin\theta \tag{4.1.3.8}$$

Propiedad: El resultado del producto vectorial entre dos vectores es un nuevo vector perpendicular a los dos primeros.

Matrices

Las matrices son empleadas en informática gráfica para aplicar transformaciones sobre el espacio 3D. Se definen simplemente como tablas bidimensionales de números, símbolos o expresiones, organizados en filas y columnas [Wik13].

Suma de matrices

Sean A y B dos matrices con igual número de filas y de columnas, la suma de matrices $A + B$ viene definida como:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix}$$

$$A + B = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \dots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \dots & a_{2n} + b_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \dots & a_{mn} + b_{mn} \end{bmatrix} \tag{4.1.3.9}$$

Producto de un escalar por una matriz

Sea A una matriz y k un escalar, el producto kA viene definido como:

$$kA = \begin{bmatrix} ka_{11} & ka_{12} & \dots & ka_{1n} \\ ka_{21} & ka_{22} & \dots & ka_{2n} \\ \dots & \dots & \dots & \dots \\ ka_{m1} & ka_{m2} & \dots & ka_{mn} \end{bmatrix} \tag{4.1.3.10}$$

Producto de matrices

Sean A y B dos matrices, su producto AB viene definido como:

$$\begin{aligned} AB &= \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & \dots & b_{1p} \\ \dots & \dots & \dots \\ b_{n1} & \dots & b_{np} \end{bmatrix} \\ &= \begin{bmatrix} a_{11}b_{11} + \ldots + a_{1n}b_{n1} & \dots & a_{11}b_{1p} + \ldots + a_{1n}b_{np} \\ \dots & \dots & \dots \\ a_{m1}b_{11} + \ldots + a_{mn}b_{n1} & \dots & a_{m1}b_{1p} + \ldots + a_{mn}b_{np} \end{bmatrix} \end{aligned} \tag{4.1.3.11}$$

Determinante de una matriz

Sea A una matriz cuadrada de orden 3, una forma sencilla de calcular su determinante $|A|$ viene dada empleando la regla de Sarrus:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{4.1.3.12}$$
$$= a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13} - a_{32}a_{23}a_{11} - a_{33}a_{21}a_{12}$$

Traspuesta de una matriz

Sea A una matriz cuadrada de orden 2, la traspuesta A^T viene definida como:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$
$$A^T = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix} \tag{4.1.3.13}$$

Inversa de una matriz

La inversa de una matriz puede ser calculada de varias formas, entre las más comunes se encuentran el método de *Gauss-Jordan* y a través del cálculo de la matriz de adjuntos. Usando este último método y partiendo de una matriz A, su inversa A^{-1} viene definida como:

$$A^{-1} = \frac{1}{|A|} Adj(A^T) \tag{4.1.3.14}$$

Transformaciones en OpenGL

OpenGL aplica las transformaciones a nivel de vértice empleando para ello matrices de transformación. Las matrices de transformación son matrices cuadradas de orden 4 que admiten tres tipos de transformaciones distintas. Dichas matrices pueden multiplicarse por un punto (o vértice) de un polígono para aplicarle la transformación que contienen. Las transformaciones además se pueden componer entre si de tal forma que una única matriz de transformación guarde información sobre los tres tipos de transformaciones existentes.

A continuación se detallan dichas transformaciones:

Traslación

Este tipo de transformaciones permiten desplazar el punto (o vértice) hacia una nueva posición. La matriz de traslación viene definida como:

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1.3.15}$$

T es la nueva posición que queremos que mantenga nuestra matriz de traslación.

Escala

Este tipo de transformaciones permiten escalar el punto (o vértice) hacia un nuevo tamaño. La matriz de escala viene definida como:

$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1.3.16}$$

S es la magnitud que queremos que mantenga nuestra matriz de escala.

Rotación

Este tipo de transformaciones permiten rotar el punto (o vértice) un ángulo determinado. La matriz de rotación viene definida como:

$$\text{Rotación respecto al eje X} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1.3.17}$$

$$\text{Rotación respecto al eje Y} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1.3.18}$$

$$\text{Rotación respecto al eje Z} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1.3.19}$$

θ es el nuevo ángulo que queremos que mantenga nuestra matriz de rotación.

Cabe destacar que las transformaciones anteriores actúan sobre el sistema de coordenadas global, es decir, toman en consideración el origen de coordenadas como el $(0, 0, 0)$ del mundo y no el local de la figura a transformar.

Dichas transformaciones, como hemos dicho antes, pueden multiplicarse entre si para conseguir una única matriz con varias transformaciones aplicadas; es lo que se denomina **matriz compuesta**. Sin embargo, hay que tener en cuenta que la composición de matrices es asociativa pero **no es conmutativa**, es decir, dependiendo de si primero trasladamos un vértice y después lo escalamos, y viceversa, podríamos obtener resultados totalmente distintos.

Las matrices de Modelo, Vista y Proyección

Las matrices de modelo, vista y proyección mantienen las transformaciones geométricas de los vértices, cámara y perspectiva respectivamente. La combinación de las tres matrices resulta en la matriz Model-View-Projection (MVP). El producto de esta matriz por cada vértice del modelo transforma dichos vértices con respecto a la información de las matrices. A continuación se incluye una explicación más detallada de las tres matrices:

Matriz de modelo

La matriz de modelo almacena las transformaciones geométricas que queremos aplicar a los vértices del objeto. Estas transformaciones pueden ser cualquiera de las vistas anteriormente: traslación, escalado y/o rotación. Como dijimos, estas transformaciones pueden multiplicarse entre sí para formar una única matriz; la matriz de modelo.

Siendo M la matriz de modelo y v el vértice que queremos transformar, el nuevo vértice v' quedaría expresado como:

$$v' = M \cdot v \tag{4.1.3.20}$$

Matriz de vista

La matriz de vista en OpenGL controla la manera en que miramos a la escena. En OpenGL es común emplear el concepto de cámara en la matriz de vista y utilizar una función para configurarla. Dicha función suele declararse de la forma `lookAt(Vec3 eye , Vec3 center, Vec3 up)`:

- El primer parámetro *eye*, define la posición de la cámara.
- El segundo parámetro *center*, define la posición hacia la que apunta la cámara.
- El tercer parámetro *up*, define la dirección superior de la cámara.

La función anterior no deja de ser una forma sencilla de definir la matriz de vista:

$$\begin{bmatrix} (\frac{up\times\frac{center-eye}{|center-eye|}}{|up\times\frac{center-eye}{|center-eye|}|})_x & (\frac{center-eye}{|center-eye|}\times\frac{up\times\frac{center-eye}{|center-eye|}}{|up\times\frac{center-eye}{|center-eye|}|})_x & (\frac{center-eye}{|center-eye|})_x & 0 \\ (\frac{up\times\frac{center-eye}{|center-eye|}}{|up\times\frac{center-eye}{|center-eye|}|})_y & (\frac{center-eye}{|center-eye|}\times\frac{up\times\frac{center-eye}{|center-eye|}}{|up\times\frac{center-eye}{|center-eye|}|})_y & (\frac{center-eye}{|center-eye|})_y & 0 \\ (\frac{up\times\frac{center-eye}{|center-eye|}}{|up\times\frac{center-eye}{|center-eye|}|})_z & (\frac{center-eye}{|center-eye|}\times\frac{up\times\frac{center-eye}{|center-eye|}}{|up\times\frac{center-eye}{|center-eye|}|})_z & (\frac{center-eye}{|center-eye|})_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.1.3.21)$$

Por defecto en OpenGL *eye* se encuentra en las coordenadas (0, 0, -1), *center* en (0, 0, 0) y *up* apunta en la dirección positiva del eje Y, (0, 1, 0).

El uso de la función anterior devolvería una nueva matriz de 4x4 con los parámetros *eye*, *center* y *up* aplicados.

Siendo M la matriz de modelo, V la matriz de vista y v el vértice que queremos transformar, el nuevo vértice v' quedaría expresado como:

$$v' = V \cdot M \cdot v \quad (4.1.3.22)$$

Matriz de proyección

Si solo empleamos las dos matrices anteriores veremos que todos los objetos de nuestra escena mantienen el mismo tamaño, independientemente de donde se sitúe la cámara. Esto no es correcto ya que estamos perdiendo el concepto de profundidad, en que los objetos más cercanos a la cámara parecen más grandes que los que están más lejos.

Otro problema es que por defecto OpenGL establece que el área de dibujado sea igual al de un cubo unitario, es decir, un cubo de tamaño 1 unidad, haciendo que cualquier cosa que dibujemos con unas coordenadas que caigan fuera de ese cubo no sea visible. La matriz de proyección permite agrandar o disminuir el tamaño de dicho cubo.

En principio para solventar el problema del cubo unitario, la matriz de proyección puede ser configurada para establecer una proyección ortográfica en la cual no existe el concepto de profundidad. La proyección ortográfica suele ser empleada en aplicaciones gráficas 2D por ejemplo. Viene definida por los parámetros

- *left* y *right*, representan las posiciones del plano de corte izquierdo y derecho, respectivamente.
- *near* y *far*, representan las posiciones del plano de corte cercano y lejano, respectivamente.
- *bottom* y *top*, representan las posiciones del plano de corte inferior y superior, respectivamente.

La matriz que define una proyección ortográfica viene dada de la forma:

$$\begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{rigth+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1.3.23}$$

Para conseguir una proyección con profundidad, que se asemeja más a la visión humana, podemos aplicar una proyección en perspectiva. Dicha proyección viene definida por los parámetros:

- *Field of View (*FoV*)*, o ángulo de visión.
- *aspect*, o relación de aspecto.
- *near* y *far*, que representan las posiciones del plano de corte cercano y lejano, respectivamente.

Para conseguir la matriz de transformación de una proyección en perspectiva, se calculan los parámetros para una proyección ortográfica y se sustituyen en la matriz anterior:

$$\begin{aligned} top &= near \cdot \tan \frac{pi}{180} \cdot FoV/2 \\ bottom &= -top \\ right &= top \cdot aspect \\ left &= -right \end{aligned} \tag{4.1.3.24}$$

Siendo M la matriz de modelo, V la matriz de vista, P la matriz de proyección y v el vértice que queremos transformar, el nuevo vértice v' quedaría expresado como:

$$v' = P \cdot V \cdot M \cdot v \tag{4.1.3.25}$$

Biblioteca de ayuda matemática

Dada la complejidad matemática con la que tratamos al desarrollar aplicaciones que hagan uno de transformaciones 3D, se valoró la idea de emplear una biblioteca externa que facilitara esta tarea.

Se tuvieron en cuenta dos bibliotecas para este desempeño: OpenGL Mathematics (GLM) [2] y Configurable Math Library (CLM) [3]. Ambas resultaban muy similares y ofrecían prácticamente las mismas características por lo que se realizaron algunas pruebas de rendimiento para ver cual de las dos sobresalía. Las pruebas que se realizaron involucraban la suma y

[2] http://glm.g-truc.net/
[3] http://cmldev.net/

multiplicación de un millón de matrices de orden 4 con precisión de coma flotante. La prueba se repitió 10 veces para cada biblioteca y se analizaron los resultados. Se dedujo de esta forma que GLM resultaba vencedor en cualquier caso, ofreciendo un rendimiento 1.3 veces superior a CLM.

4.2 La arquitectura de red

La ejecución de tareas de alto rendimiento en dispositivos móviles tiende a resultar en una experiencia poco satisfactoria de cara al usuario debido a la limitada capacidad de estos terminales. A lo largo de los años se han utilizado diversas tecnologías y protocolos como Remote Procedure Call (RPC), para delegar la ejecución de dichas tareas en otros dispositivos con mayores recursos. Desde entonces y con la emergencia de los servicios web se ha popularizado el uso de otras tecnologías como REpresentational State Transfer (REST) y Simple Object Access Protocol (SOAP) que pueden ser utilizados para este propósito.

A la hora de realizar una arquitectura de red se debe de buscar la mejor solución posible y, en caso de que ya exista, replicarla siguiendo el principio de «no reinventar la rueda». El problema de implementar una solución por red es que existen numerosas soluciones, las cuales deben ser valoradas a la hora de decidirse definitivamente por una.

Antes de continuar se introducirán algunas definiciones que se emplearán a lo largo de esta sección.

4.2.1 Definiciones

Cliente/Servidor

Se trata de un tipo de arquitectura de red en que los computadores de una red asumen el rol de servidor, el cual selectivamente comparte sus recursos, y el cliente, el cual inicia el contacto con el servidor para usar dichos recursos [Pad82].

Sockets

Un Socket es definido por ser un identificador único para o desde el cual la información es transmitida en una red [Win71]. Se caracterizan por una única combinación de:

- Socket local: Dirección Internet Protocol (IP) local y número de puerto.
- Socket remoto: Empleado solo en sockets Transmission Control Protocol (TCP).
- Protocolo: Un protocolo de transporte (e.g. TCP, User Datagram Protocol (UDP), RAW IP, u otros).

HTTP

HyperText Transfer Protocol (HTTP) es un protocolo de nivel de aplicación para sistemas de información *hypermedia* distribuidos y colaborativos. Se trata de un protocolo genérico

que no guarda información sobre el estado que puede ser usado para multitud de tareas aparte de *hypertexto*, como por ejemplo sistemas de gestión de objetos distribuidos [FGM+99].

RPC

RPC es un protocolo para la ejecución remota de código; cuando un procedimiento remoto es invocado el entorno ejecutor es suspendido, los parámetros son transmitidos por la red al entorno donde se ejecutará dicho procedimiento y dicho procedimiento será ejecutado.

Cuando el procedimiento termina su ejecución y produce sus resultados, dichos resultados son devueltos al entorno ejecutor para su procesado [BN84].

SOAP

SOAP es un protocolo diseñado por IBM, Microsoft, Userland, y DevelopMentor que soporta RPC (y otras peticiones) sobre HTTP y el uso de mensajes XML [Mar01].

El Listado 4.3 muestra un ejemplo de mensaje SOAP cuya estructura puede verse en la Figura 4.6.

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Listado 4.3: Ejemplo de mensaje SOAP.

REST

REST es un estilo de arquitectura consistente en una serie de restricciones aplicadas a componentes, conectores y elementos de datos dentro de un sistema *hypermedia* distribuido. REST ignora los detalles de la implementación de los componentes y la sintaxis del protocolo para centrarse en el papel de los componentes, las restricciones de sus interacciones con otros componentes y su interpretación de elementos de datos significativos [FT02].

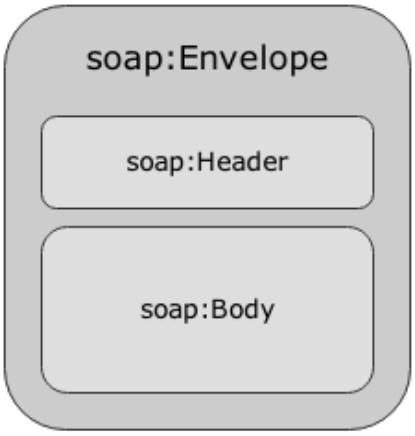

Figura 4.6: Estructura de un mensaje SOAP.

4.2.2 Estado del arte

Casos de estudio

Frente a las distintas tecnologías nombradas anteriormente, el uso de sockets puede parecer una opción a contemplar a la hora de desarrollar aplicaciones exigentes que dependan de cómputo remoto debido a su inherente rendimiento y simplicidad de despliegue. Sin embargo no siempre suponen la mejor opción debido a la dificultad de implementación. Los autores de [NSOJA13] desarrollaron una arquitectura para la visualización de mapas 3D en tiempo real utilizando sockets para la implementación de las comunicaciones; el servidor lanzaba un hilo de ejecución que escuchaba en un socket de red esperando conexiones de clientes y creando nuevas instancias del servidor por cada conexión existente. Los resultados de dicha implementación concluyeron que incluso un PC de escritorio era capaz de proporcionar una navegación fluida a través de los mapas a una gran cantidad de clientes de forma concurrente.

Recientemente y debido a la emergencia de los servicios web, tecnologías como REST o SOAP han alcanzado un punto álgido debido a su facilidad de uso y posibilidades. Se ha descartado, sin embargo, proponer un caso de uso de SOAP debido a su elevada carga de red por el empleo mensajes basados en Extensible Markup Language (XML) que incluyen distintas cabeceras impuestas por SOAP en cada mensaje [OO02]. En [ECCDPM12] los autores emplearon un servidor basado en arquitectura REST para controlar un robot a través de conexiones HTTP. Dado que la arquitectura REST está basada en recursos y verbos, los autores implementaron dos tipos de recursos:

- Para comprobar el estado de la conexión.
- Para obtener/actualizar la posición del robot.

Para el envío de información entre el cliente y el servidor se escogió el protocolo XML debido a la alta disponibilidad de intérpretes en lenguajes de alto nivel como Java. El uso de

REST en este caso fue motivado por el hecho de ser independiente de la plataforma y por ser fácilmente implementable en aquellos lenguajes que soportan servicios REST o en aquellos que solo ofrecen sockets.

Comparativa

De acuerdo a lo visto en la anterior sección, si tratamos de comparar HTTP y sockets llegamos a la conclusión de que la única desventaja de HTTP frente a estos últimos es el tamaño de los *data frames* enviados y recibidos, debido a que la cabecera HTTP es añadida cada vez a dichos datos [AFAN13]. Dado que el formato de los mensajes también influye en el rendimiento de la aplicación, entre los más populares para este cometido se encuentran XML y JavaScript Object Notation (JSON). Análisis empíricos demostraron que el uso de sockets resulta en una solución menos productiva y propensa a fallos aunque la cantidad de datos transmitidos sea inferior al transmitido por HTTP. Entre los formatos de mensajes nombrados, los archivos JSON poseen un tamaño más reducido que los XML, lo que los hace ideales a la hora de transferir información en entornos donde la velocidad es crítica [AFAN13].

Como evolución sobre el uso de sockets, RPC supuso un avance en la creación de servicios web y computación distribuida, debido a la encapsulación que ofrecía sobre los tradicionales sockets. Sin embargo, respecto al rendimiento frente a una arquitectura REST, al estar este último basado en estándares ampliamente usados en entornos Web, no requiere emplear estándares adicionales, lo cual evita la dependencia de plataformas específicas reduciendo así el uso de recursos del sistema [XJY09].

Por otra parte, SOAP puede verse como el sucesor natural de RPC, ya que toma la manera de transportar datos, interacción y formato de mensaje (envoltura, cabecera y cuerpo) [PPAB11]. Si tratamos de comparar SOAP con una arquitectura REST, el primero puede verse seriamente amenazado. Frente a SOAP, y al igual que frente a sockets, los servicios basados en REST pueden ser más escalables y confiables, además de ofrecer un mayor rendimiento. Sin embargo dicho rendimiento puede verse afectado por una de las restricciones que implica usar una arquitectura REST; el uso de un protocolo cliente/servidor que no soporta estados, como lo es HTTP, al contrario que SOAP, el cual confía de forma implícita en el control de flujo de estados [LG10]. Esto último, sin embargo, no debe de ser un problema a la hora de utilizarlo en un entorno de computación distribuida en el que solo se invoque a procedimientos remotos y estos devuelvan un resultado.

Si orientamos la arquitectura REST a los procesos de negocio de una aplicación podemos instaurar principios Web en sistemas de cómputo intensivo. En [XZLS08], por ejemplo, se realizó una infraestructura que soportara procesos de negocio orientados a REST. Entre algunas de sus características, se encontraba

- el uso de verbos HTTP estándar,

- exponer toda la información como recursos y promover el uso de comunicaciones libres de contexto, sin estados y de alta visibilidad

entre otras.

4.2.3 Conclusión

El uso de tecnologías empleadas para ofrecer servicios web resulta en una buena opción a la hora de desarrollar un entorno de computación distribuida en que un dispositivo móvil descargue tareas intensivas en otra máquina con mayores recursos.

Entre las observadas en este documento podemos afirmar que una implementación de arquitectura REST puede significar una forma sencilla, simple y eficaz a la hora de desplegar un sistema de este tipo en un entorno multiplataforma. El uso de otras tecnologías presenta algunos inconvenientes frente a REST, que hemos enumerado a lo largo de este documento, como pueden ser la lentitud del entorno sobre sistemas críticos y la dificultad de implementación.

Sin embargo, debemos tener en cuenta los requisitos y condiciones disponibles. La arquitectura de red debe ser desplegada como mínimo sobre sistemas GNU/Linux empleando el lenguaje de programación C++ y que soporte una comunicación entre arquitecturas ARM y x86_64. Por esto, una implementación por sockets supone una manera estándar y común de desplegar una arquitectura de red, la cual permite una mayor flexibilidad y control sobre esta a la hora de hacer ciertas optimizaciones sobre la plataforma. A costa de esto perdemos ciertas facilidades de escalabilidad que nos proporcione una arquitectura REST sobre un servidor HTTP; buscamos sin embargo eficiencia y velocidad en la comunicación, por lo que adoptar una implementación por sockets se ve justificada.

4.3 Reproducción de audio

La reproducción de sonidos desde archivo no supuso ningún problema. Ya se había trabajado antes este ámbito y se usó directamente lo que ya se conocía, **SDL_Mixer**. Pese a esto se decidió realizar algunas pruebas en la unidad de detección y despliegue para asegurarse el correcto funcionamiento.

Después de cargar algunos archivos de audio, tanto codificados como sin codificar, el sistema se comportó correctamente; no se manifestaron problemas de latencia, ni un bajo rendimiento durante la carga de los archivos.

Las condiciones de las pruebas fueron realistas; el sistema se encontraba funcionando perfectamente detectando documentos y enviando las imágenes por red, por lo que los resultados obtenidos fueron más que satisfactorios.

4.4 Lenguaje de *scripting*

La creación de un lenguaje de *scripting* que fuera lo suficientemente sencillo de utilizar y de desarrollar no supuso mayor problema.

Se pretendía que el lenguaje fuera lo suficientemente autónomo para no depender de ninguna utilidad externa para su creación, es decir, no se buscaba construir ni utilizar procesadores de lenguaje para algo que se pretendía fuera simple. Por lo tanto, la construcción del lenguaje se relacionó íntimamente con el lenguaje de programación C++, de tal forma que fuera éste quien interpretara el contenido del archivo, procesándolo línea a línea y actuando en consecuencia con cada instrucción encontrada.

Se estudió el formato de archivo 3D .OBJ [Obj95]. Dicho formato sirve para almacenar información sobre los vértices, caras y texturas de un modelo geométrico 3D. La sintaxis de dicho formato es muy simple y se basa en la definición de líneas de texto que son leídas secuencialmente por alguna herramienta de importación de modelos para reconstruir el modelo que guarda el archivo.

En próximos capítulos se profundizará más en detalle sobre el lenguaje de *scripting* realizado.

4.5 Soporte de videollamadas

Debido a que la unidad de detección y despliegue mantenía ocupada la cámara continuamente, se decidió reutilizar la biblioteca de visión por computador, **OpenCV**, utilizada por GrayAR en principio para la detección y análisis de los documentos. De esta forma se empleó dicha biblioteca para interceptar las imágenes capturadas por la cámara, comprimirlas y enviarlas por red hacia el otro extremo de la videollamada. En el otro extremo se hizo lo mismo pero mandando las imágenes de una nueva cámara web a la unidad para su despliegue usando el proyector.

4.6 Diseño software

En este apartado trataremos las herramientas que hemos empleado para la realización de un pequeño prototipo 3D del proyecto ARgos y las bibliotecas estudiadas para el renderizado de texto en OpenGL ES 2.0.

4.6.1 Herramientas externas

Como una pequeña demostración, se decidió realizar un prototipo virtual del proyecto ARgos que mostrara el funcionamiento del sistema de forma limitada. Por ser la herramienta de modelado 3D libre y multiplataforma por excelencia, se empleó **Blender** para tal trabajo.

Blender incluye un motor de videojuegos que facilitó la tarea de implementar la escena 3D del sistema en un medio interaccionable para el usuario.

Además, toda la lógica fue realizada empleando el lenguaje de programación Python, por estar íntimamente relacionado con Blender.

4.6.2 Bibliotecas externas

La carga de fuentes de texto en OpenGL ES fue una de las tareas que más dificultad conllevó. No existe sin embargo a día de hoy más que una implementación totalmente funcional de una biblioteca que permita la carga de fuentes true-type desde archivo y que funcione sobre OpenGL ES; **freetypeGlesRpi** realiza bien su trabajo, sin embargo la biblioteca se encontraba desarrollada totalmente en C, por lo que hubo que adaptarla a BelfegAR y C++.

Capítulo 5

Método de trabajo

La elección de una buena metodología de trabajo que se adapte de forma natural al proyecto a desarrollar resulta crucial a la hora de realizar de forma satisfactoria su implementación. A lo largo de este capítulo se detallará el método de trabajo elegido. Del mismo modo se listarán las herramientas software y hardware utilizadas.

5.1 Metodología de trabajo

Dada la naturaleza de BelfegAR y su estrecha relación con GrayAR, se necesitaba emplear una metodología que permitiera a ambas partes del proyecto ARgos trabajar de forma paralela añadiendo nuevas características y corrigiendo los errores existentes para finalmente probar la funcionalidad correcta del sistema completo.

Por lo anterior se impuso un método de trabajo basado en reuniones semanales e hitos a cumplir de manera iterativa e incremental respetando siempre los requisitos de diseño del sistema. Dichos hitos serán detallados en el Capítulo 6.

BelfegAR basa su diseño en una arquitectura orientada a componentes y subsistemas, de tal forma que dichos componentes pueden ser implementados y probados de forma aislada sin requerir la total implementación de GrayAR. Cuando se finalizaba la implementación de cada módulo, se realizaban las pruebas de integración pertinentes y se completaba el hito correspondiente.

5.1.1 Scrum

Según se ha detallado anteriormente, la metodología que más se aproxima a lo explicado podría ser *Scrum*. *Scrum* se define como un marco de trabajo que sirve para afrontar problemas de forma adaptable y lograr soluciones de alto valor [SS13]. Otros autores [Kni07] definen *Scrum* como una metodología ágil que puede ser aplicada a casi cualquier proyecto. Aun así, es principalmente usada en el desarrollo de software. *Scrum* se adapta muy bien a proyectos cuyos requisitos cambian constantemente. El proceso de desarrollo evoluciona a través de iteraciones planificadas al comienzo de estas y realizando una revisión del trabajo realizado finalmente para comprobar si se ha cumplido con la iteración.

El principio clave de *Scrum* es su conciencia de que durante un proyecto los clientes pue-

den cambiar de parecer respecto a lo que quieren, haciéndoles impredecibles, aceptando por tanto que el problema no puede ser totalmente comprendido desde el principio, centrándose entonces en maximizar la habilidad del equipo para responder de forma rápida ante nuevos requisitos. Este tipo de planteamiento sería imposible de afrontar en un proceso de desarrollo basado en cascada.

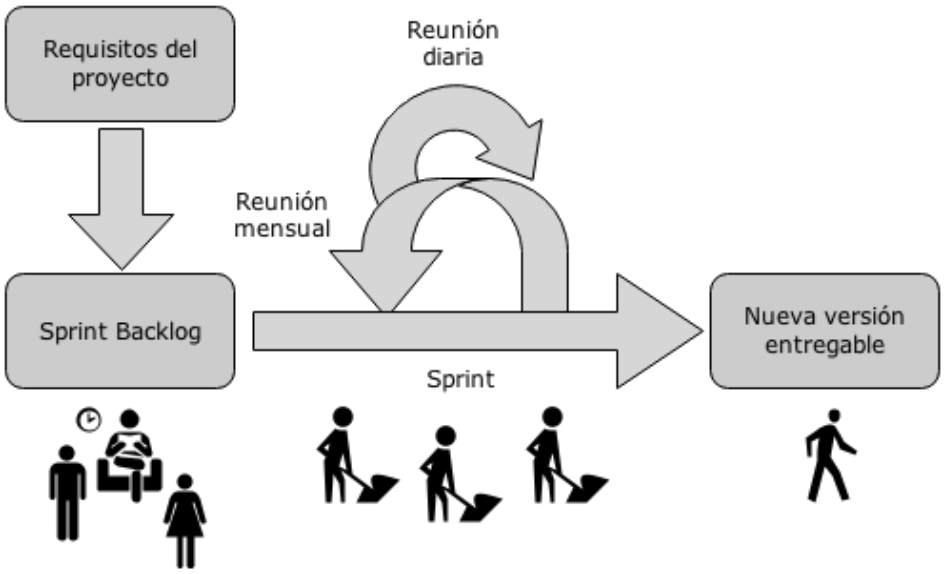

Figura 5.1: Marco de trabajo Scrum.

La Figura 5.1 muestra una visión general de como funciona *Scrum*. A continuación se explicarán en detalle los distintos componentes del marco de trabajo Scrum y cómo se ha aplicado en el desarrollo de este proyecto.

Roles del equipo Scrum

El equipo de *Scrum* se compone principalmente de 3 roles fundamentales: **Product Owner**, **Equipo de Desarrollo** y **Scrum Master** [SS13]. El equipo se auto-organiza a sí mismo asignándose entre ellos los roles que mejor saben hacer.

Product Owner

El *Product Owner* es responsable de maximizar el valor del producto y el trabajo del *Equipo de Desarrollo*, es decir, representa la voz del cliente. Se trata de la única persona capaz de gestionar el *Product Backlog*, entre lo que se incluye:

- Indicar las *historias de usuario* de forma clara en el *Product Backlog*.
- Ordenar las *historias de usuario* del *Product Backlog* para lograr mejor los objetivos.
- Optimizar el valor del trabajo que el *Equipo de Desarrollo* realiza.
- Asegurar que el *Product Backlog* es visible, transparente y claro para todos, y mostrar al equipo que es lo próximo en lo que se va a trabajar.

- Asegurar que el *Equipo de Desarrollo* entiende las *historias de usuario* del *Product Backlog*.

Las decisiones del *Product Owner* deben ser respetadas para afrontar el proyecto de forma satisfactoria. Sus decisiones se hacen efectivas de manera visual en el *Product Backlog* y el *Equipo de Desarrollo* debe cumplir con los requisitos expuestos por el *Product Owner*; ningún integrante del equipo puede indicar al *Equipo de Desarrollo* trabajar en otro conjunto de requisitos, ni siquiera entre ellos.

En ARgos, Juan Carlos López López se identificaría con este rol por representar al cliente.

Equipo de Desarrollo

El equipo de desarrollo se compone de profesionales encargados de entregar el producto al cliente. Son los únicos capaces de aumentar la versión del proyecto y de decidir cuando una *historia de usuario* se ha completado o no.

Entre las características del *Equipo de Desarollo* se encuentran:

- Auto-organización. Nadie, ni siquiera el *Scrum Master*, puede decirle al *Equipo de Desarrollo* cómo transformar las *historias de usuario* del *Product Backlog* en finalizadas.
- Los *Equipos de Desarrollo* son multifuncionales, es decir, todos poseen las habilidades necesarias para completar las *historias de usuario* del *Product Backlog*.
- *Scrum* no asigna títulos a los miembros del *Equipo de Desarrollo* más allá de «desarrollador» a pesar del trabajo que realice cada miembro.
- *Scrum* no reconoce sub-equipos en el *Equipo de Desarrollo*, a pesar de ciertos dominios que necesitan ser abarcados como «pruebas» o «análisis de negocio».
- Los miembros del *Equipo de Desarrollo* pueden tener ciertas especialidades o centrarse en diferente áreas, pero la responsabilidad del proyecto pertenece íntegramente al *Equipo de Desarrollo* en su conjunto.

Por lo general, el *Equipo de Desarrollo* suele ser de entre 3 y 9 personas; menos de tres personas disminuye la interacción y resulta en ganancias de productividad inferiores, mientras que más de nueve personas requiere demasiada coordinación.

Este rol estaría compuesto en ARgos por Santiago Sánchez Sobrino y Manuel Hervás Ortega, y en BelfegAR por el primero.

Scrum Master

Se encarga de asegurar que la metodología *Scrum* sea comprendida y empleada por el resto del equipo. Esto se consigue asegurándose de que el equipo se adhiera a la teoría, prácticas y reglas de *Scrum*.

Se puede categorizar el trabajo que realiza el *Scrum Master* hacia los otros individuos del equipo:

Apoyo al Product Owner

- Encuentra técnicas para una gestión efectiva del *Product Backlog*.
- Ayuda al equipo a entender la necesidad de mantener las *historias de usuario* lo más claras y concisas posibles.
- Asegura que el *Product Owner* sabe como organizar el *Product Backlog* para maximizar su valor.
- Ayuda a entender y practicar el desarrollo ágil.
- Facilita los *eventos de Scrum*, ya sea por petición o por necesidad.

Apoyo al Equipo de Desarrollo

- Entrena al *Equipo de Desarrollo* en la auto-organización y multifuncionalidad.
- Ayuda al *Equipo de Desarrollo* a crear productos de alto valor.
- Elimina impedimentos que dificulten el progreso del *Equipo de Desarrollo*.
- Facilita los *eventos de Scrum* ya sea por petición o por necesidad.
- Entrena al *Equipo de Desarrollo* en entornos organizaciones en los que *Scrum* aún no se ha adoptado y comprendido en su totalidad.

Apoyo a la Organización

- Dirige y entrena la organización en su adopción de *Scrum*.
- Planifica las implementaciones de *Scrum* dentro de la organización.
- Provoca cambios que aumenten la productividad del equipo.
- Trabaja con otros *Scrum Masters* para aumentar la efectividad de la aplicación de *Scrum* en la organización.

Este rol estaría representado por Carlos González Morcillo por ser el tutor de este Trabajo Fin de Grado.

Eventos y prácticas de Scrum

Durante la aplicación del método *Scrum* en un proyecto se deben atender ciertos eventos y prácticas especiales. Dichos eventos son usados en *Scrum* para crear regularidad y para minimizar el número de reuniones no definidas por *Scrum*. Todos los eventos poseen un límite de tiempo, es decir, tienen una duración máxima que no puede ser acortada ni prolongada de ninguna manera.

Entre los eventos y prácticas que define *Scrum* se encuentran:

Product Backlog

Se trata de una lista ordenada de todo lo que podría requerir el producto final. El *Product Owner* es el responsable del *Product Backlog* incluyendo su contenido, disponibilidad y organización.

Un *Product Backlog* nunca está completo. En su fase más temprana solo recoge los requisitos fundamentales y bien conocidos. Conforme el producto evoluciona, así lo hace el *Product Backlog*, añadiendo nuevos requisitos a este; mientras que el producto exista, así lo hará el *Product Backlog*.

El *Product Backlog* recoge las historias de usuario, las cuales están formadas de una descripción, prioridad, estimación y valor del requisito, función o mejora que representan. Sin embargo *Scrum* no obliga a añadir únicamente historias de usuario al *Product Backlog*, sino cualquier *item* que aporte valor a los clientes.

Conforme el producto es usado y gana valor y los clientes proporcionan retroalimentación, el *Product Backlog* se convierte en una lista más grande y exhaustiva. Los requisitos nunca dejan de cambiar; cambios en los requisitos de negocio, condiciones de mercado o tecnología pueden provocar cambios en el *Product Backlog*.

Sprint

Se trata del corazón de *Scrum*. Los *Sprints* son etapas de duración límite (como un mes o menos), durante las cuales se crean nuevas versiones del producto atendiendo a los requisitos completados.

Cada *Sprint* puede ser considerado un proyecto con una duración de un mes. Igual que los proyectos, los *Sprints* son usados para conseguir algo. Cada *Sprint* tiene una definición de qué va a ser construido, un diseño y un plan que guiará su creación, el trabajo y el producto final.

Los *Sprints* están limitados a un mes. Cuando la duración de un *Sprint* es mayor a un mes, la definición de qué se está construyendo puede cambiar y por tanto la complejidad y el riesgo pueden verse incrementados.

Sprint Planning Meeting

El trabajo que se va a realizar en un *Sprint* es definido en el *Spring Planning*, el cual es creado de forma colaborativa por el equipo.

Los *Sprint Plannings* son eventos de duración limitada a 8 horas como máximo, dependiendo de si el *Sprint* tiene una duración de un mes. En caso de *Sprints* más cortos la duración de este evento normalmente también se ve reducida. El *Scrum Master* se asegura de que el evento tenga lugar y sus asistentes entienden su propósito. El *Scrum Master*, además, es quien comunica al resto del equipo la necesidad de mantener el evento dentro de su límite de tiempo.

En los *Sprint Plannings* se realizan las siguientes actividades:

- Elegir el trabajo a ser realizado.
- Preparar el *Sprint Backlog* que detalle el tiempo que llevará realizar ese trabajo con el equipo.
- Identificar y comunicar la cantidad de trabajo que se va a realizar durante el *Sprint* actual.
- Si la duración del evento es de 8 horas, las primeras 4 horas se emplean para que el equipo priorice los *items* del *Product Backlog*. Las otras 4 horas finales, sirven para que el *Equipo de Desarrollo* elabore un plan para el *Sprint* que resulte en el *Sprint Backlog*.

En general, el *Sprint Planning* intenta responder a dos preguntas:

- ¿Qué puede ser hecho en este *Sprint*?
- ¿Cómo se completará el trabajo elegido?

Sprint Backlog

Se trata del conjunto de *items* del *Product Backlog* seleccionados para el próximo *Sprint*. Dichos *items* son añadidos hasta que el *Equipo de Desarrollo* crea que tiene suficiente trabajo para llenar el *Sprint*. Para ello, el *Equipo de Desarrollo* realiza preguntas para ver cómo de factible es realizar cierto *item*

La organización del *Sprint Backlog* depende de cada equipo. Por norma general, los *items* suelen ser categorizadas en disposición de columnas divididas en «Por hacer», «En progreso» y «Finalizado». Herramientas como Trello [1] facilitan la labor de mantener un *Sprint Backlog* organizado.

Una vez que el *Sprint Backlog* es rellenado, nadie aparte del *Equipo de Desarrollo* puede añadir *items* a este.

Daily Scrum Meeting

El *Daily Scrum* es un evento de 15 minutos de duración para el *Equipo de Desarrollo* en el que se sincronizan las actividades y se crea un plan para las próximas 24 horas. Esto se hace inspeccionando el trabajo realizado desde el último *Daily Scrum* y prediciendo el trabajo que puede ser hecho antes del siguiente.

Durante este evento los miembros del *Equipo de Desarrollo* explican:

- Qué hicieron ayer que ayudará al *Equipo de Desarrollo* cumplir con el *Sprint*.
- Qué harán hoy para ayudar al *Equipo de Desarrollo* a cumplir con el *Sprint*.
- Si ven algún impedimento que dificulte al *Equipo de Desarrollo* cumplir con el *Sprint*.

[1] `https://trello.com/`

El *Scrum Master* se asegura de que el *Equipo de Desarrollo* cumple con el evento, pero es el propio *Equipo de Desarrollo* el que debe conducirlo en primera instancia. Además, el *Scrum Master* se asegura también de que en este evento solo participen los miembros del *Equipo de Desarrollo*.

Sprint Review

Al final del *Sprint* se lleva a cabo el *Sprint Review*. En esta etapa se revisa el trabajo completado y el que no se llegó a completar, y se presenta el trabajo completado a los clientes en forma de demostración.

Los *Sprint Review* tienen de duración 4 horas como máximo. Si el *Sprint* dura menos de un mes normalmente la duración de esta etapa se ve reducida.

El resultado del *Sprint Review* es un nuevo *Product Backlog* revisado que define nuevos y probables *items* que se realizarán en el próximo *Sprint*.

Sprint Retrospective

Esta etapa presenta una oportunidad al equipo para dar sus impresiones sobre el *Sprint* que acaban de completar. Esto se hace con el propósito de continuar mejorando el proceso de *Scrum*.

La duración límite de este evento es de 3 horas, aunque igual que antes si el *Sprint* dura menos de un mes también durará menos el *Sprint Retrospective*.

Al finalizar este evento el equipo debería haber identificado nuevas mejoras que se incluirán en el próximo *Sprint*. La implementación de dichas mejoras supone la adaptabilidad del equipo al método *Scrum*. Las mejoras pueden ser implementadas en cualquier momento, sin embargo, el *Sprint Retrospective* proporciona una oportunidad formal de centrarse en la inspección del método y la adaptabilidad.

5.2 Herramientas empleadas

En esta sección se detallan las herramientas que han sido empleadas para la implementación de BelfegAR, además de las necesarias para la elaboración de este documento, entre otras.

5.2.1 Medios hardware

Entre los medios necesarios para la puesta en marcha del sistema y realización de las pruebas, se han requerido los siguientes medios hardware:

- **Raspberry Pi**[2]**:** Se trata de la unidad de detección y despliegue del sistema. En el Anexo A pueden verse las características oficiales del modelo empleado.

[2] `http://www.raspberrypi.org/`

- **Proyector portátil:** Se trata del medio principal que permite el despliegue de gráficos del sistema. El proyector empleado es un Optoma PK320 que soporta una resolución panorámica máxima de 854x480.
- **Cámara:** Se trata de una minicámara de 5MP (`RPI CAMERA BOARD` con un sensor Omnivision 5647) que se conecta directamente a la Raspbery Pi a través de su conector Camera Serial Interface (CSI). La cámara es empleada para la detección de documentos y la realización de videollamadas.
- **Computador:** Para el desarrollo del proyecto ha sido necesario hacer uso de un computador conectado por red a la plataforma Raspberry Pi, para poder acceder remotamente a ella y de esta forma ejecutar las pruebas. Dicho computador ha sido proporcionado por el laboratorio Oreto para el desarrollo del proyecto, con un procesador Intel Core 2 Duo T5500 1.66 GHz y 2GB de memoria RAM.
- **Altavoz:** Se trata de un circuito integrado con altavoz propio, modelo Waveshare LM386, utilizado para la reproducción de audio por el sistema.
- **Tarjeta de memoria:** Para el correcto funcionamiento de la unidad de detección y despliegue del sistema, se necesita una tarjeta de memoria cuyos tiempos de lectura y escritura fueran los más rápidos posibles. En nuestro caso empleamos una MicroSDHC Trascend de 32GB clase 10, que proporciona unas velocidades de lectura y escritura de 20 MB/s y 17 MB/s respectivamente.

5.2.2 Medios software

A continuación se enumeran las diversas herramientas software empleadas y diferenciadas por categorías:

Sistemas operativos

- **Debian GNU/Linux** [3]**:** Para la elaboración del trabajo se ha elegido esta distribución GNU/Linux, concretamente la versión *testing* por ofrecer un equilibrio idóneo entre estabilidad, actualizaciones de software y facilidad de uso.
- **Linux Mint** [4]**:** También se ha empleado esta distribución GNU/Linux, en su versión 17 o *Quiana* y Xfce, para comprobar la compatibilidad del sistema entre diferentes distribuciones. Se ha elegido esta por su popularidad y por estar basada en Ubuntu.

Herramientas de desarrollo

- **Emacs**[5]**:** El versátil editor Emacs se ha utilizado en su versión 24.3.1 para la elaboración de este documento como entorno de desarrollo y como software de acceso

[3] `http://www.debian.org/`
[4] `http://www.linuxmint.com/`
[5] `http://www.gnu.org/software/emacs/`

remoto mediante *SFTP* para editar archivos en la Raspberry Pi. Se han empleado, además, algunos modos o *plugins* como Emacs Code Browser (ECB) [6], que añade algunas funcionalidades de un entorno de desarrollo completo a Emacs. Para la elaboración de este documento se ha empleado el paquete *AUCTeX* [7] para integrar las funciones de LaTeXen el propio editor.

- **OpenSSH** [8]**:** Para realizar accesos remotos a la unidad de detección y despliegue se ha empleado esta utilidad CLI en su versión 6.6. Además, también se han empleado otras utilidades gráficas que funcionan sobre el mismo protocolo Secure SHell (SSH), como es el propio gestor de archivos Thunar [9], en su versión 1.6.3, proporcionado en el entorno de escritorio Xfce [10].

- **GCC** [11]**:** Se ha utilizado el compilador *g++* para C++ de la colección de compiladores GNU. Concretamente se ha empleado la versión 4.8.2 para poder hacer uso de las nuevas características introducidas en el nuevo estándar C++11.

- **GDB** [12]**:** Tanto para depurar la aplicación cliente como la aplicación del lado del servidor se ha empleado esta utilidad en su versión 7.7. La depuración del cliente se ha realizado de forma remota, sin embargo, para la depuración del servidor se empleó una interfaz gráfica que operaba sobre GDB, llamada *Kdbg* [13] en su versión 2.5.4.

- **Git** [14]**:** Se ha empleado este sistema de control de versiones para mantener un repositorio con el proyecto y la documentación para mantener un historial de cambios y un acceso centralizado. El servicio de repositorios utilizado ha sido proporcionado por *Bitbucket* [15]. Concretamente se ha empleado el cliente en su versión 1.9.1.

- **Make** [16]**:** Para la generación de los ejecutables y como sistema de construcción se han empleado *Makefiles* para compilar y enlazar el proyecto junto a dicha herramienta en su versión 3.81.

- **Scratchbox 2** [17]**:** Se trata de un entorno completo de compilación cruzada empleado para poder construir el proyecto directamente desde el computador de desarrollo sin depender de la *Raspberry Pi*. En el Anexo B se detalla el proceso de instalación y puesta a punto del entorno. Se ha utilizado el paquete en su versión 2.3.42.

[6] `http://ecb.sourceforge.net/`
[7] `http://www.gnu.org/software/auctex/`
[8] `http://www.openssh.com/`
[9] `http://docs.xfce.org/xfce/thunar/start`
[10] `http://www.xfce.org/`
[11] `https://gcc.gnu.org/`
[12] `http://www.gnu.org/software/gdb/`
[13] `http://www.kdbg.org/`
[14] `http://git-scm.com/`
[15] `https://bitbucket.org/`
[16] `http://www.gnu.org/software/make/`
[17] `https://maemo.gitorious.org/scratchbox2`

- **QEMU** [18]**:** Emulador de máquinas virtuales requerido por *Scratchbox 2* para la instalación del entorno de compilación cruzada. Se ha empleado en su versión 2.0.0.
- **Blender** [19]**:** Se ha empleado este entorno de modelado 3D, en su versión 2.69, para la elaboración del prototipo virtual de BelfegAR.
- **Raspbian** [20]**:** Distribución GNU/Linux basada en Debian utilizada como sistema operativo de la unidad de detección y despliegue, Raspberry Pi. Aunque se ha empleado una imagen de la distribución con fecha 25-09-2013 se ha mantenido el sistema actualizado en la rama *texting* de Debian.

Lenguajes de programación

- **C++: [Str13]** Por ser un lenguaje orientado a objetos, que ofrece un rendimiento óptimo bajo plataformas empotradas y debido a la alta disponibilidad de distintas bibliotecas existentes para este lenguaje, C++ se ha convertido en el lenguaje elegido para implementar el proyecto. También se ha empleado el lenguaje C para adaptar algunas bibliotecas externas implementadas en dicho lenguaje para que se adecuen a las necesidades del proyecto. Se han empleado características de la versión C++11 del estándar.
- **GLSL: [MGS08]** El lenguaje de programación de *shaders* para OpenGL. Para la elaboración de los distintos componentes gráficos se hace uso de distintos *shaders* que sirvan para dibujar círculos, aplicar texturas o simplemente transformar los vértices de los componentes.
- **Python: [Lut07]** Blender requiere que la lógica programable se haga en este lenguaje. Para nuestro prototipo virtual de BelfegAR se empleó este lenguaje para llevar a cabo toda la implementación lógica del servidor, el cuál se encontraba incorporado directamente sobre la aplicación realizada en Blender. La versión de Python empleada ha sido la 2.7.6.
- **Bash: [NVAV07]** Para crear algunas pequeñas utilidades se empleó este lenguaje de *scripting shell*.

Bibliotecas

- **OpenGL ES 2.0: [MGS08]** Como dijimos anteriormente, se ha empleado esta versión de OpenGL por soportar aceleración gráfica en la unidad de detección y despliegue.
- **SDL_Mixer 1.2** [21]**:** Por su facilidad de uso y potencia se ha empleado esta biblioteca para la reproducción de audio del sistema.

[18] `http://wiki.qemu.org/Main_Page`
[19] `http://www.blender.org/`
[20] `http://www.raspbian.org/`
[21] `http://www.libsdl.org/projects/SDL_mixer/release-1.2.html`

- **OpenCV 2.3.1: [BK13]** Pese a ser usada en su mayor parte por GrayAR, *OpenCV* ha sido usada como base para construir el sistema de videollamadas.
- **freetypeGlesRpi** [22]**:** Como se comentó anteriormente, son escasas las bibliotecas de carga y despliegue de textos basados en fuentes *true-type*. Esta biblioteca ofrece una solución a tal problema.
- **GLM 0.9.5:** Se trata de una biblioteca matemática que abstrae al desarrollador de las operaciones algebraicas empleadas a la hora de realizar transformaciones 3D (véase § 4.1.3).
- **Simple OpenGL Image Library (SOIL) 1.07** [23]**:** Se trata de una pequeña biblioteca escrita en C usada principalmente para cargar imágenes como texturas en OpenGL. Soporta una gran cantidad de formatos, como PNG, JPEG y BMP, entre otros.
- **Boost.Asio 1.50** [24]**:** C++ no especifica nada relativo a redes o *sockets* en su estándar. Por esto mismo se ha empleado la biblioteca *Asio* de *Boost* para gestionar la comunicación por red.

Documentación

- **LaTeX** [25]**:** Para la elaboración de este documento se ha empleado esta herramienta de creación de documentos profesionales. Concretamente, se ha utilizado la implementación *TeX Live* [26] en su versión lanzada en 2013.
- **draw.io** [27]**:** Se trata de una aplicación web para la creación de todo tipo de diagramas de forma sencilla. Se ha utilizado para elaborar la mayoría de diagramas de este documento.
- **Gimp** [28]**:** El programa de manipulación de imágenes de GNU GIMP es un editor de imágenes avanzado. Se ha utilizado en su versión 2.8.14 para realizar pequeños retoques a algunas imágenes de este documento.
- **Inkscape** [29]**:** Herramienta para la creación de gráficos vectoriales. Algunas imágenes de este documento han sido creadas utilizando esta herramienta en su versión 0.48.4.
- **Doxygen** [30] Se trata de un generador de documentación en línea para multitud de lenguajes de programación. Se ha empleado en su versión 1.8.6 para generar la documentación técnica del *framework* del proyecto.

[22] https://github.com/PDKK/freetypeGlesRpi
[23] http://www.lonesock.net/soil.html
[24] http://www.boost.org/doc/libs/1_50_0/doc/html/boost_asio.html
[25] http://www.latex-project.org/
[26] https://www.tug.org/texlive/
[27] https://www.draw.io/
[28] http://www.gimp.org/
[29] http://www.inkscape.org/es/
[30] http://www.stack.nl/~dimitri/doxygen/

Capítulo 6

Resultados

A lo largo de este capitulo se discutirán los aspectos más relevantes del desarrollo de BelfegAR. Se planteará un enfoque descriptivo de cada uno de los subsistemas explicando su funcionamiento en detalle. Se presentarán además algunos diagramas que faciliten la comprensión del sistema. Del mismo modo se introducirán las distintas iteraciones llevadas a cabo en el proyecto. Se introducirán también algunas herramientas externas al proyecto que se crearon para hacer más sencillo el proceso de desarrollo. Finalmente, se hablará sobre los recursos y costes del sistema, es decir, medidas de rendimiento y estadísticas recogidas del sistema.

Tras introducir en el Capítulo 1 un diagrama general de los subsistemas que conforman BelfegAR (véase Figura 1.3), se presenta en la Figura 6.1 un esquema con mayor nivel de detalle.

Se indica a continuación el cometido de cada uno de los subsistemas:

- *Subsistema de Delegación de Tareas:* Se encarga de enviar la información necesaria obtenida en el cliente hacia el servidor para su procesado.

- *Subsistema de Comunicación:* Se encarga del procesado de la información y de enviar las indicaciones de qué hacer con la nueva información resultante al cliente.

- *Subsistema de Scripts:* Se encarga de gestionar, analizar e interpretar los diferentes *Scripts*. También se encarga de su carga desde disco.

- *Subsistema de Representación Gráfica:* Se ocupa de crear, gestionar y mostrar los diferentes componentes gráficos del sistema.

- *Subsistema de Audio:* Se encarga de cargar, gestionar y reproducir los archivos de sonido del sistema.

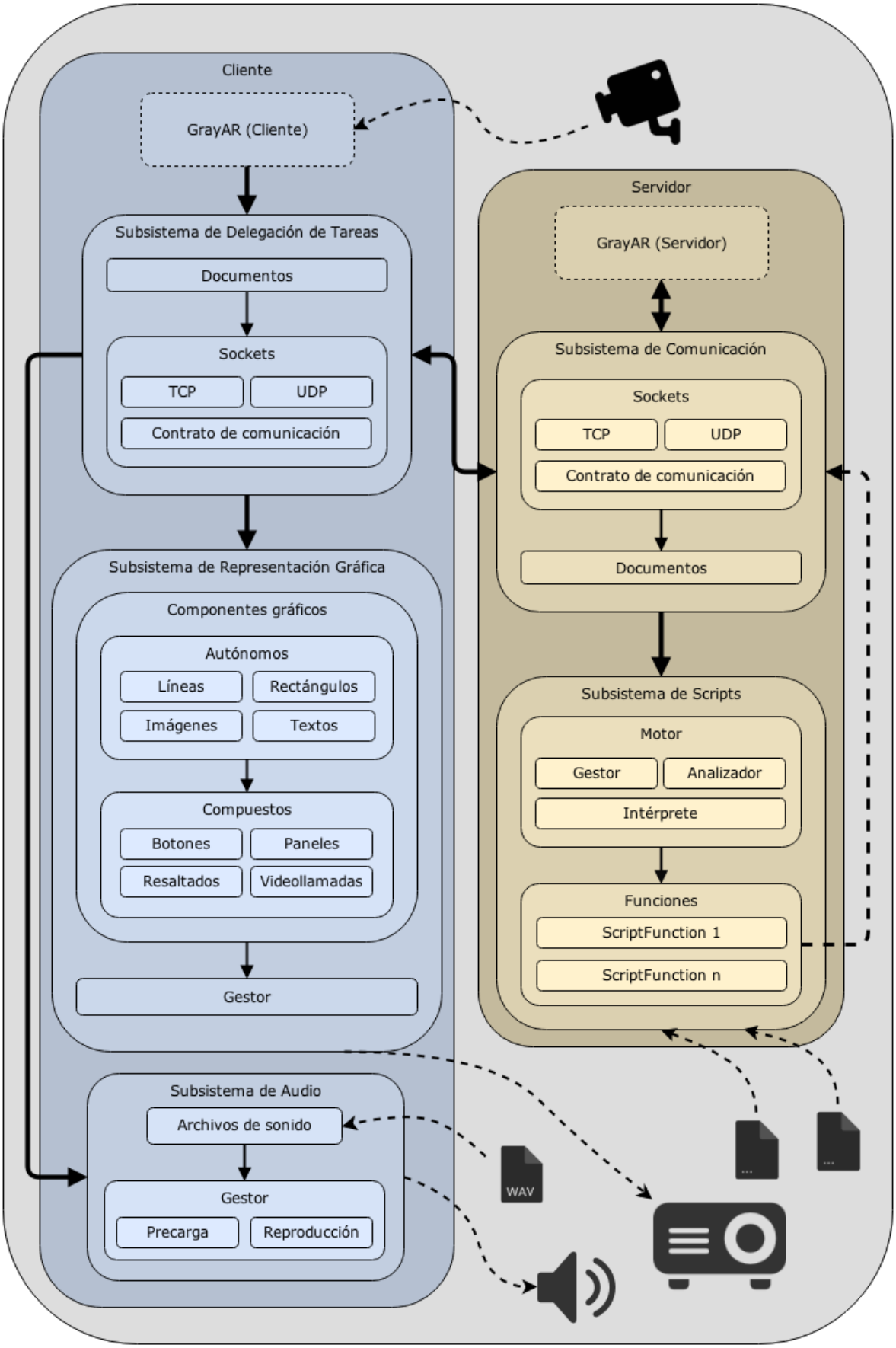

Figura 6.1: Esquema de subsistemas de BelfegAR.

6.1 Arquitectura del sistema

El diseño de los componentes que forman BelfegAR mantiene, en la gran mayoría de ocasiones, gran independencia minimizando el acoplamiento en su definición. Sin embargo los subsistemas de delegación de tareas y comunicación tienen cierta dependencia con el resto de subsistemas para su correcto funcionamiento.

En esta sección se procederá a explicar en detalle los subsistemas de BelfegAR haciendo una clara diferencia entre la parte del cliente y la del servidor.

Antes de entrar en los detalles técnicos de cada subsistema, se introducirá de forma general el funcionamiento del sistema completo mediante diagramas y descripciones.

6.1.1 Funcionamiento general

La Figura 6.2 muestra un diagrama de flujo con el funcionamiento general del sistema. Las competencias de GrayAR se han dibujado con cuadros punteados para diferenciarlas de BelfegAR.

El diagrama representa una idea general del funcionamiento del sistema. Algunas partes como la lectura de *scripts* o la carga de recursos gráficos y sonoros se ha omitido para mantener cierta simplicidad.

En detalle, la ejecución del sistema comienza con la pre-inicialización de BelfegAR. En esta primera etapa se inician los requisitos mínimos que no dependen del servidor, entre los que se encuentran:

- Configuración del manejador de señales para permitir interrumpir la ejecución mediante el registro de la señal `SIGINT`.
- Invocación de la función `bcm_host_init()` requerida por la Raspberry Pi antes de poder llamar a cualquier función que interactúe con la GPU.
- Inicialización del registro de eventos (`Log`) del cliente.

El registro de la señal `SIGINT` resulta fundamental para poder terminar la ejecución del sistema de forma limpia, liberando todos los recursos del sistema impidiendo que se produzcan fugas de memoria. La interrupción por tanto se puede realizar empleando cualquier técnica que emita la señal `SIGINT`, como la combinación de teclas `Control-C` o el uso del comando `kill` desde la terminal.

Pese a que el registro de eventos puede no ser visible si la aplicación se auto-ejecuta al arrancar el sistema, este se puede guardar en un archivo para su posterior inspección redirigiendo la salida (`STDOUT`) de la aplicación mediante el operador > de Linux. Si la aplicación se ha iniciado por red, a través del protocolo `SSH` por ejemplo, el registro de eventos será mostrado en pantalla en tiempo real.

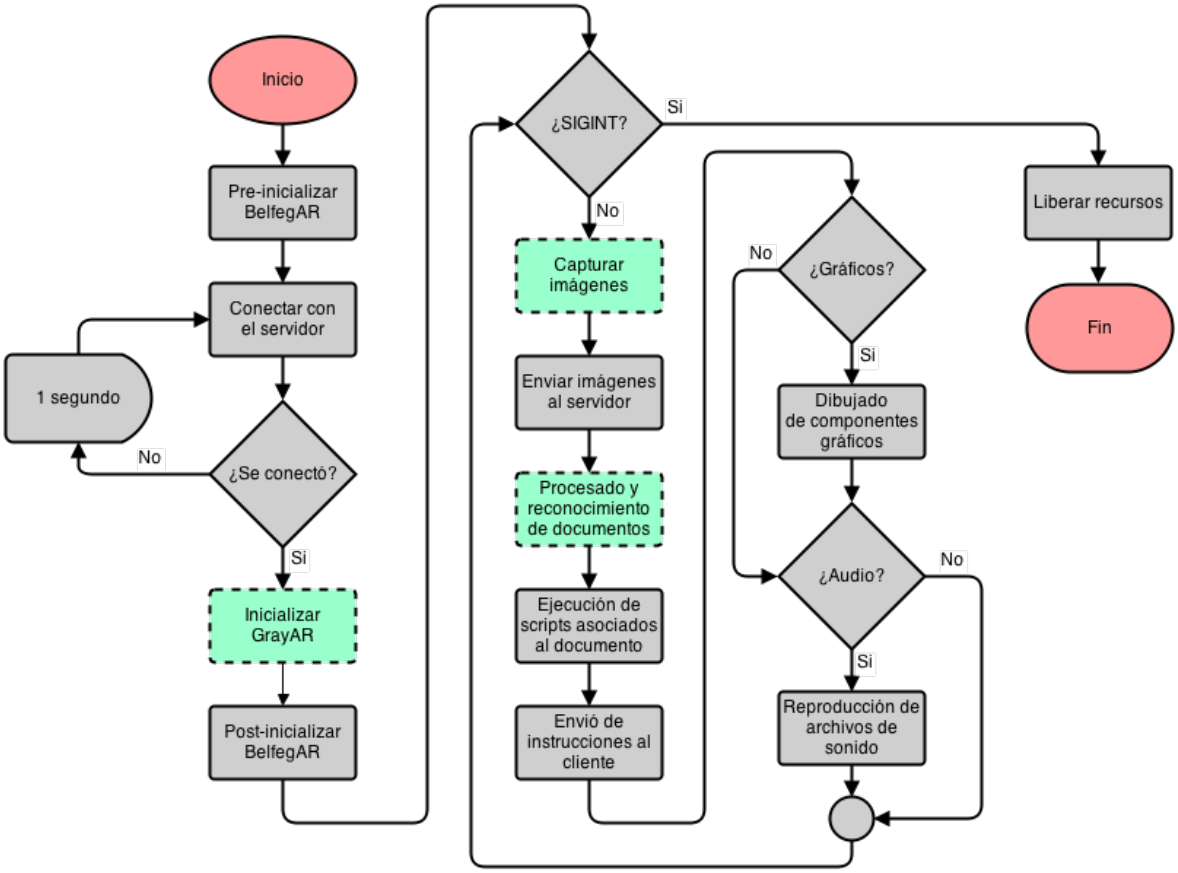

Figura 6.2: Diagrama de flujo del sistema.

Una vez completada la pre-inicialización el cliente entra en un bucle de conexión del que no sale hasta que ha conseguido establecer una conexión con el servidor. La conexión se realiza mediante *sockets* TCP que aseguran una comunicación adecuada y fiable. Una vez conectado satisfactoriamente con el servidor, el cliente inicializa todo el sistema GrayAR para dar paso a la etapa final de inicialización de BelfegAR o post-inicialización.

En el proceso de post-inicialización se recupera información vital obtenida de GrayAR. Dicha información corresponde con la matriz de proyección necesitada por BelfegAR para dibujar correctamente los elementos gráficos en el entorno. Después, el sistema procede a inicializar el contexto gráfico de OpenGL ES con la matriz de proyección proporcionada. En esta fase también se configuran los subsistemas de audio y gráficos con las rutas donde buscar los recursos a gestionar.

Con todo inicializado correctamente se pasa a la etapa principal de sistema. Esta etapa consiste en un conjunto de acciones que se repiten continuamente hasta que se interrumpa el sistema por parte del usuario o por algún tipo de error no recuperable. Dichas acciones consisten en la captura de imágenes por parte de GrayAR y el envío de éstas hacia el servidor. A partir de este momento todo el procesado se realiza en el servidor.

En lo que respecta a GrayAR, en la parte del servidor se ocupa de todo el procesado de las imágenes y de la detección de los documentos que encuentren en estas. Una vez detectados los documentos entra en juego BelfegAR para cargar los *scripts* asociados a dichos documentos. Una vez cargados, los *scripts* son analizados e interpretados. Por cada comando reconocido en el *script* se ejecutan ciertas funciones en el servidor que deciden qué procesos realizar y qué información enviar al cliente. Dicha información puede ser dibujar cierto componente gráfico o reproducir un archivo de sonido a petición. También puede derivar en realizar alguna interacción sobre el documento, como girarlo, por ejemplo, para iniciar una videollamada.

Finalmente, en el cliente se analiza la información recibida (si existe) y se actúa en consecuencia, dando paso otra vez al inicio del proceso.

6.1.2 Cliente

En esta sección se detallará, mediante diagramas de clases y explicaciones en profundidad, el funcionamiento de los subsistemas que operan en el cliente.

Subsistema de Delegación de Tareas

El subsistema de delegación de tareas hace posible que el sistema funcione de la forma más fluida posible. Se encarga de transmitir la información vital al servidor para que este pueda procesarla de forma autónoma y obtener los resultados.

Solo cuenta con una clase que define distintas estructuras de datos. La clase solo realiza un único cometido; el envío, recepción y reconstrucción de la información. La Figura 6.3 muestra el diagrama de clases del subsistema. Se han omitido algunas funciones y parámetros de la clase `TaskDelegation` para mostrar solo lo fundamental y explicado en este apartado (el CD adjunto contiene el diagrama completo). El lector puede consultar el diagrama de clases completo y la funcionalidad detallada, tanto en el código adjunto en el CD como en la documentación técnica generada con Doxygen.

Planteamiento inicial

Antes de la implementación final se intentó construir una arquitectura de red basada en *sockets* UDP. Esta solución planteaba una mayor eficiencia que emplear *sockets* TCP debido a la ausencia de paquetes de control de errores, entre otros.

Los problemas que planteaba esta solución eran mayores que las ventajas que ofrecía, entre las que se enumeran:

- **Orden de paquetes:** El envío de paquetes mediante *sockets* UDP no garantiza que estos lleguen en orden. Esto supone un problema porque el orden de transmisión de datos en BelfegAR es crucial para mantener la correcta sincronización entre cliente

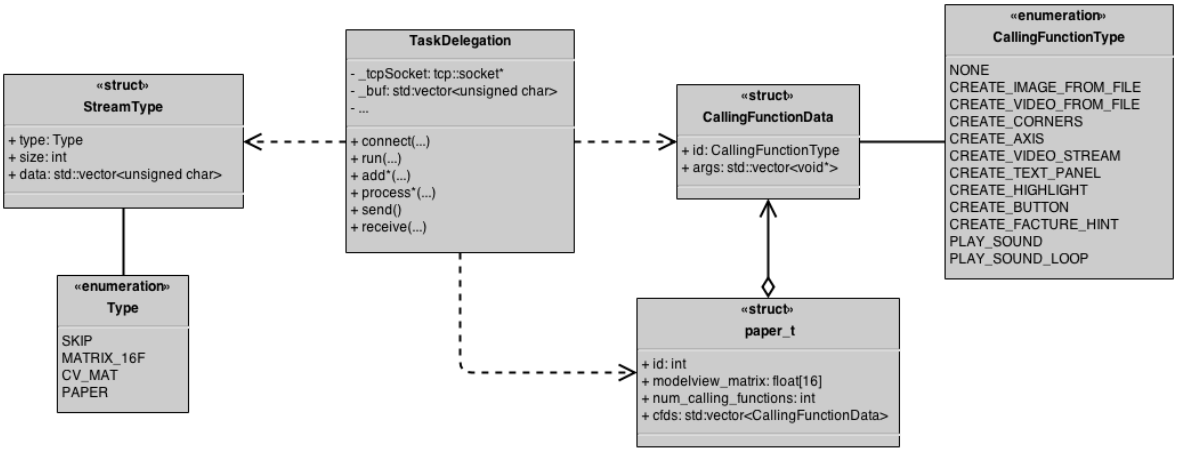

Figura 6.3: Diagrama de clases del subsistema de delegación de tareas.

y servidor. La solución a esto implicaría controlar dicho orden de paquetes mediante un nuevo campo que indique el orden del paquete, para luego poder reconstruir la secuencia original.

- **Confiabilidad:** UDP no garantiza que todos los paquetes sean recibidos por el servidor. Por tanto, por cada nueva recepción, sería necesario enviar un nuevo paquete de confirmación de llegada al origen. En caso de que el cliente no reciba dicha confirmación, supondría un paquete perdido que habría que reenviar al servidor.
- **Control de errores:** Al no ser orientado a conexión UDP no mantiene un estado de la conexión y por tanto no existe el concepto de «establecer conexión».

Como vemos, los problemas que plantea la lista anterior son salvables en mayor o menor medida. Sin embargo, TCP ya nos proporciona lo anterior, por lo que sería en vano realizar los cambios necesarios para corregir la anterior lista; no hay que caer de esta forma en optimizaciones prematuras frente a diseños funcionales y más simples.

Comunicación

Para realizar la comunicación entre el cliente y el servidor se estableció una conexión usando *sockets* TCP, proporcionados por la biblioteca *Boost.Asio*, para mantener un enlace confiable. La conexión es realizada a través de una dirección IP y un puerto proporcionados al ejecutar el servidor.

Recepción y envío de la información

A la hora de enviar información entre el cliente y el servidor, se presentó un problema de comunicación en que tanto el cliente como el servidor no conocían de antemano los datos

que estaban siendo recibidos. Esto hacía imposible la comunicación.

Por todo lo anterior, se estableció un contrato o interfaz común entre el cliente y el servidor, implementado en ambas partes, que debían de cumplir estrictamente si querían poder recibir correctamente la información emitida por la otra parte.

La Figura 6.4 muestra el convenio seguido por el cliente y el servidor para el envío de la información.

En dicha figura, el primer campo de la estructura indica el tipo de información que va a transportar. Este tipo puede ser cualquiera de los siguientes:

- `MATRIX_16F`: Representa un tipo de dato definido por un conjunto de punteros a 16 variables de tipo float o `float* matrix`.
- `CV_MAT`: Representa un tipo de dato definido por una imagen de OpenCV o `cv::Mat`.
- `PAPER`: Representa un tipo de dato personalizado correspondiente a un documento.
- `SKIP`: Se trata de un tipo especial que indica que no se ha enviado ni recibido nada, y por tanto el resto de campos de la estructura se encuentran vacíos.

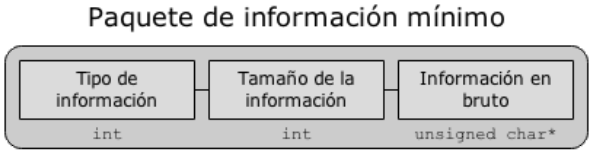

Figura 6.4: Convención de envío de paquetes entre el cliente y el servidor.

El tamaño, como su nombre indica, define el tamaño en *bytes* del campo información bruta, de tal forma que la parte receptora sepa cuántos *bytes* de información leer cuando reciba un paquete.

A la hora de realizar envíos de información, BelfegAR solo transmite información de tipo `CV_MAT`, es decir, la imagen capturada de la cámara y convertida a dicho formato. La información es enviada como un flujo de *bytes* que es reconstruida en el servidor atendiendo a los campos tipo y tamaño de la estructura presentada.

Almacenado de la información

El subsistema mantiene un *buffer* interno donde guarda la información que será enviada al servidor. La manera de interactuar con el *buffer* puede verse en el Listado 6.1.

```
1 addMatrix16f(model_matrix);
2 addMatrix16f(view_matrix);
3 addCvMat(an_image);
4 send();
```

Listado 6.1: Ejemplo de guardado de la información en el *buffer* de envío.

El *buffer* es rellenado mediante funciones del tipo `add*(...);` y cuando está listo se envía mediante la función `send();`, la cual vacía el *buffer* una vez se ha realizado el envío de la información. Este enfoque nos permite establecer dos estados para la información; de almacenaje y de envío.

Internamente, las funciones `add*(...)` invocan a funciones `memcpy(...);` para copiar la información a nivel de *bytes*. Una vez que toda la información se ha copiado a variables temporales, se puede insertar en el *buffer* de forma homogénea, siempre respetando el orden de los campos:

1. Tipo de información.
2. Tamaño de la información.
3. La propia información.

Procesado de la información recibida

Una vez realizado el envío de los datos, el cliente queda a la espera de recibir de parte del servidor la información necesaria que le indique qué hacer a continuación.

Cuando el cliente recibe información del servidor, intenta reconstruir una estructura de tipo `paper_t`, un contenedor con información crítica necesaria por BelfegAR. `paper_t` resulta de un subconjunto de una clase mayor `Paper`, competencia de GrayAR. Esta estructura viene definida por los siguientes campos:

- Un identificador del documento recibido.
- El producto de las matrices de modelo y vista de ese documento.
- El número de funciones a ejecutar para ese documento.
- Una lista con las funciones a ejecutar y sus parámetros.

La lista de funciones viene indexada por un identificador que indica la función que el sistema ejecutará. Se establece por tanto otro convenio de comunicación con las funciones que el servidor puede invocar en el cliente. Estas son:

- `DRAW_IMAGE_FROM_FILE`: Dibuja una imagen desde el disco.
- `DRAW_VIDEO_FROM_FILE`: Dibuja un vídeo desde el disco.
- `DRAW_CORNERS`: Dibuja esquinas visuales sobre el documento.
- `DRAW_AXIS`: Dibuja un eje de coordenadas.
- `INIT_VIDEO_STREAM`: Inicia una videollamada.
- `DRAW_TEXT_PANEL`: Dibuja un panel de texto.
- `DRAW_HIGHLIGHT`: Dibuja un área de resaltado.

- `DRAW_BUTTON`: Dibuja un botón.
- `DRAW_FACTURE_HINT`: Dibuja un cuadro con instrucciones para tratar el documento.
- `PLAY_SOUND`: Reproduce un sonido.
- `PLAY_SOUND_DELAYED`: Reproduce un sonido en bucle con un retraso de tiempo entre reproducción indicado.
- `NONE`: No se realiza ninguna acción.

Gestión de errores

El subsistema establece un protocolo de gestión de errores, por el cual tratará de recuperarse ante cualquier situación no controlada. Esto se consigue mediante captura de excepciones y el uso de códigos de errores.

Tanto si la conexión es cortada por el cliente, como por el servidor, cualquiera de las partes quedará a la espera hasta que la otra vuelva a estar operativa para volver a establecer la conexión.

El Listado 6.2 presenta un fragmento de la función de reconexión con el servidor y su gestión de errores.

```
1  int TaskDelegation::reconnect() {
2    try {
3      _error = 0;
4
5      Log::info("Trying to reconnect to the server.");
6      boost::asio::connect(*_tcpSocket,
7                           _tcpResolver->resolve({_ip, _port}));
8      Log::success("Reconnection succeeded.");
9    }
10   catch(boost::system::system_error const& e) {
11     Log::error("Could not reconnect to the server. "
12                + std::string(e.what()));
13     _error = -1;
14   }
15
16   return _error;
17 }
```

Listado 6.2: Función de reconexión del subsistema de delegación de tareas.

Ventajas de la implementación

Las ventajas de emplear *sockets* frente a otras soluciones fueron discutidas ampliamente en el Capítulo 4.

La implementación realizada es lo suficientemente simple y potente como para satisfacer lo más correctamente posible los requisitos necesarios; haber empleado RPC, habría supuesto mayor carga de trabajo para lograr el mismo resultado.

Subsistema de Representación Gráfica

El subsistema de representación gráfica supone el mayor reto de BelfegAR. Debe ser lo suficientemente versátil y modular como para definir nuevos componentes gráficos de forma sencilla y poder reutilizar los ya existentes para formar nuevos más complejos.

Este subsistema se encarga de la gestión de todos los componentes gráficos, es decir, el ciclo de vida de cada componente gráfico permanece ligado a este subsistema hasta la interrupción del sistema completo, donde son liberados de la memoria.

La Figura 6.5 muestra un diagrama de clases simplificado de todo el subsistema de representación gráfico. Se han omitido detalles para mantener cierta simplicidad (en el CD adjunto puede verse el diagrama completo). En próximos apartados se detallará cada una de las clases.

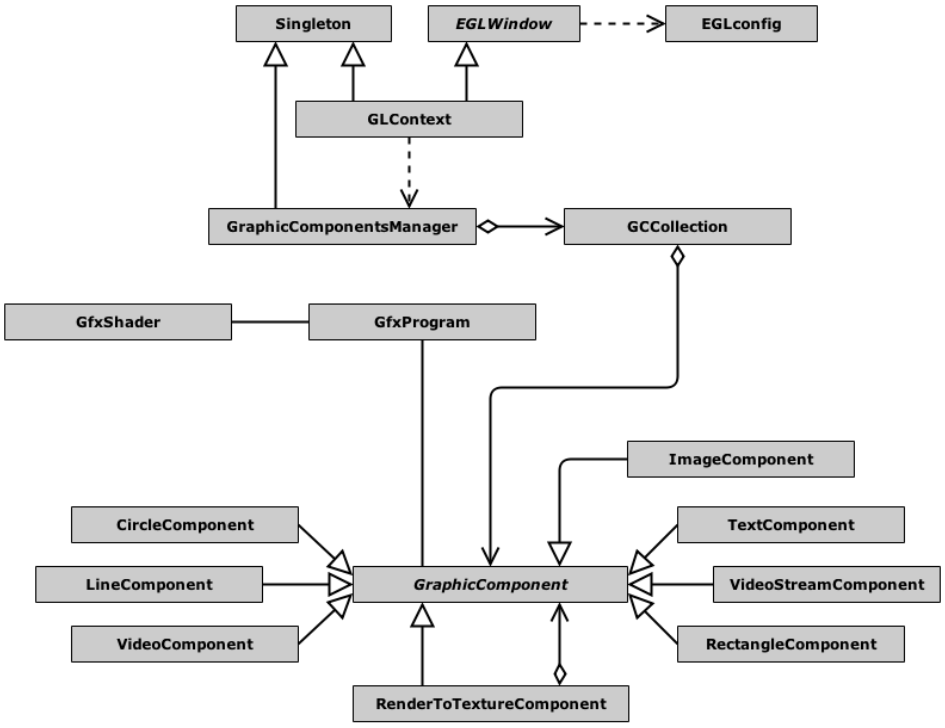

Figura 6.5: Diagrama de clases del subsistema de representación.

Prototipos iniciales

Pese a que la plataforma de despliegue no soportaba gráficos acelerados por hardware si no se empleaba OpenGL ES 2.0, se decidió realizar algunas pruebas de rendimiento utilizando la biblioteca SDL.

La prueba consistía en el dibujado de varios rectángulos y una cadena de texto indicando el número de fotogramas por segundo renderizados. Como era de esperar, el rendimiento no fue el deseado, ofreciendo únicamente 27 fotogramas por segundo.

Tras las pruebas, se inició la implementación en OpenGL ES 2.0. Dicha implementación se encontraba basada en los ejemplos del libro *OpenGL ES 2.0 Programming Guide* [MGS08]. Los ejemplos proporcionaban un marco de trabajo que proporcionaba ciertas ayudas a la hora de inicializar el contexto gráfico y de los bucles de actualización y renderizado, los cuales eran invocados mediante funciones de retrollamada, tal y como se hace en la biblioteca de ayuda OpenGL Utility Toolkit (GLUT) para OpenGL.

Finalmente, dicho marco de trabajo quedó descartado porque se prefirió mantener un control absoluto de la aplicación para poder configurarla con los parámetros deseados sin tener que modificar el código de los ejemplos del libro.

Tras realizar una prueba de rendimiento similar a la llevada a cabo con SDL se obtuvo un rendimiento de 62 fotogramas por segundo.

Inicialización y configuración

Para poder emplear gráficos acelerados por hardware usando OpenGL ES 2.0, se debe de solicitar al *hardware* de la Raspberry Pi una superficie de dibujado acelerada. Esto se realiza interactuando con la API EGL. EGL es una interfaz de comunicación entre OpenGL ES y el sistema de renderizado nativo de la plataforma.

La Figura 6.6 [Bee13] muestra las capas que forman la arquitectura de vídeo de la Raspberry Pi.

El subsistema trata de crear un contexto de OpenGL ES acelerado mediante la interacción con EGL, con las dimensiones deseadas. El sistema por defecto funciona a una resolución de 800x600, por lo que la superficie solicitada será de ese tamaño.

Finalmente, y una vez inicializado el contexto, se procede a inicializar el gestor de componentes gráficos, con los directorios donde debe buscar las imágenes, fuentes y vídeos a cargar. También se guarda en él la matriz de proyección proporcionada anteriormente por GrayAR para su aplicación a cada componente gráfico posteriormente.

Actualización y dibujado

El subsistema mantiene un bucle de actualización y renderizado de los componentes gráficos a través de su gestor.

El bucle de renderizado se encarga de dibujar todos los componentes gráficos creados y activos. Previo dibujado, se aplica a cada componente gráfico el producto de las matrices de

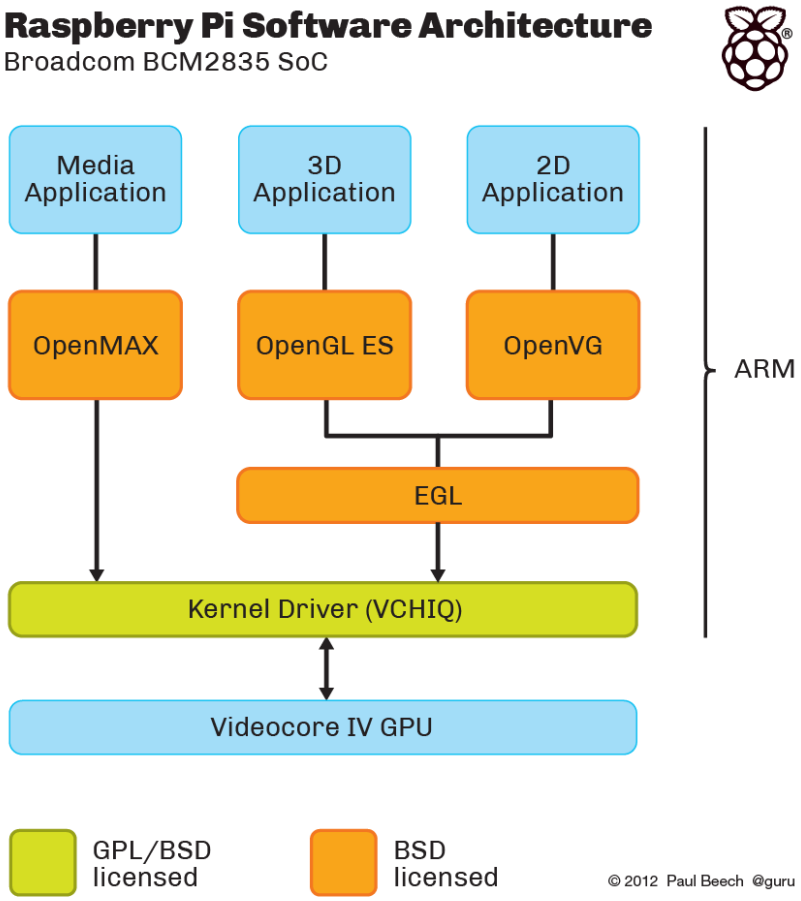

Figura 6.6: Arquitectura de software de vídeo de la Raspberry Pi.

modelo y vista recibidas a través del subsistema de delegación de tareas desde el servidor. Gracias a esto, los componentes gráficos quedarán fijos sobre el documento, de tal forma que si éste se mueve, los componentes gráficos también se moverán.

El Listado 6.3 muestra la aplicación de las matrices y el dibujado de los componentes gráficos.

```
1  void GLContext::update(paper_t& paper) {
2    // Get the ModelView matrix from the paper and apply it
3    glm::mat4 modelview_matrix = glm::make_mat4(paper.modelview_matrix);
4    GraphicComponentsManager::getInstance().update(modelview_matrix);
5  }

7  void GLContext::render() {
8    // Clears the screen
9    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
10   glViewport(0, 0, _width, _height);

12   // Render all objects
13   GraphicComponentsManager::getInstance().renderAll();
```

```
15      // To update we need to swap the buffers
16      swapBuffers();
17  }
```

Listado 6.3: Actualización y renderizado de los componentes gráficos.

De lo visto hasta ahora, la Figura 6.7 complementa las explicaciones dadas. Se han omitido algunas variables miembro y métodos, para mantener el gráfico lo más simple posible pero a la vez dando una visión descriptiva de lo que hace cada clase.

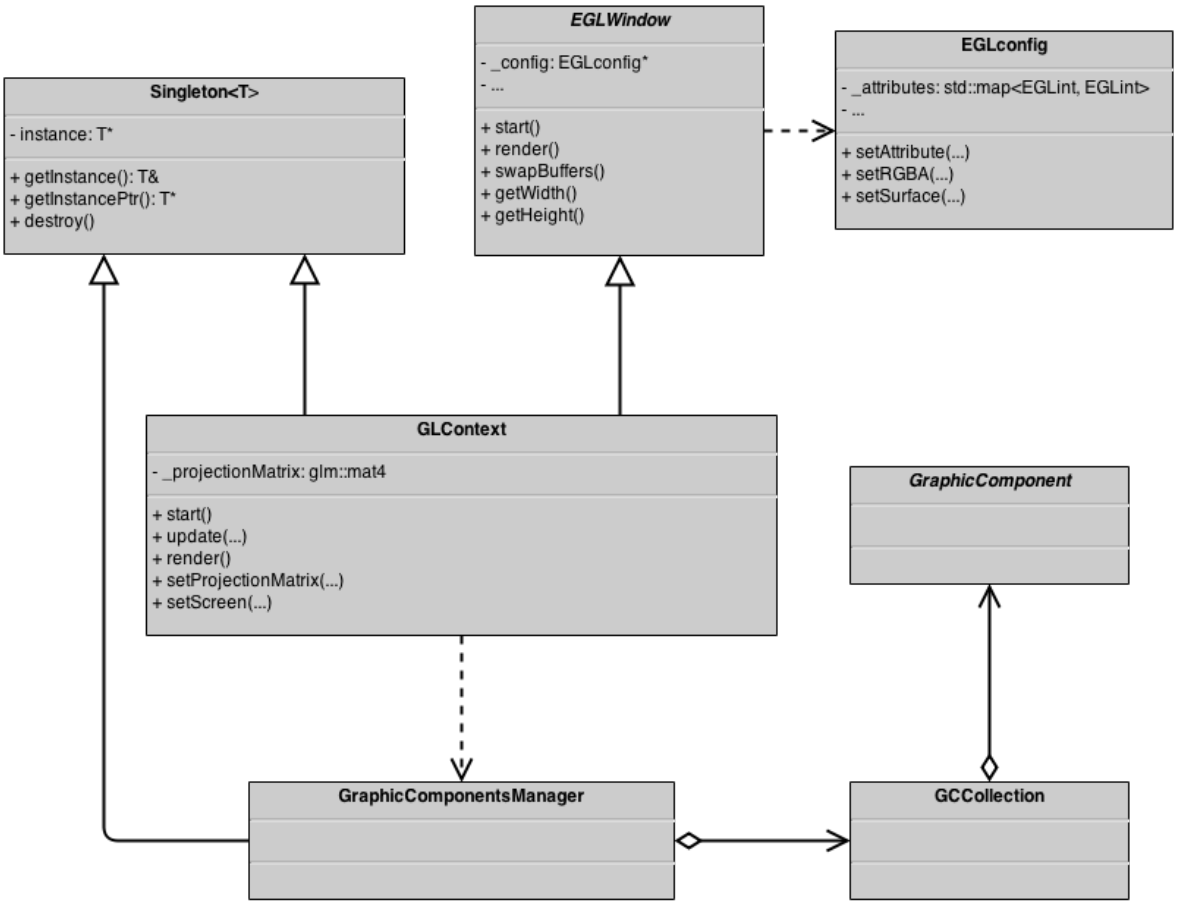

Figura 6.7: Diagrama de clases del subsistema de representación gráfica (inicialización y configuración).

En dicha figura, la clase `EGLWindow` se encargaría de inicializar el contexto de OpenGL ES lidiando con todas las operaciones de interacción con `EGL`. `GLContext` por su parte, resulta en una implementación de `EGLWindow` que básicamente se encarga de la actualización y el renderizado de los componentes gráficos.

Componentes gráficos

Los componentes gráficos representan cualquier elemento visual del sistema. Todos los componentes gráficos heredan de un componente común o padre que define las operaciones comunes de todos sus hijos. Dichas operaciones abarcan todo el abanico de transformaciones, la carga y aplicación de programas *shader* y las operaciones de dibujado generales.

La Figura 6.8 muestra el diagrama de clases detallado de la parte de componentes gráficos.

Debido al cauce o *pipeline* programable que impone OpenGL ES 2.0, todos los componentes gráficos deben de tener un programa *shader* con el que trabajar. Así, las clases `GfxShader` y `GfxProgram` se encargan de esta tarea. La primera se ocupa de cargar los archivos con el código fuente en lenguaje OpenGL Shading Language (GLSL) desde el disco, para inmediatamente compilarlos y enlazarlos. La misma clase se encarga de cargar los dos tipos de *shader* que requiere cada componente gráfico (*vertex shader* y *fragment shader*). Por su parte, `GfxShader` se encarga de crear el programa a partir de los dos *shaders* y de mantener una referencia a este para su posterior uso.

La clase `GraphicComponent` mantiene una instancia de la clase `GfxShader`, ya que cada componente gráfico requiere al menos un *shader* para funcionar, como hemos dicho antes. Entre las operaciones que proporciona la clase, se encuentran:

- `void loadGLProgram(const std::string& vShader, const std::string& fShader)`
  Crea un programa de OpenGL a partir de los dos archivos *shader* proporcionados.
- `void setColor(GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f);`
  Establece el color de este componente gráfico.
- `void rotate(GLfloat angle, glm::vec3 const & axis);`
  Rota el componente gráfico `angle` grados sobre el eje `axis`.
- `void scale(glm::vec3 const & factors);`
  Escala el componente gráfico por la proporción indicada en `factors`.
- `void translate(glm::vec3 const & translation);`
  Desplaza el componente gráfico la distancia indicada en `translation`.
- `void setModelViewMatrix(const glm::mat4& modelViewMatrix);`
  Establece la matriz de modelo y vista del componente gráfico. Dicha matriz debe de ser la propia obtenida a partir documento analizado.
- `void setProjectionMatrix(const glm::mat4& projectionMatrix);`
  Establece la matriz de proyección del componente gráfico.
- `void show(bool show = true);`
  Indica si el componente gráfico es visible o no.

- `void render();`

  Renderiza el componente gráfico.

- `void specificRender()= 0;`

  Función de renderizado específica, a implementar por cada especialización de componente gráfico que indique como debe de ser dibujado dicho componente.

- `void setUpShader()= 0;`

  Configuración del *shader* específica, a implementar por cada especialización de componente gráfico que indique con qué parámetros del *shader* tratar.

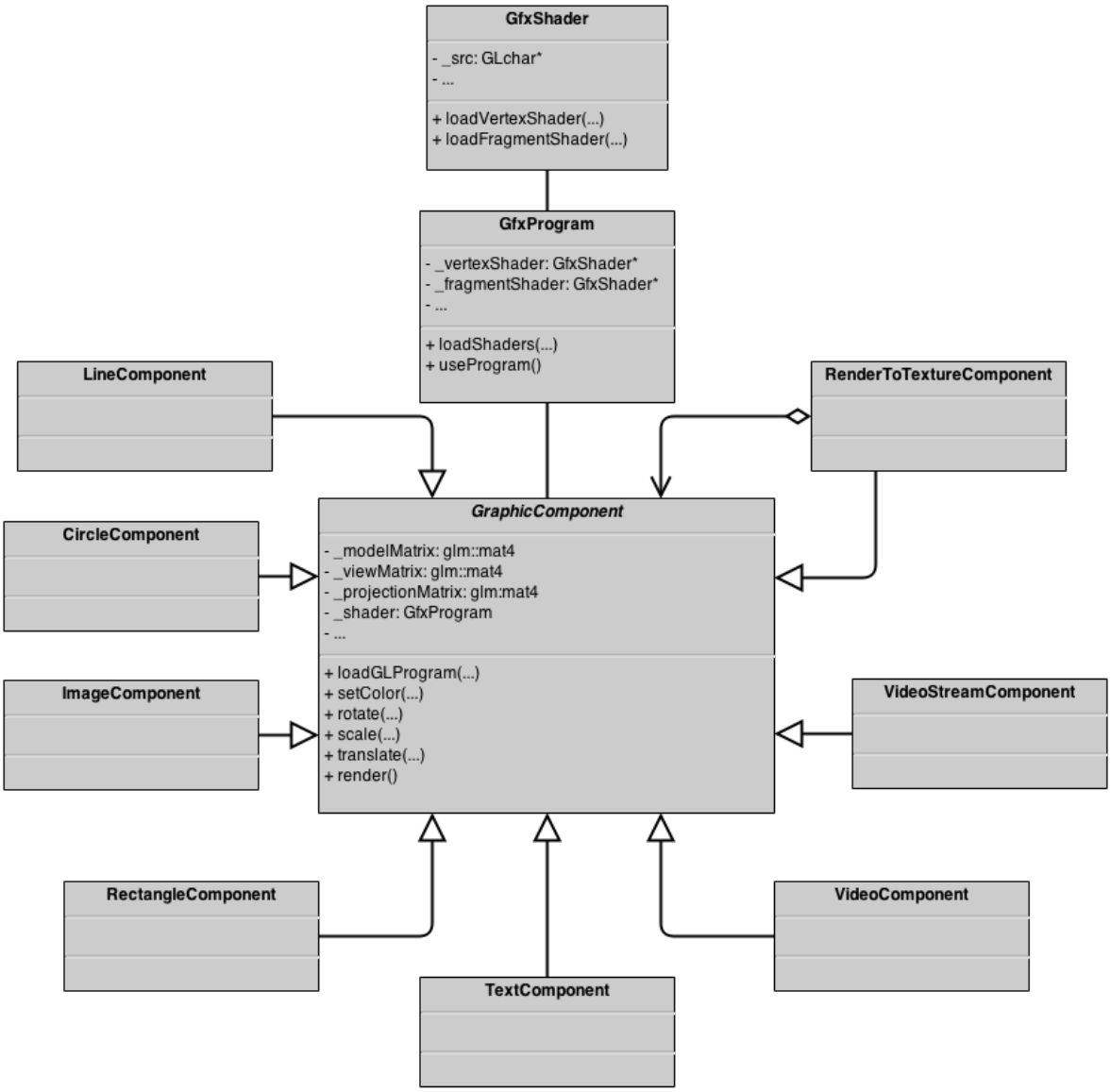

Figura 6.8: Diagrama de clases del subsistema de representación gráfica (componentes gráficos).

Cada componente gráfico presenta un comportamiento propio, de ahí que deban de ser implementados de forma separada. A continuación se indica la lista de componentes gráficos y sus propiedades más importantes:

LineComponent

Este componente se encarga de dibujar líneas rectas definidas por un punto de origen y un punto de destino. Para ser dibujado, emplea una lista de dos índices que indican las posiciones indicadas.

El *shader* empleado es el mismo que el del componente gráfico que define un rectángulo. Los únicos atributos de una línea pueden ser sus extremos y su color. El Listado 6.4 muestra el código GLSL de dicho *shader*.

```
1  // Vertex shader
2  uniform mat4 u_mvp;
3  attribute vec4 a_position;

5  void main(void) {
6    gl_Position = u_mvp * a_position;
7  }

9  // Fragment shader
10 precision mediump float;
11 uniform vec4 u_color;

13 void main(void) {
14   gl_FragColor = u_color;
15 }
```

Listado 6.4: *Shader* compartido por los componentes gráficos «línea» y «rectángulo».

RectangleComponent

Este componente dibuja un rectángulo definido por sus cuatro esquinas y un color de relleno. Cabe destacar, que para dibujar un rectángulo se deben definir los triángulos que lo componen. En el caso de este componente, se han definido dos únicos triángulos, aunque se podrían haber definido más a costa de menor rendimiento. Para definir los dos triángulos que definen el rectángulo, se emplean un total de 6 índices que indiquen el orden en que dibujar las esquinas del triángulo. El Listado 6.5 muestra esto. Los triángulos además, deben de ser indicados en sentido horario o anti-horario. Así, y en sentido horario, el rectángulo del listado quedaría definido dos triángulos cuyos vértices quedan indexados como (0, 1, 2) y (0, 2, 3).

```
1  /**
2  *    0__1
3  *    |\ |
4  *    | \|
5  *    3__2
6  */
```

```
_indices = new GLushort[6] { 0, 1, 2, 0, 2, 3 };
_vertexData = new GLfloat[20] {
  // X        Y        Z
  -_width,  _height, 0.0f, // Top-left
   _width,  _height, 0.0f, // Top-right
   _width, -_height, 0.0f, // Bottom-right
  -_width, -_height, 0.0f, // Bottom-left
};
```

Listado 6.5: Definición de los triángulos que conforman un rectángulo.

CircleComponent

Este componente gráfico se ocupa de dibujar un círculo definido por un radio y un color de relleno. El círculo en realidad es un cuadrado transparente sobre el que se aplica un *shader* para dibujar un círculo. Dicho *shader* emplea coordenadas UV para seleccionar la zona del rectángulo donde pintar. Las coordenadas UV se usan para mapear una textura de un tamaño arbitrario sobre un objeto tridimensional cuyas coordenadas de texturizado o UV se encuentran situadas en el intervalo [0, 1].

El Listado 6.6 muestra el código GLSL del *shader* que emplea este componente gráfico.

```
// Vertex shader
attribute vec4 a_position;
attribute vec4 a_texCoord;
uniform mat4 u_mvp;
varying vec2 textureCoordinate;

void main() {
  gl_Position = u_mvp * a_position;
  textureCoordinate = a_texCoord.xy;
}

// Fragment shader
varying highp vec2 textureCoordinate;
const highp vec2 center = vec2(1.0, 1.0);
const highp float radius = 1.0;
uniform vec4 u_color;

void main() {
  highp float distanceFromCenter =
    distance(center, textureCoordinate);
  lowp float checkForPresenceWithinCircle =
    step(distanceFromCenter, radius);
  gl_FragColor = u_color * checkForPresenceWithinCircle;
}
```

Listado 6.6: *Shader* usado por el componente gráfico «círculo».

TextComponent

Este componente resuelve la problemática del renderizado de texto. Para ello, hace uso de la biblioteca freetypeGlesRpi mediante la aplicación del patrón de diseño «Adapter». La biblioteca se encuentra escrita en C, por lo que se necesitaba adaptarla para que mantuviera la misma homogeneidad que el resto de componentes. `TextComponent` se encarga de ello.

La biblioteca funciona de tal forma, que crea una imagen con el conjunto de caracteres de la fuente y tamaño indicados. A esta imagen se le suele conocer como *texture-atlas*. Esto es debido a que a la hora de renderizar los textos indicados, se recorrerá esta imagen en busca de los caracteres necesarios y se extraerán para su dibujado.

Este componente soporta algunos métodos especiales, a diferencia de los anteriores:

- `void setFont(const std::string& filename, GLfloat size);`
  Establece la fuente que será usada para renderizar el texto con este componente.
- `void setText(std::wstring strtext, GLfloat r, GLfloat g, GLfloat b, GLfloat a);`
  Establece una cadena de texto a dibujar en el próximo ciclo de renderizado, sustituyendo cualquier texto que exista.
- `void addText(std::wstring strtext, GLfloat r, GLfloat g, GLfloat b, GLfloat a);`
  Hace lo mismo que el método anterior, pero sin sustituir el texto, sino añadiéndolo a continuación.

El Listado 6.7 muestra el código GLSL del *shader* que emplea este componente gráfico.

```
1  // Vertex shader
2  uniform mat4 u_mvp;
3  attribute vec3 a_position;
4  attribute vec4 a_color;
5  attribute vec2 a_st;
6  varying vec2 v_frag_uv;
7  varying vec4 v_color;
8  void main(void) {
9    v_frag_uv = a_st;
10   gl_Position = u_mvp * vec4(a_position, 1);
11   v_color = a_color;
12 }
13 // Fragment shader
14 precision mediump float;
15 uniform sampler2D texture_uniform;
16 varying vec2 v_frag_uv;
17 varying vec4 v_color;
18 void main(void) {
19   gl_FragColor = vec4(v_color.xyz,
20     v_color.a * texture2D(texture_uniform, v_frag_uv).a); }
```

Listado 6.7: *Shader* usado por el componente gráfico «texto».

ImageComponent

Este componente se encarga del dibujado de imágenes. Una imagen viene definida a través de una primitiva de tipo rectángulo con unas coordenadas UV específicas. Este componente se apoya en el uso de la biblioteca SOIL, la cual permite cargar una imagen desde disco y obtener un puntero a la zona de la memoria donde ha quedado guardada. Posteriormente, dicho puntero es utilizado durante la creación de una textura de OpenGL, para convertir la imagen a un formato utilizable.

Este método proporciona también algunos métodos propios:

- `void loadImageFromFile(const std::string& file_name);`
  Permite cargar una imagen desde disco, a partir de su nombre de archivo.
- `void loadImageFromMat(cv::Mat& mat);`
  Permite cargar una imagen procedente de un `Mat` de OpenCV, en una textura propia de OpenGL. Esto es útil para poder aplicar transformaciones geométricas a la imagen, que de otra forma no se podrían aplicar a través de OpenCV.

El Listado 6.8 muestra la implementación del método `loadImageFromFile` y el uso de la biblioteca SOIL.

```
1  void ImageComponent::loadImageFromFile(
2      const std::string& file_name) {
3    int width, height, channels;
4
5    unsigned char* buffer =
6      SOIL_load_image(file_name.c_str(), &width, &height,
7                      &channels, SOIL_LOAD_AUTO);
8    if(buffer == NULL) {
9      Log::error("Image '" + file_name + "' loaded incorrectly");
10     Log::error(std::string(SOIL_last_result()));
11     exit(1);
12   }
13   else {
14     Log::success("Image '" + file_name + "' ("
15                  + std::to_string(width) + "x"
16                  + std::to_string(height)
17                  + ") successfully loaded");
18   }
19
20   GLenum format;
21   switch(channels) {
22     case 3:
23     format = GL_RGB;
24     break;
25     case 4:
26     format = GL_RGBA;
27     break;
28   }
29
30   // Use tightly packed data
31   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
```

```
33    // Generate a texture object
34    glGenTextures(1, &_textureId);

36    // Bind the texture object
37    glBindTexture(GL_TEXTURE_2D, _textureId);

39    // Set the filtering mode
40    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
41                    GL_CLAMP_TO_EDGE);
42    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
43                    GL_CLAMP_TO_EDGE);
44    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
45                    GL_LINEAR);
46    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
47                    GL_LINEAR);

49    // Create the texture
50    glTexImage2D(GL_TEXTURE_2D, 0, format, width, height, 0,
51                 format, GL_UNSIGNED_BYTE, buffer);

53    // Free the image data
54    SOIL_free_image_data(buffer);
55  }
```

Listado 6.8: Carga de imágenes desde archivo a textura de OpenGL.

El Listado 6.9 muestra el código GLSL del *shader* que emplea este componente gráfico.

```
1   // Vertex shader
2   uniform mat4 u_mvp;
3   attribute vec4 a_position;
4   attribute vec2 a_texCoord;
5   varying vec2 v_texCoord;

7   void main(void) {
8     gl_Position = u_mvp * a_position;
9     v_texCoord = a_texCoord;
10  }

12  // Fragment shader
13  precision mediump float;
14  uniform sampler2D s_texture;
15  varying vec2 v_texCoord;

17  void main(void) {
18    gl_FragColor = texture2D(s_texture, v_texCoord);
19  }
```

Listado 6.9: *Shader* usado por el componente gráfico «imagen».

VideoComponent

Este componente se ocupa de la carga y reproducción de archivos de vídeo. Presenta un comportamiento similar al del anterior componente, ya que un vídeo en realidad es únicamente una secuencia de imágenes. Para la carga de vídeos, no se ha querido añadir otra dependencia al proyecto y se ha optado por reutilizar la biblioteca OpenCV.

Los métodos propios de este componente son:

- `void loadVideoFromFile(const std::string& fileName);`
  Permite cargar un archivo de vídeo desde el disco. Existen ciertas limitaciones a la hora de cargar vídeos, ya que depende de los *codecs* instalados en el sistema, de los cuales hace uso OpenCV.
- `void setLoop(bool loop);`
  Establece si el vídeo debería comenzar de nuevo al terminar o no.

El *shader* utilizado por este componente es el mismo que el empleado por el componente de «imagen» (véase Listado 6.9).

VideoStreamComponent

Este componente se encarga de iniciar y mantener una videollamada con el servidor. Cuando se inicia una videollamada, el servidor abre una ventana con las imágenes que recibe del cliente, para que el operario pueda visualizar el documento con el que está trabajando el cliente. Del mismo modo, el cliente recibirá una nueva imagen de una cámara USB instalada en el servidor, que será mostrada a través del proyector para su visualización.

La videollamada se realiza sobre otro hilo de ejecución, con el propósito de no sobrecargar el hilo principal del cliente. Cuando la videollamada se inicia, se lanza dicho hilo de ejecución que lo que hace es quedar a la espera de recibir las nuevas imágenes del servidor mediante un *socket* UDP. Si no se reciben imágenes durante más de 1 segundo, la videollamada queda cancelada.

OpenGL no permite la creación de nuevas texturas desde un hilo distinto al que creó el contexto de renderizado. Por esto, la creación de la textura se deriva en el hilo principal a partir del fotograma recibido en el hilo. Sin embargo, esta solución es propensa a condiciones de carrera e interbloqueos, ya que el fotograma de vídeo es accedido desde dos hilos distintos y en tiempos no controlados. Para solventar esto, se emplean semáforos que controlen el acceso a dicho semáforo desde los hilos.

El componente solo dispone de un método público propio:

- `void startReceivingVideo(unsigned short port);`
  Inicia el hilo de recepción de vídeo del servidor, en el puerto indicado.

El *shader* utilizado por este componente es el mismo que el empleado por el componente de «imagen» y de «vídeo» (véase Listado 6.9).

RenderToTextureComponent

Este componente permite agrupar varios componentes gráficos, que serán dibujados y transformados como uno solo. Los componentes agrupados son dibujados empleando una proyección ortográfica que establece una equivalencia de medida entre píxeles y

unidades del mundo. Gracias a esto, se pueden construir nuevas imágenes convertidas a texturas de OpenGL y aplicadas en nuestro caso sobre una figura rectangular de dimensiones indicadas. Este componente es empleado para la creación de algunos *widget*, como botones o paneles de ayuda del documento.

La técnica empleada por este componente gráfico se denomina «render a textura», porque en vez de renderizar hacia el dispositivo de salida, se renderiza hacia una textura para su posterior utilización. La implementación se ha realizado mediante FrameBuffer Object (FBO)s de tal forma que se emplea un *framebuffer* ligado a una textura de OpenGL, sobre el que se dibuja la escena. Posteriormente, se emplea la textura nombrada con el contenido de la escena para dibujar el nuevo objeto.

Este componente define dos métodos públicos y propios:

- `void addGraphicComponent(GraphicComponent* graphicComponent);`
  Añade un componente gráfico para ser usado en el proceso de «render a textura». Dicho componente gráfico no debe de llevar aplicada ninguna matriz de transformación obtenida de los documentos.
- `GraphicComponent* getGraphicComponent(int index);`
  Obtiene el componente gráfico por su índice en la lista.

El *shader* utilizado por este componente es el mismo que el empleado por el componente de «imagen», de «vídeo» y de «videollamada» (véase Listado 6.9).

Gestión y composición de componentes gráficos

Pese a que la solución proporcionada por el `RenderToTextureComponent` facilita la tarea de componer componentes gráficos, hay situaciones que requieren que cada componente gráfico mantenga su propia identidad, es decir, sus propias matrices de transformación procedentes de los documentos. Para esto, se ha creado una clase contenedora que gestiona de forma conjunta los componentes gráficos que contenga.

Además, también se necesitaba una manera centralizada de gestionar todos los componentes gráficos y de crear otros más complejos. Para ello, se ha implementado el patrón `Singleton` en la clase `GraphicComponentsManager`. El patrón `Singleton` garantiza que solo exista una única instancia de la clase y un acceso global a ésta.

La Figura 6.9 muestra la idea general de lo explicado en este apartado.

La clase `GraphicComponentsManager` se ocupa de gestionar todos los componentes gráficos del subsistema. Además, también permite la creación de otros más complejos mediante una implementación similar a la del patrón «Builder»; se emplea la técnica de encadenamiento de métodos no para crear el objeto, sino para configurarlo después de su creación, de ahí que no sea una implementación fiel a la original.

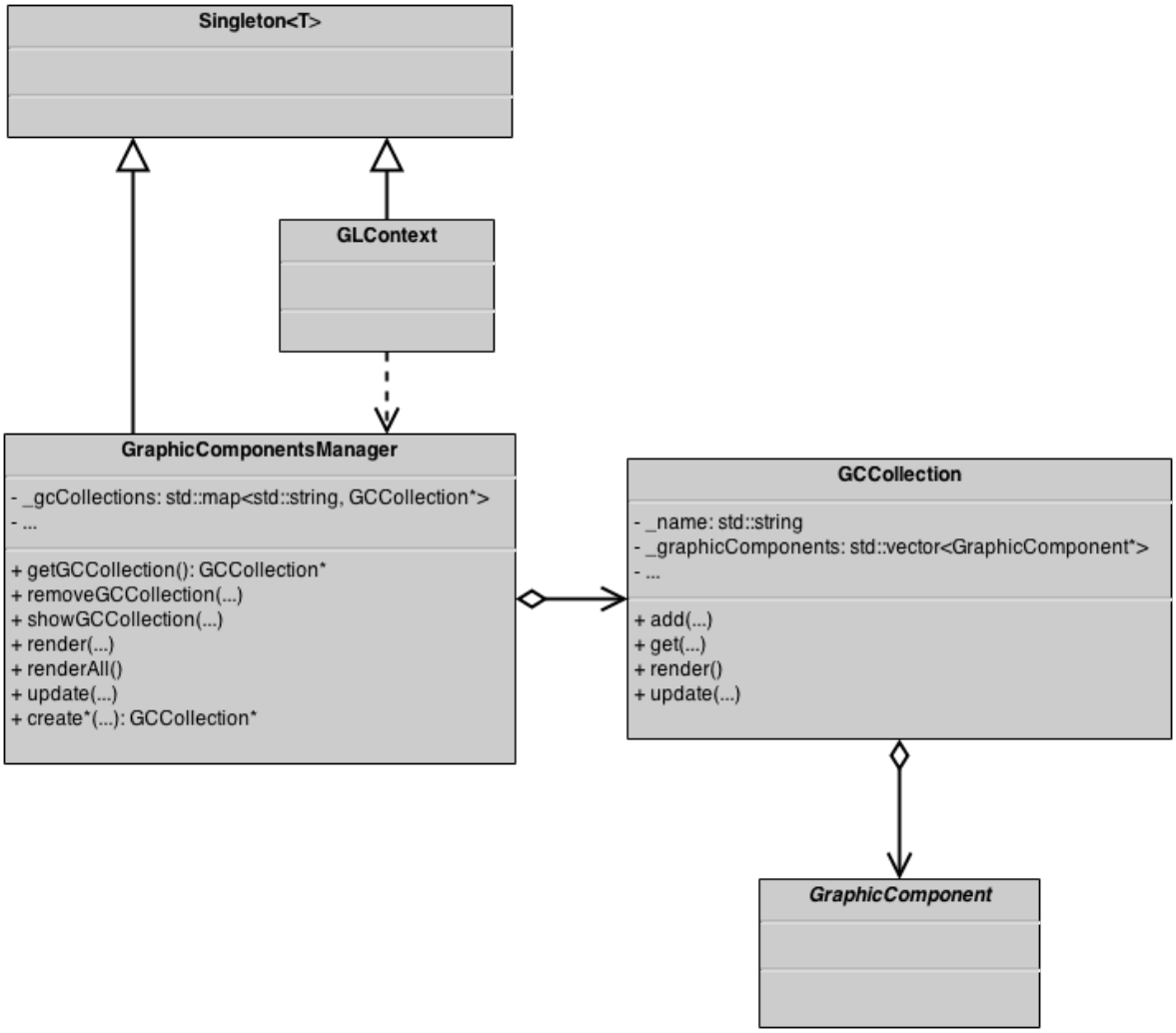

Figura 6.9: Diagrama de clases del subsistema de representación gráfica (gestión y composición de componentes gráficos).

La clase es configurable con las rutas de los directorios donde debe buscar los recursos de imagen, vídeo y fuentes, mediante inyección de dependencias a través de métodos *setters*. La clase gestora no controla componentes gráficos individuales, sino como se introdujo al principio de la sección, colecciones de componentes. Dichas colecciones se encuentran almacenadas en un contenedor asociativo o `std::map` indexados por su nombre, para su fácil recuperación y acceso. Estas colecciones se guardan en dicho contenedor mediante *punteros inteligentes*. Los punteros inteligentes explotan el *idiom* RAII, por el cual el ciclo de vida de los objetos es gestionado automáticamente.

Los punteros inteligentes permiten la gestión automática de la memoria reservada durante la creación dinámica de objetos. Esto se consigue, manteniendo un conteo de referencias junto al puntero original, de tal forma que cuando un puntero pierde posesión del objeto, el contador de referencias disminuye, y viceversa. Cuando dicho contador llega a 0, el objeto es destruido y la memoria liberada, de acuerdo a su destructor.

Las colecciones de componentes vienen definidas por la clase `GCCollection`. Fundamentalmente, permite añadir y recuperar componentes gráficos a una colección para tratarlos como si fueran un único componente. Los componentes se almacenan en una lista mediante un índice numérico y al igual que antes, como punteros inteligentes.

Ventajas de la implementación

El subsistema de representación gráfica supone una implementación modular y escalable. Nuevos componentes gráficos pueden ser añadidos en el futuro, simplemente heredando de la clase `GraphicComponent` e implementando los métodos oportunos.

Los componentes gráficos además son independientes entre si, por lo que el acoplamiento entre ellos es inexistente.

Por último, el cargar los programas *shaders* desde el disco, supone su posterior modificación por parte de los usuarios, para lograr nuevos efectos aplicados a componentes gráficos ya existentes.

Subsistema de Audio

El subsistema de audio permite la gestión de los archivos de sonido. Se recomienda ver el Anexo E, el cual explica como crear archivos de audio de texto hablado.

El subsistema consta de una única clase que implementa el patrón *Singleton*. De esta forma, la gestión de audio queda centralizada y accesible desde cualquier clase. La Figura 6.10 muestra dicho diagrama de clases.

Planteamiento inicial

El subsistema de audio fue una de las partes del proyecto que menos tiempo tardó en implementarse. Esto fue debido a que ya se conocía en profundidad la API de `SDL_Mixer` y fue por eso por lo que se planteó su uso desde un principio.

Pese a lo anterior, también se planteó el uso de la API `OpenAL`. Sin embargo, quedó descartado su uso, debido a la complejidad asociada a la hora de realizar algunas tareas básicas, como simplemente cargar un archivo de audio `.wav` y reproducirlo. Tampoco resuelve la problemática de la decodificación de formatos de audio; depende del programador cargar el archivo, descomprimirlo y enviarlo a `OpenAL`.

`SDL_Mixer` sin embargo resolvía todos los problemas anteriores de la manera más limpia y simple posible.

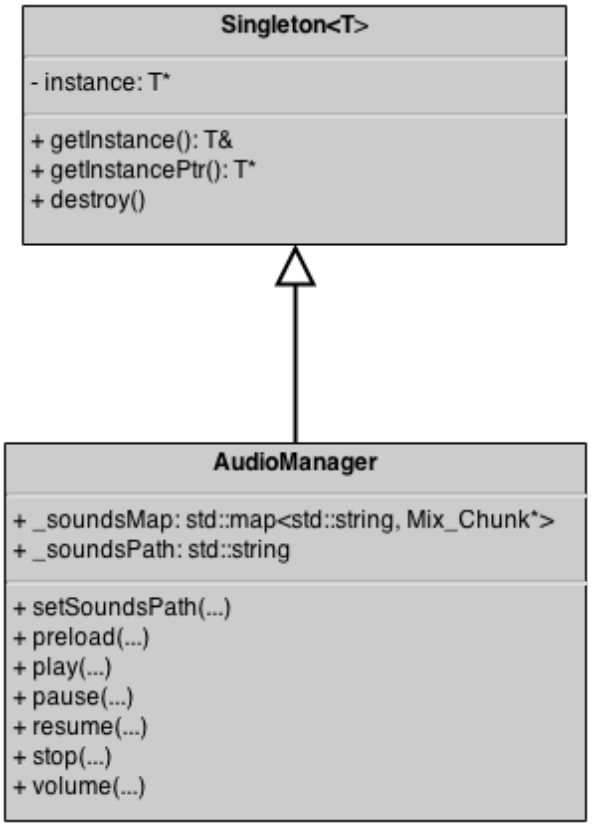

Figura 6.10: Diagrama de clases del subsistema de audio.

Configuración

La única configuración que requiere el subsistema de audio se realiza en el propio constructor, invocado automáticamente al obtener la referencia al gestor por primera vez. El constructor se encarga de inicializar el subsistema de audio de la biblioteca SDL y el dispositivo de audio por parte de `SDL_Mixer`.

Manualmente, la única configuración que se requiere es que se indique la ruta de directorios desde donde se van a cargar los sonidos.

Carga de archivos

La carga de sonidos se puede realizar de forma manual o automática. La forma manual, permite precargar los sonidos antes de reproducirlos para que de esta forma, el rendimiento del sistema no se vea afectado. Si por el contrario optamos por la carga automática, a la hora de reproducir un archivo primero se comprueba que éste exista. Si no existe, se carga inmediatamente y a continuación se reproduce. El Listado 6.10 muestra este comportamiento.

```
void AudioManager::play(const std::string& file_name) {
  if(_soundsMap.find(file_name) != _soundsMap.end()) {
    Mix_PlayChannel(-1, _soundsMap[file_name], 0);
  }
  else {
    preload(file_name);
    play(file_name);
  }
}
```

Listado 6.10: Carga de sonido no precargado al reproducir archivo.

El subsistema también permite cargar todos los archivos del directorio de una sola vez. El Listado 6.11 muestra el método que se encarga de esto. El método recupera la lista de archivos del directorio donde están ubicados los sonidos y los carga uno a uno.

```
void AudioManager::preloadAll() {
  DIR *dir;
  dirent *ent;

  if((dir = opendir(_soundsPath.c_str())) != NULL) {
    while((ent = readdir(dir)) != NULL) {
      if((strcmp(ent->d_name, ".") != 0)
         && (strcmp(ent->d_name, "..") != 0))
        preload(ent->d_name);
    }
    closedir(dir);
  }
  else {
    Log::error("There was an error loading the sounds from '"
               + _soundsPath + "'. Aborting");
    SDL_Quit();
  }
}
```

Listado 6.11: Precarga de todos los archivos del directorio de sonidos.

Gestión de sonidos

La gestión de los sonidos se realiza manteniendo un contenedor asociativo (`std::map`) que mantenga la lista de sonidos cargados indexados por su nombre de archivo. De esta forma, las operaciones que se realizan sobre los sonidos son fácilmente realizables al poder referirse a ellos directamente por su nombre.

Algunas de las operaciones que se pueden realizar sobre los sonidos, son reproducirlos, pausarlos, controlar su volumen (tanto el individual como el global del sistema), o detenerlos completamente.

Ventajas de la implementación

El uso de un gestor de audio independiente del resto de subsistemas, supone ciertas ventajas, basadas principalmente en la portabilidad del subsistema. Además, el uso de `SDL_Mixer` para la carga, descompresión y reproducción de audio resulta en una manera muy simple de tratar con los sonidos de la aplicación.

El subsistema también evita la carga de sonidos duplicados, comprobando la existencia de éste a partir de su nombre antes de cargarlo. No se ha implementado una comprobación de *hash*, ya que sobrecargaría la aplicación, ya que hay que recordar que el subsistema se ejecuta en el cliente.

6.1.3 Servidor

Al igual que antes, se explicarán los subsistemas que integran el servidor mediante el enfoque ya planteado.

Subsistema de Comunicación

El subsistema de comunicación debe de mantener una conexión activa con el subsistema de delegación de tareas del cliente y esperar continuamente por información procedente de este. Además, también debe de proporcionarle la información necesaria al cliente, fruto del procesado de GrayAR al analizar documentos y del subsistema de *scripts*.

El corazón del subsistema de comunicaciones es la clase `Communicator`. En este caso, se omite el diagrama de clases por ser prácticamente idéntico al de la clase `TaskDelegation`. Esto se explicará en próximos párrafos.

Planteamiento inicial

Para el subsistema de comunicación se plantearon las mismas ideas que para el subsistema de delegación de tareas, las cuales fueron descartadas por una solución basada en *sockets* TCP, como hemos dicho.

Recibiendo información del cliente

La información recibida del cliente es simplemente el fotograma capturado por la cámara en ese preciso instante. Una vez recibido el paquete de información con el fotograma, se valida comprobando sus cabeceras tipo y tamaño, y se procede a su procesado por GrayAR.

El Listado 6.12 muestra la implementación de la recepción de información. Se han omitido algunas comprobaciones de errores para mantener el código lo más simple posible para su comprensión.

```
1  void Communicator::readStreamTypeFromSocket(tcp::socket &socket,
2                                              StreamType &st) {
3    unsigned char type_buf[sizeof(int)];
4    unsigned char size_buf[sizeof(int)];
5    unsigned char* data_buf;
6
7    // Type
8    boost::asio::read(socket,
9                      boost::asio::buffer(&type_buf, sizeof(int)));
10   memcpy(&st.type, &type_buf, sizeof(int));
11
12   // Size
13   boost::asio::read(socket,
14                     boost::asio::buffer(&size_buf, sizeof(int)));
15   memcpy(&st.size, &size_buf, sizeof(int));
16
17   // Data
18   data_buf = new unsigned char[st.size];
19   boost::asio::read(socket,
20                     boost::asio::buffer(data_buf, st.size));
21   st.data.insert(st.data.end(), &data_buf[0], &data_buf[st.size]);
22   delete [] data_buf;
23 }
24
25 void Communicator::receive() {
26   while(1) {
27     try {
28       StreamType st;
29       readStreamTypeFromSocket(*_tcpSocket, st);
30
31       switch(st.type) {
32         case Type::CV_MAT:
33           processCvMat(st);
34           _receive = true;
35           _condReceive.notify_one();
36           break;
37         default:
38           break;
39       }
40
41       if(!_buff.empty()) {
42         send();
43         _buff.clear();
44       }
45     }
46     catch(boost::system::system_error const& e) {
47       Log::error("Connection lost with the client. "
48                  + std::string(e.what()));
49       _tcpSocket->close();
50       break;
51     }
52   }
53 }
```

Listado 6.12: Recepción de la información por el subsistema de comunicación.

Una vez recibido el fotograma y procesado, si existía algún documento en este que generara algún tipo de información como matrices de transformación o acciones invocadas por un *script*, se procede al envío de dicha información de vuelta al cliente.

Concurrencia en videollamadas

A la clase `Communicator` le corresponde además la responsabilidad de iniciar el envío de fotogramas al cliente cuando ocurre una videollamada. En ese caso, inicia un nuevo hilo de ejecución, que abrirá un *socket* UDP sobre el que se transmitirán los fotogramas de vídeos recogidos por la *webcam* del servidor. Además, también mostrará al usuario los fotogramas recibidos del cliente.

El Listado 6.13 muestra la creación de dicho hilo de ejecución y el proceso de envío de fotogramas. De nuevo, se han omitido partes de la implementación para mantener cierta simplicidad.

```
1  ...
2
3  if(initVideoConference) {
4    if(_threadDone) {
5      _threadDone = false;
6      _videoThread =
7        std::make_shared<std::thread>(&Communicator::startVideoStream,
8                                      this);
9      _videoThread->detach();
10   }
11 }
12
13 ...
14
15 void Communicator::startVideoStream() {
16   const std::string ip =
17     _tcpSocket->remote_endpoint().address().to_string();
18
19   const std::string port("9999");
20
21   const string videoStream = "Video Stream";
22
23   boost::asio::io_service ioService;
24   udp::socket udpSocket(ioService, udp::endpoint(udp::v4(), 0));
25   udp::resolver udpResolver(ioService);
26   udp::resolver::query query(udp::v4(), ip, port);
27   udp::endpoint udpEndpoint = *udpResolver.resolve(query);
28
29   cv::VideoCapture cam(0);
30   cam.set(CV_CAP_PROP_FORMAT, CV_8UC1);
31   cam.set(CV_CAP_PROP_FRAME_WIDTH, 320);
32   cam.set(CV_CAP_PROP_FRAME_HEIGHT, 240);
33
34   cv::Mat frame;
```

```
36    while(initVideoConference) {
37      _receive = false;

39      std::unique_lock<std::mutex> lock(_mutex);
40      while(!_receive) {
41        _condReceive.wait(lock);
42      }

44      imshow(videoStream, currentFrame);

46      if(!cam.grab()) continue;
47      if(!cam.retrieve(frame) || frame.empty()) continue;
48      if(cv::waitKey(30) >= 0) break;

50      cv::flip(frame, frame, 0);
51      cv::cvtColor(frame, frame, cv::COLOR_BGR2RGB);
52      sendCvMatVideoStream(frame, udpSocket, udpEndpoint);
53    }

55    destroyWindow(videoStream);
56    _threadDone = true;
57  }
```

Listado 6.13: Inicio del hilo de transmisión de vídeo y proceso de envío.

El envío de vídeo se encuentra bloqueado hasta que no se recibe un nuevo fotograma de parte del cliente. De esta forma, se establece una comunicación «ping-pong» en que una parte no envía vídeo hasta que la otra recibe.

Manteniendo el contrato

Como este subsistema se comunica directamente con el subsistema de delegación de tareas, el contrato de envío de información debía mantenerse, por lo que las distintas enumeraciones y estructuras de datos quedan replicadas también en el servidor.

Ventajas de la implementación

La arquitectura de red la conforman este subsistema y el de delegación de tareas, por tanto se aplican las mismas ventajas para este subsistema que para el segundo.

Subsistema de Scripts

El subsistema de *scripts* conforma otra parte fundamental de BelfegAR. Es principalmente utilizado para indicar acciones a realizar en los documentos indicados. Para cada documento registrado previamente en el sistema, se le asigna un archivo ARgos Script (ARS) mediante una especificación XML definida por GrayAR. Los *scripts* permanecen en memoria sin activar hasta que se identifique un documento que tenga algún *script* registrado, momento por el cual se activa. La Figura 6.11 muestra el diagrama de clases de este subsistema.

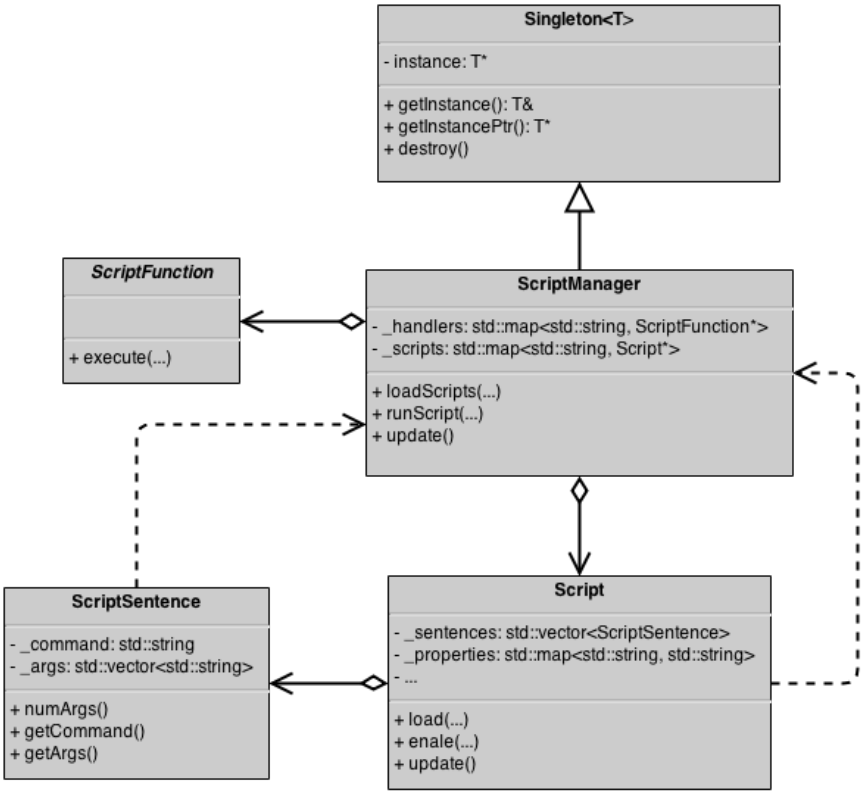

Figura 6.11: Diagrama de clases del subsistema de *scripts*.

Especificación del lenguaje

El lenguaje de *scripts* ARS se basa principalmente en los archivos en formato **OBJ!**, aunque con matices muy distintas. Mientras que el primero es simplemente un contenedor de información de vértices y demás atributos de objetos 3D, ARS es un contenedor de instrucciones a ejecutar secuencialmente.

Las instrucciones son llamadas «sentencias» y solo se permite una por línea. Las sentencias siguen una sintaxis tal que:

```
<carácter_de_control> <función> <arg_1> <arg_2> <arg_n> ...
```

Donde cada *token* o elemento de la sentencia significa:

- **<caracter de control>:** Puede ser uno de los siguientes:
  - # Indica que esta sentencia será ignorada (no se ejecutará).
  - > La sentencia precedida por este carácter será ejecutada.
  - ! Una vez encontrado este carácter, el resto del *script* será ignorado (no se ejecutará).
- **<función>:** Se trata del nombre de la función a invocar.
- **<arg_1><arg_2><arg_n>...:** Los argumentos de la función anterior.

El Listado 6.14 muestra un ejemplo de un *script* que dibuja un panel de texto, reproduce un audio y comprueba una región del documento.

```
# Draw a red (1,0,0) text panel at (0,0,0)
> DrawTextPanel 1 0 0 "Sample text" 0 0 0

# Play a sound file
> PlaySound sample_sound.wav 0

# Check the document within the squared region (1,18,5,2)
> CheckRegion 1 18 5 2

# Whatever is under the !, won't be executed
!
> PlaySound another_sound.wav 0
```

Listado 6.14: Ejemplo de *script* ARS.

Arquitectura

La arquitectura del subsistema de *scripts* se basa principalmente en un diseño de agregación, donde cada clase tiene bien definida la tarea para la que está hecha.

Partiendo desde abajo, un *script* estaría formado por varias sentencias o instancias de la clase `ScriptSentence`. Esta clase se encarga de interpretar líneas de código del *script*, dividirlas en los *tokens* anteriormente vistos y, cuando llegue el momento, ejecutarlas.

El Listado 6.15 muestra el procesamiento de los *tokens* de una sentencia del *script*. Después de procesar los *tokens*, obtenemos por una parte el nombre de la función, y por otra, su lista de argumentos.

```
void ScriptSentence::decouple(const std::string& scriptSentence) {
  std::string token;
  std::istringstream parser(scriptSentence);
  parser.ignore(1); // Ignore '>' character

  parser >> std::ws;
  std::getline(parser, _command, ' '); // Command name

  parser >> std::ws;
  while(std::getline(parser, token, ' ')) {
    _args.push_back(token.c_str()); // Arguments
    parser >> std::ws;
  }
}
```

Listado 6.15: Método procesador de los *tokens* de una sentencia.

Si subimos un nivel de abstracción, llegamos a la clase `Script`. Esta clase, representaría un archivo de *script* en memoria. Posee una lista de sentencias y otra lista de propiedades, entre las que se pueden encontrar el nombre del *script*.

La clase `Script` permite cargar, activar y actualizar un *script*. Durante la carga del *script* desde el disco, se analizan una a una las líneas del archivo y se crean nuevas instancias de la clase `ScriptSentence` que serán guardadas en la lista de sentencias. En esta parte se analizan los caracteres de control para actuar de forma apropiada ante cada uno de ellos. El Listado 6.16 muestra la carga de archivos ARS desde disco.

```
1  int Script::load(const std::string& path) {
2    int num_lines = 0;
3
4    std::ifstream file;
5    try {
6      file.open(path);
7
8      if(!file)
9        throw std::ios::failure("Error opening the script file: "
10                               + path);
11
12     std::string line;
13     while(std::getline(file, line)) {
14       ++num_lines;
15
16       switch(line[0]) {
17         case '!':
18           // Stop parsing
19           return num_lines;
20         case '#':
21           // Comment - Ignore it
22           break;
23         case '>':
24           // Command - Process it
25           _sentences.push_back(ScriptSentence(line));
26         break;
27       }
28     }
29   } catch(const std::exception& e) {
30     Log::error(e.what());
31   }
32
33   return num_lines;
34 }
```

Listado 6.16: Carga de sentencias de un *script*.

Una vez cargadas las sentencias de un *script* en memoria y previa activación del *script*, este se actualiza continuamente, leyendo y ejecutando una a una las sentencias. Esto no se hace inmediatamente; por cada ciclo de la aplicación se ejecuta una sentencia del *script*, para no bloquear el resto de la aplicación mientras se ejecuta el *script*. El Listado 6.17 muestra el método de ejecución del *script*. Se puede observar además que la ejecución del *script* puede ser detenida durante un periodo de tiempo indicado, a través de una función *scripteable*, las cuales serán introducidas en el apartado siguiente.

```
void Script::update() {
  if(!_enabled)
    return;

  if(_secondsToWait != 0.0f) {
    if(_timer.getMilliseconds() >= (_secondsToWait * 1000)) {
      _secondsToWait = 0.0f;
      runOneSentence();
      return;
    }
  }

  runOneSentence();
}
```

Listado 6.17: Ejecución del *script*.

Por último y subiendo un nivel más de abstracción, se encuentra la clase `ScriptManager`. Esta clase se encarga de toda la gestión relativa a los *script*; desde su carga hasta su ejecución. La clase mantiene una lista de *scripts* cargados indexados por su nombre. Esto facilita el acceso y ejecución desde otras partes del programa. El Listado 6.18 muestra la carga de todos los *scripts* que estén asociados a un documento. La lista de *scripts* cargables, es devuelta por el gestor de configuración, competencia de GrayAR.

```
void ScriptManager::loadScripts(const std::string& path) {
  if(!_scripts.empty())
    _scripts.clear();

  const std::vector<std::string>& script_file_names =
    ConfigManager::getScriptsList();

  for(auto& script_file_name : script_file_names) {
    Log::info("Loading script '" + script_file_name
              + "' from '" + path + script_file_name + "'");

    Script* script = new Script(*this);
    script->setProperty("filename", script_file_name);
    int num_lines = script->load(path + script_file_name);

    if(num_lines > 0)
      Log::success("Loaded " + std::to_string(num_lines)
                   + " sentences from script '"
                   + script_file_name + "'");
    else
      Log::error("Loaded " + std::to_string(num_lines)
                 + " sentences from script '"
                 + script_file_name + "'");

    _scripts[script_file_name] = script;
  }
}
```

Listado 6.18: Carga de *scripts* mediante el gestor.

Aparte de lo anterior, el gestor de *scripts* se encarga también de mantener una lista de funciones `scripteables`, es decir, funciones en C++ que pueden ser invocadas desde un *script*. Se trata de una lista de punteros a función indexados por el nombre de la función. Haciendo uso del polimorfismo y la herencia, se mantiene una clase padre `ScriptFunction` que sirve de interfaz para todas las funciones que la implementen. Además, define algunos métodos útiles para formatear los argumentos pasados a estas funciones, ya que para mantener la interfaz, los argumentos son pasados como cadenas de texto. El Listado 6.19 muestra el método del gestor que se encarga de realizar la asociación de funciones *scripteables* con su nombre.

```
1  void ScriptManager::fillHandlers() {
2    _handlers["Wait"] = new WaitScriptFunction;
3    _handlers["DrawImage"] = new DrawImageScriptFunction;
4    _handlers["DrawVideo"] = new DrawVideoScriptFunction;
5    _handlers["DrawCorners"] = new DrawCornersScriptFunction;
6    _handlers["DrawAxis"] = new DrawAxisScriptFunction;
7    _handlers["DrawVideoStream"] = new DrawVideoStreamScriptFunction;
8    _handlers["DrawTextPanel"] = new DrawTextPanelScriptFunction;
9    _handlers["DrawHighlight"] = new DrawHighlightScriptFunction;
10   _handlers["DrawButton"] = new DrawButtonScriptFunction;
11   _handlers["DrawFactureHint"] = new DrawFactureHintScriptFunction;
12   _handlers["PlaySound"] = new PlaySoundScriptFunction;
13   _handlers["CheckRegion"] = new CheckRegionScriptFunction;
14 }
```

Listado 6.19: Asociación de funciones *scripteables* con su nombre.

Conocido esto, podemos introducir la ejecución de las sentencias del *script*. Los Listados 6.20 y 6.21 muestran la ejecución de las sentencias a nivel de *script* y a nivel de sentencia, respectivamente. La instancia de la clase `ScriptManager` es pasada mediante inyección de dependencias a la clase `ScriptSentence` para reducir el acoplamiento. Nótese el acceso directo a una variable privada de la clase `ScriptManager` desde una instancia de la clase `ScriptSentence`; esto es posible al declarar la segunda como `friend` de la primera.

```
1  void Script::runOneSentence() {
2    if(_sentences.empty())
3      return;
4
5    _sentences[_currentSentence++].execute(*this, _scriptManager);
6
7    if(_currentSentence >= _sentences.size()) {
8      _currentSentence = 0;
9      if(!_repeat)
10       _enabled = false;
11   }
12 }
```

Listado 6.20: Ejecución de una sentencia del *script*.

```
1 void ScriptSentence::execute(Script& owner,
2                               const ScriptManager& scriptManager) {
3   scriptManager._handlers.at(_command)->execute(owner, _args);
4 }
```

Listado 6.21: Ejecución de la sentencia en si.

Funciones *scripteables*

La Figura 6.12 muestra el diagrama de clases relativo a la clase `ScriptFunction`. Dicha clase, como comentamos anteriormente, proporciona una interfaz a que deben implementar todas las clases que hereden de ella.

Las clases hijas, deben implementar el método `execute`, que será invocado automáticamente cada vez que se ejecute una sentencia del *script*. Sin embargo, las clases especializadas pueden crear otros métodos libremente.

Este enfoque proporciona una solución sencilla y eficaz, a la hora de asociar funciones a su propio nombre, en lenguajes que no soportan reflexión o *reflection* por su término en inglés, es decir, que no permiten el uso de técnicas para inspeccionarse a si mismos en tiempo de ejecución, porque no poseen tal capacidad. C++ presenta este problema.

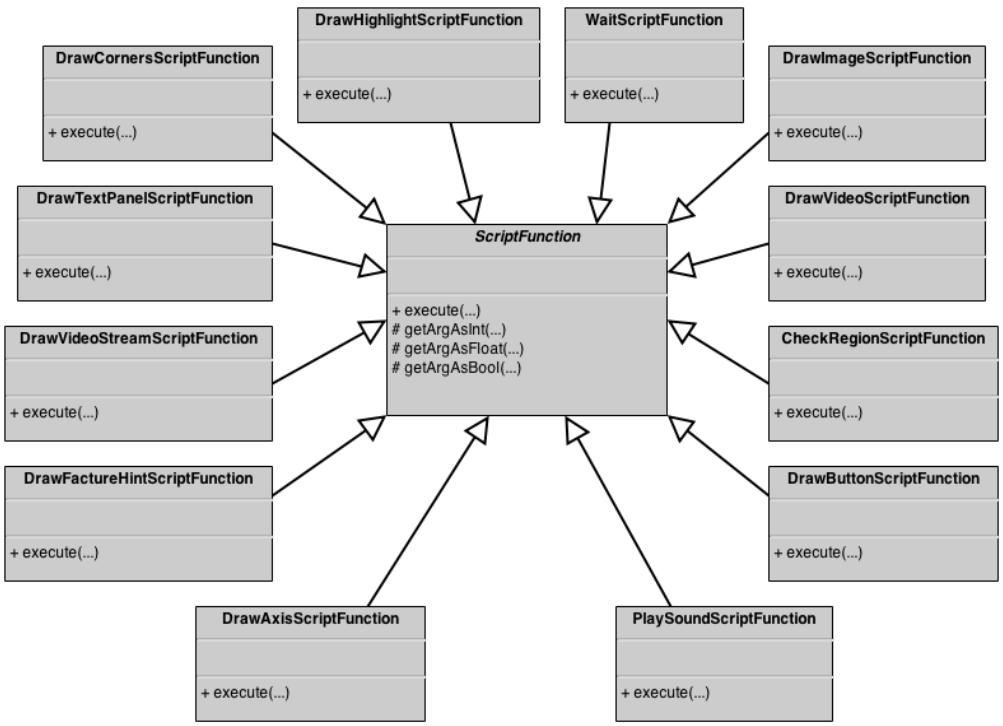

Figura 6.12: Diagrama de clases relativo a la clase `ScriptFunction`.

Ventajas de la implementación

El uso de un lenguaje *scripts* propio, supone una manera de moldear el lenguaje para que cumpla exactamente tus expectativas. Haber usado una solución externa basada en otro lenguaje de *scripts* como Python o Lua, habría supuesto la carga adicional de una complejidad de implementación que no se cree necesaria.

Las ventajas que ofrece al usuario son variadas, permitiéndole gran flexibilidad a la hora de establecer acciones a realizar para un documento basándose en un lenguaje de *scripts* muy sencillo de utilizar.

6.1.4 Clases de utilidad

A continuación se presentan algunas clases de utilidad comunes al cliente y el servidor. Pese a que no se han mostrado en ningún diagrama de clases, su utilidad es más que suficiente como para dedicarles su propia sección.

Registro de eventos o *Log*

Todos los subsistemas de BelfegAR mantienen un registro de eventos con todos los sucesos que ocurren durante su ejecución. Se trata de una clase estática, que permite la escritura de mensajes a la salida estándar (`STDOUT`) o a un fichero, junto a un *timestamp* que indica la fecha y la hora del suceso.

La clase, permite la salida de texto en color. Al inicio de BelfegAR y durante la fase de pre-ínicialización, se invoca el método `Log::setColouredOutput(isatty(fileno(stdout)));`. Dicho método establece la salida de texto en color solo si la salida estándar soporta colores. En cualquier otro caso, las sucesivas llamadas a los métodos de la clase `Log` mostrarán texto sin color. Esto es útil, porque el uso de colores implica añadir códigos numéricos a los mensajes que son interpretados por la salida estándar si ésta soporta el uso de colores. Sino, los mensajes serán mostrados con los códigos correspondientes a los colores sin ser éstos aplicados en realidad. Esta situación puede darse a la hora de redirigir la salida estándar a un fichero, donde no queremos que se empleen colores.

La clase además proporciona algunos métodos útiles, para imprimir vectores y matrices de 4x4. También soporta el uso de distintos niveles de evento:

- *plain*: Registra un mensaje sin colores.
- *info*: Registra un mensaje de información. Color azul.
- *error*: Registra un mensaje de error. Color rojo.
- *success*: Registra un mensaje de éxito. Color verde.
- *video*: Registra un mensaje relativo a vídeo. Color amarillo.

El Listado 6.23 muestra la implementación del método `void success(const std::string& msg, const std::string& filename);`. Por otro lado, el Listado 6.22 muestra los códigos de color unicode que pueden ser utilizados por la clase `Log`.

```
/**
* Codes for some colours
* FG_* states for foreground colours
* BG_* states for background colours
*/
enum Colour {
  FG_DEFAULT          = 39,
  FG_BLACK            = 30,
  FG_RED              = 31,
  FG_GREEN            = 32,
  FG_YELLOW           = 33,
  FG_BLUE             = 34,
  FG_MAGENTA          = 35,
  FG_CYAN             = 36,
  FG_LIGHT_GRAY       = 37,
  FG_DARK_GRAY        = 90,
  FG_LIGHT_RED        = 91,
  FG_LIGHT_GREEN      = 92,
  FG_LIGHT_YELLOW     = 93,
  FG_LIGHT_BLUE       = 94,
  FG_LIGHT_MAGENTA    = 95,
  FG_LIGHT_CYAN       = 96,
  FG_WHITE            = 97,

  BG_DEFAULT          = 49,
  BG_RED              = 41,
  BG_GREEN            = 42,
  BG_BLUE             = 44,
};
```

Listado 6.22: Lista de posibles colores a usar por la clase `Log`.

```
void Log::success(const string& msg, const string& filename) {
  if(coloured_output)
    std::cout << "\033[" << Colour::FG_LIGHT_GREEN << "m";

  std::cout << currentDateTime() << " [SUCCESS] " << msg << std::endl;

  if(coloured_output)
    std::cout << "\033[" << FG_DEFAULT << "m";

  if(!filename.empty()) {
    std::ofstream ofs(filename, std::ofstream::app);
    ofs << currentDateTime() << " [SUCCESS] " << msg << std::endl;
    ofs.close();
  }
}
```

Listado 6.23: Implementación de la función `success` de la clase `Log`.

Temporizadores o *Timer*

Los temporizadores son recursos muy útiles para medir el transcurso del tiempo. En nuestro caso, se ha implementado una clase `Timer` para controlar algunos aspectos relativos al tiempo, como la cancelación de una videollamada si no se ha recibido vídeo durante 1 segundo.

La clase se ha implementado haciendo uso de la biblioteca `chrono` del estándar. Implementaciones anteriores en C no proporcionaban el tiempo real, sino los ciclos de CPU, por lo que la información obtenida no era demasiado precisa.

Esta implementación permite obtener el tiempo que ha pasado entre dos intervalos, a una precisión que abarca desde microsegundos a horas.

El Listado 6.24 muestra un ejemplo de uso de la clase `Timer`.

```
1 Timer t;
2 t.start();
3 // Some long processing that
4 // lasts about 4 seconds
5 // ...
6 std::cout << t.getSeconds() << std::endl // Would print 4
```

Listado 6.24: Ejemplo de uso de la clase `Timer`.

6.2 Evolución del proyecto

La evolución del proyecto se ha llevado a cabo a lo largo de distintas etapas, que reúnen la identificación de requisitos, revisión bibliográfica, adquisición del hardware necesario e implementación de la solución.

En esta sección, se explicarán los anteriores puntos y además se introducirán las medidas de rendimiento o *profiling* realizadas a BelfegAR.

6.2.1 Análisis de requisitos

De acuerdo a la naturaleza de BelfegAR orientado a personas con necesidades especiales, la identificación de requisitos se llevó a cabo con estos usuarios en mente. Se elaboró una lista de requisitos básicos a cumplir:

- El sistema debe de ser fácilmente accesible y usable.
- El sistema debe permitir el soporte remoto de sus usuarios.
- El sistema debe basar su implementación en tecnologías libres.
- El sistema debe ser multiplataforma.
- El sistema debe de estar basado en componentes de bajo coste.

Una vez cumplidos esos requisitos básicos, nuevos requisitos aparecían conforme se desarrollaba el proyecto, como mejoras en la accesibilidad añadiendo nuevas funcionalidades gráficas y auditivas.

Se ideó además una lista de componentes gráficos general que debía ser implementada como mínimo. Dicha lista indicaba las ideas generales de qué debía de hacer cada componente.

6.2.2 Revisión bibliográfica

Antes de iniciar el proyecto, se estudió el estado del arte en cuestión de bibliotecas de despliegue gráfico. Después de realizar algunas pruebas con el hardware sobre que se debía de ejecutar el sistema, se entendió la necesidad de emplear aceleración por hardware para el tema de despliegue gráfico.

No fue sin embargo hasta que entró en escena GrayAR, cuando se empezaron a investigar soluciones de delegación de tareas en equipos remotos, debido a la gran carga que suponía el procesamiento de imágenes mediante visión por computador a la Raspberry Pi.

El tema de audio se estudió en una de las últimas etapas del proyecto, cuando casi todo lo demás ya se encontraba implementado.

6.2.3 Iteraciones

A continuación, se indicarán las iteraciones realizadas durante el desarrollo de BelfegAR, detallando los objetivos obtenidos en cada una de ellas.

Iteración 1

En esta primera iteración, se empezó implementando el sistema de superficies aceleradas por hardware mediante `EGL` para conseguir un contexto de OpenGL ES funcional y estable. Se implementó además la carga, compilación, enlazado y activación de programas OpenGL basados en *shaders*.

Se creó también el primer componente gráfico; `RectangleComponent`. Éste solo se dibujaba de forma fija en el escenario, y la posición llegó a controlarse modificando directamente los vértices de la figura en lugar de aplicar matrices de transformación.

Pese a que todo acabó funcionando correctamente, surgieron varios problemas al emplear la versión empotrada de OpenGL, ya que al emplear un cauce programable, la dificultad aumentaba al eliminar el uso de las matrices `MODELVIEW` y `PROJECTION`, y las correspondientes operaciones con los métodos `glPushMatrix` y `glPopMatrix` para operar sobre ellas. También se eliminó de esta versión los métodos de construcción de objetos 3D, como `glBegin`, `glVertex` y `glEnd`, pasándose a utilizar listas de vértices con el *shader* correspondiente para su renderizado. El nuevo modo de funcionamiento supuso a OpenGL el soporte de paralelismo en su cauce de renderizado, además de aumentar el rendimiento general de las operacio-

nes, al no dejar toda la carga al procesador, en el que se requería al menos la invocación de una función por vértice.

La meta que se propuso para esta primera iteración fue conseguir un contexto de OpenGL ES sobre el que poder trabajar, la instalación de dependencias del proyecto y la generación de los *Makefiles* necesarios para su construcción.

Esta iteración estuvo comprendida entre las fechas 02/12/2013 y 08/01/2014.

Iteración 2

Una vez obtenido un entorno de despliegue gráfico funcional con un primer componente gráfico, se procedió a mejorar dicho componente para que se le pudieran aplicar varias transformaciones 3D. La primera transformación que se probó fue la de *translación*, para comprobar si funcionaba correctamente. Una vez implementada, se implementaron operaciones para *escalar* y *rotar* la figura.

Se creó además un nuevo componente gráfico; `LineComponent`. A partir de este, se comprendió mejor el uso de las listas de vértices. Manipulando los colores de las líneas e instanciando tres de ellas, se consiguió crear un eje de coordenadas.

También se investigó el tema de la compilación cruzada, ya que la construcción del sistema usando directamente la Raspberry Pi podía durar varios minutos. En el Anexo B se encuentra de forma detallada como se hizo esto.

En esta iteración, se afianzaron los conocimientos sobre OpenGL ES 2.0 y se ideó un nuevo componente gráfico. También se aplicaron las primeras transformaciones que permitían mover, escalar y rotar un objeto. Por último se empezó a hacer uso de un entorno de compilación cruzada para construir el sistema.

Esta iteración estuvo comprendida entre las fechas 08/01/2014 y 24/02/2014.

Iteración 3

En esta iteración GrayAR ya podía detectar documentos y proporcionar sus matrices de modelo y vista, además de la matriz de proyección del entorno. Gracias a esto, se pudieron aplicar dichas matrices a los componentes gráficos para posicionarlos sobre los documentos.

Surgieron ciertas confusiones al principio relacionadas con las unidades del mundo de OpenGL y las definidas como centímetros en la realidad.

Se realizó un refactorizado completo a los dos componentes gráficos existentes, extrayendo comportamiento y atributos comunes y se creó la clase abstracta `GraphicComponent`, de la cual heredarían el resto de componentes gráficos en el futuro.

Se creó además el componente gráfico *ImageComponent*, el cual suponía el reto de cargar

imágenes desde disco. Se estudiaron por tanto algunas bibliotecas que aportaran dicha funcionalidad, deduciendo que SOIL era la mejor opción, por ser una biblioteca pequeña y que solo realiza la función que queríamos: cargar imágenes.

También se creó en esta iteración el componente de texto o `TextComponent`, necesario para renderizar texto, usado en primera instancia para mostrar el número de fotogramas por segundo que alcanzaba el sistema.

Esta iteración concluyó de esta forma con un sistema de componentes gráficos ya afianzado, y fácilmente escalable y mantenible, que se adaptaban correctamente al documento.

Esta iteración estuvo comprendida entre las fechas 24/02/2014 y 10/03/2014.

Iteración 4

Como veremos en la sección de herramientas complementarias desarrolladas (véase § 6.3), GrayAR requería una forma de mostrar un patrón de círculos configurable para su herramienta de calibración. Por ello, se ideó un nuevo componente gráfico que permitiera dibujar círculos de forma sencilla; `CircleComponent`. La lógica de creación del círculo fue realizada en el propio *shader* para aliviar la carga de la unidad. Esto supuso el estudio más en profundidad del lenguaje GLSL.

También se creó un componente gráfico que permitiera cargar vídeos desde el disco; `VideoComponent`. Debido a limitaciones de tiempo, se decidió realizar la decodificación del vídeo por *software* mediante OpenCV. Esto será discutido en el Capítulo 7.

Debido al bajo rendimiento que ofrecía la plataforma, se ideó una arquitectura de red que permitiera ejecutar las tareas de visión por computador en una máquina de mayor potencia. Se esbozó pues una primera versión del subsistema de delegación de tareas, que permitía el envío de matrices serializadas por red, mediante *sockets* TCP.

Los resultados de esta iteración pues, fueron el desarrollo de dos nuevos componentes gráficos y un primer esbozo del subsistema de delegación de tareas.

Esta iteración estuvo comprendida entre las fechas 10/03/2014 y 31/03/2014.

Iteración 5

En esta iteración, se consolidó el subsistema de delegación de tareas, permitiendo enviar más tipos de datos por red y estructurando el sistema de tal forma que hiciera un uso obligatorio de la delegación de tareas.

Se implementó también la arquitectura del subsistema de comunicaciones para poder migrar GrayAR al servidor. Con la arquitectura de red montada, se estableció el contrato de comunicación que debían seguir tanto el cliente como el servidor para la transmisión de la información.

A estas alturas del desarrollo, el sistema ya contaba con la arquitectura necesaria y escalable para añadir nuevos componentes gráficos y tipos de datos en la comunicación cliente-servidor.

Esta iteración estuvo comprendida entre las fechas 31/03/2014 y 28/04/2014.

Iteración 6

Una vez establecida una arquitectura cliente-servidor funcional, se pretendió cubrir en esta iteración el sistema de soporte al usuario mediante videollamadas con la creación del `VideoStreamComponent`. Para ello, se tuvo que cambiar la arquitectura del servidor para soportar una nueva comunicación a través de otro puerto. Se implementó una comunicación mediante *sockets* UDP paralela a la comunicación principal de los subsistemas de delegación de tareas y comunicación para el envío de vídeo desde el servidor.

También se implementó cierta interacción primitiva con el usuario, de tal forma que el cliente decidía que componentes gráficos mostrar en función del documento analizado. Esta primera aproximación permitía darle la vuelta al documento para iniciar una videollamada, de tal forma que el documento por detrás conformaba otro identificador distinto a la otra cara, y cancelarla al volver a darle la vuelta.

Esta iteración concluyó con la implementación funcional del componente de videollamadas y una primera aproximación a lo que sería la interacción del usuario con el sistema.

Esta iteración estuvo comprendida entre las fechas 28/04/2014 y 12/05/2014.

Iteración 7

En vistas a crear componentes gráficos más complejos a partir de otros ya existentes, y como si de un programa de dibujado se tratará, se creó el `RenderToTextureComponent`. Se ideó en primera instancia para crear mensajes de ayuda que indicaran al usuario las posibles acciones a realizar con el documento.

El estudio del uso de la técnica de «render a textura», requirió cierto tiempo por introducir una gran variedad de nuevos conceptos como *framebuffers* compuestos de *renderbuffers* y *texturas*, además del uso de una matriz de proyección ortogonal única para este componente.

Finalmente, se creó un método *ayudante* para construir dichas ayudas al documento mediante parámetros como el tamaño de la imagen de ayuda, su color y los mensajes que contendría.

Esta iteración concluyó con la creación de un componente gráfico que permitía formar nuevos componentes más complejos, y un método para crear uno de esos componentes específicos.

Esta iteración estuvo comprendida entre las fechas 12/05/2014 y 23/06/2014.

Iteración 8

Debido a la cantidad de componentes gráficos existentes a estas alturas del proyecto, se creo un gestor que se ocupara de administrar su creación, actualización, dibujado y destrucción, es decir, su ciclo de vida completo. Estamos hablando de la clase `GraphicComponentsManager`.

También se creó un contenedor de componentes gráficos o `GCCollection` que permitiera la gestión conjunta de componentes que compartieran cierto propósito. Desde este momento, el gestor de componentes gráficos se encargaba de administrar estos contenedores.

El gestor también proporcionaba métodos creacionales de componentes gráficos complejos como botones y paneles de texto empleando componentes de «render a textura», y videollamadas sobre superficies de imagen mediante colecciones de componentes individuales.

El objetivo de esta iteración se vio reflejado en el afianzamiento del subsistema de representación gráfica.

Esta iteración estuvo comprendida entre las fechas 23/06/2014 y 07/07/2014.

Iteración 9

Con el subsistema de representación gráfica finalizado, se inició y completó el desarrollo del subsistema de audio en esta iteración.

Se adquirió una placa con altavoz integrado que se conectó a la Raspberry Pi para proporcionarle posibilidades de audio, ya que por defecto ésta no incluye características de sonido.

Se empleó una frecuencia de 16.000 Hz para la reproducción de sonidos, la cual proporcionaba una calidad adecuada al altavoz utilizado.

También se realizaron algunas mejoras en el código fuente, y se corrigieron algunos *bugs*. Después se realizó un refactorizado global para eliminar funciones y partes de código obsoletas.

Esta iteración supuso el funcionamiento de la mayor parte del sistema. Solo quedaba especificar las acciones a realizar por cada documento analizado.

Esta iteración estuvo comprendida entre las fechas 07/07/2014 y 14/07/2014.

Iteración 10

En la iteración final, se estudió una manera de facilitar al usuario la especificación de diferentes acciones a realizar para ciertos documentos. Se ideó pues el subsistema de *scripts*, que permitía definir archivos interpretables por el servidor para su procesado y posterior envío de resultados al cliente.

El desarrollo del subsistema de *scripts* se basó en una implementación ya realizada para

el Trabajo Fin de Curso (TFC) del «Curso de Experto en Desarrollo de Videojuegos (3ra. edición 2013/2014)», **The Secret of Liches** [1]. La nueva implementación supuso una mejor estructura del código y un mayor uso del polimorfismo para la invocación de funciones *scripteables* desde C++.

Durante esta iteración, también se escribió parte de la documentación del proyecto y el manual de usuario del sistema, entre otros documentos como el manual de compilación cruzada, que en un principio se encontraba externo a este documento.

Esta iteración estuvo comprendida entre las fechas 14/07/2014 y 25/08/2014.

6.3 Herramientas complementarias desarrolladas

A continuación se introducirán algunas herramientas independientes creadas para facilitar algunas tareas del desarrollo del proyecto.

6.3.1 OGL Pattern

GrayAR requiere de un sistema de calibrado mediante patrones. Sabiendo esto, se desarrolló una aplicación que mostrara un patrón de círculos configurable. La Figura 6.13 muestra dicho patrón.

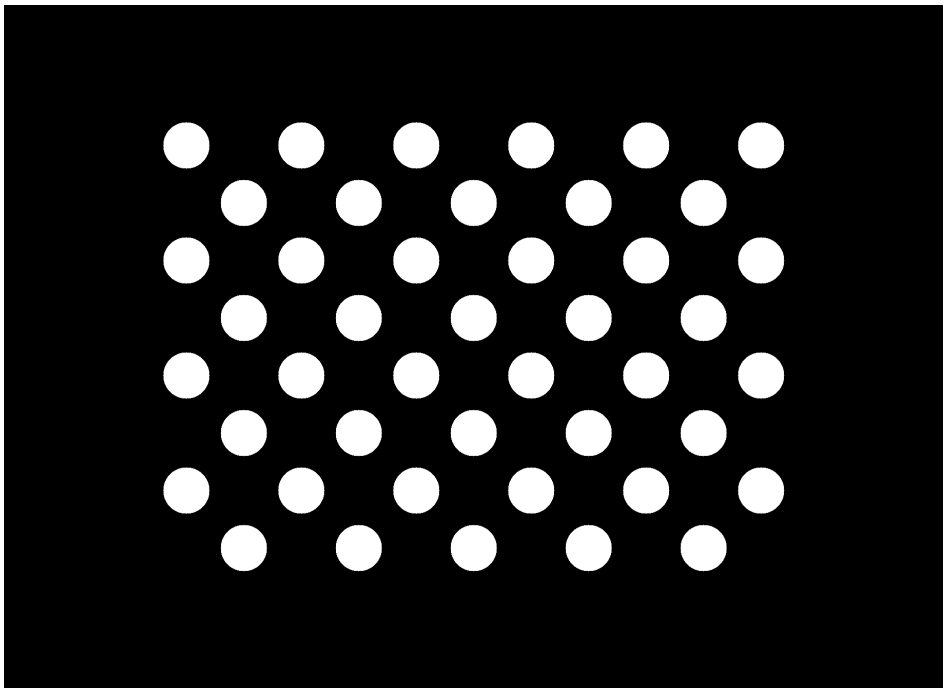

Figura 6.13: Patrón de círculos desarrollado para GrayAR.

El diseño de la aplicación siguió las líneas de desarrollo de BelfegAR, es decir, se construyó un subconjunto con una arquitectura de componentes gráficos mínima, con tal de mantener la mayor simplicidad posible. Se reemplazó la carga de *shaders* desde archivos por dos

[1] Vídeo del videojuego disponible en `https://www.youtube.com/watch?v=J5ap3Gi4Dq8`

shaders incluidos directamente en el código fuente C++. Además, se realizó un *port* de la biblioteca `android.opengl.Matrix` a C++ para no tener que depender de GLM para una aplicación tan simple. Dicho *port* proporcionaba operaciones de transformaciones geométricas a través de una clase estática que aplicaba las transformaciones a las matrices pasadas como argumentos.

El Listado 6.25 muestra las funciones que proporciona la clase `Matrix`.

```
1  class Matrix {
2  public:
3    static void setIdentityM(float* m);
4    static void transposeM(float* mTrans, float* m);
5    static bool invertM(float* mInv, float* m);
6    static float length(float x, float y, float z);
7
8    static void multiplyMM(float* result, float* lhs, float* rhs);
9    static void multiplyMV(float* resultVec, float* lhsMat,
10                          float* rhsVec);
11
12   static void frustumM(float* m, float left, float right,
13                        float bottom, float top, float near,
14                        float far);
15   static void orthoM(float* m, float left, float right,
16                      float bottom, float top, float near, float far);
17   static void perspectiveM(float* m, float fovy, float aspect,
18                            float zNear, float zFar);
19
20   static void rotateM(float* m, float a, float x, float y, float z);
21   static void rotateM(float* rm, float* m, float a, float x,
22                       float y, float z);
23   static void setRotateEulerM(float* rm, float x, float y, float z);
24
25   static void scaleM(float* sm, float* m, float x, float y, float z);
26   static void scaleM(float* m, float x, float y, float z);
27
28   static void translateM(float* m, float x, float y, float z);
29   static void translateM(float* tm, float* m, float x, float y,
30                          float z);
31
32
33   static void setLookAtM(float* m, float eyeX, float eyeY, float eyeZ,
34                          float centerX, float centerY, float centerZ,
35                          float upX, float upY, float upZ);
36 };
```

Listado 6.25: Métodos de la clase `Matrix`.

Fundamentalmente, la aplicación implementa dos métodos: uno para crear círculos independientes y otro para crear un *grid* o tabla de círculos que represente al patrón de la Figura 6.13. El Listado 6.26 muestra la implementación del método `createCircle` utilizado para crear círculos independientes, mientras que e Listado 6.27 muestra la implementación del método `createCirclesGrid` empleado en la creación del patrón de círculos.

```
GLContext::Circle* GLContext::createCircle(GLfloat x, GLfloat y,
    GLfloat z, GLfloat size) {
  GLfloat vertices[] = {
  //    X      Y     Z     U      V
    -size,  size, 0.0f, 0.0f,  0.0f,
    -size, -size, 0.0f, 0.0f,  1.0f,
     size, -size, 0.0f, 1.0f,  1.0f,
     size,  size, 0.0f, 1.0f,  0.0f
  };
  const char vShaderStr[] =
    "attribute vec4 a_position;           \n"
    "attribute vec4 a_texCoord;           \n"
    "uniform mat4 u_mvp;                  \n"
    "varying vec2 textureCoordinate;      \n"
    "void main() {                        \n"
    "  gl_Position = u_mvp * a_position;  \n"
    "  textureCoordinate = a_texCoord.xy; \n"
    "}                                    \n";
  const char fShaderStr[] =
    "varying highp vec2 textureCoordinate;       \n"
    "const highp vec2 center = vec2(0.5, 0.5);   \n"
    "const highp float radius = 0.5;             \n"
    "void main() {                               \n"
    "  highp float distanceFromCenter =          \n"
    "    distance(center, textureCoordinate);    \n"
    "  lowp float checkForPresenceWithinCircle = \n"
    "    step(distanceFromCenter, radius);       \n"
    "  gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0)   \n"
    "    * checkForPresenceWithinCircle;         \n"
    "}                                           \n";
  Circle* circle = new Circle;
  circle->programObject = loadGLProgram(vShaderStr, fShaderStr);
  circle->positionHandler =
    glGetAttribLocation(circle->programObject, "a_position");
  circle->texCoordHandler =
    glGetAttribLocation(circle->programObject, "a_texCoord");
  circle->mvpHandler =
    glGetUniformLocation(circle->programObject, "u_mvp");
  circle->x = x; circle->y = y; circle->z = z; circle->size = size;
  memcpy(circle->vertices, vertices, 20*sizeof(float));

  Matrix::setIdentityM(circle->mvp);

  return circle;
}
```

Listado 6.26: Método de creación de círculos `createCircle`.

```
void GLContext::createCirclesGrid(GLfloat x, GLfloat y,
  GLint width, GLint height, GLfloat circles_size, GLfloat spacing) {
  for(int i = 0; i < width; ++i) {
    for(int j = 0; j < height; ++j) {
      _circles.push_back(createCircle(x+(i*spacing), y+(j*spacing),
                         0.0f, circles_size)); } } }
```

Listado 6.27: Método de creación del patrón de círculos `createCirclesGrid`.

Tras finalizar la herramienta, esta fue empleada por GrayAR, previo añadido de las funciones de visión por computador necesarias, también competencias de GrayAR.

6.3.2 BelfegAR Videostreamer

Antes de desarrollar el componente gráfico `VideoStreamComponent`, se aisló el problema en una aplicación autónoma que replicara el funcionamiento del sistema a pequeña escala. Se implementó de este modo una arquitectura de red basada en *sockets* UDP que sirviera para el envío y recepción de los fotogramas de vídeo obtenidos de las cámaras del cliente y del servidor.

La aplicación interaccionaba directamente con una versión temprana del sistema completo ARgos. Para ejecutarla, se debía pasar por línea de órdenes la dirección IP y el puerto de escucha del cliente donde debería esperar por fotogramas de vídeo.

El Listado 6.28 muestra el bucle principal de la aplicación.

```
...
cv::Mat frame;
while(1) {
  // Grab a new frame from the camera
  if(!cam.grab()) continue;
  if(!cam.retrieve(frame) || frame.empty()) continue;
  if(cv::waitKey(30) >= 0) break;

  // Send the new frame to the endpoint
  cv::flip(frame, frame, 0);
  cv::cvtColor(frame, frame, cv::COLOR_BGR2RGB);
  size_t bytes = sendCvMat(frame);
  if(bytes > 0) {
    Logger::Log::info("Frame: "
                      + std::to_string(frame.cols) + "x"
                      + std::to_string(frame.rows) + ", "
                      + std::to_string(bytes) + " bytes."
                     );
  }

  // Receive frame from the endpoint and show it to the user
  cv::Mat received;
  receiveCvMat(received);
  cv::flip(received, received, 0);
  imshow("Video Stream", received);
}
```

Listado 6.28: Bucle de captura, envío y recepción de fotogramas de «BelfegAR Videostreamer».

Una vez comprobado el correcto funcionamiento de la aplicación se pasó a refactorizar y migrar el código a BelfegAR.

6.3.3 Prototipo virtual de ARgos

En una de las primeras etapas del proyecto y para probar la arquitectura de red, se decidió realizar un modelo virtual de todo el sistema ARgos, pero que implementara únicamente el funcionamiento de BelfegAR junto a las matrices de modelo, vista y proyección proporcionadas por GrayAR.

Este modelo del sistema cumplía con las siguientes características:

- Visualización directa del entorno del sistema.
- Envío de fotogramas a través de red mediante *sockets* UDP.
- Envío de matrices de modelo, vista y proyección del documento a través de la red.
- Visualización en tiempo real de los fotogramas y matrices recibidas en el servidor.
- Interacción del usuario con diversos elementos del entorno mediante teclado y ratón.

Dado que no entra dentro del ámbito de este documento explicar como se ha realizado toda la implementación en Blender de la demostración se procederá únicamente a comentar la parte de red, omitiendo aspectos relacionados con el Blender Game Engine (BGE), y a mostrar algunas imágenes del sistema funcionando.

El sistema se ayuda de una primitiva versión cliente de la clase `TaskDelegation` de BelfegAR, la cual inicia un bucle de envío de información constante al servidor. La información enviada se compone de las matrices de modelo, vista y proyección, y del fotograma capturado por la cámara en ese momento. La información es copiada en un *buffer* como una secuencia de bytes y transmitida al servidor para su procesado. Cabe destacar, que los *sockets* UDP por estándar no permiten el envío de paquetes que ocupen más de 65536 bytes, por lo que la imagen se comprimía en JPEG manteniendo un 80 % de la calidad original. El Listado 6.29 muestra los métodos empleados para construir el *buffer* de datos que se enviarán al servidor.

```
1  void TaskDelegation::addMatrix(const float* matrix) {
2    int size = 16 * sizeof(float);
3    unsigned char sMatrix[size];
4    memcpy(sMatrix, matrix, size);
5    _buff.insert(_buff.end(), &sMatrix[0], &sMatrix[size]);
6  }

8  void TaskDelegation::addCvMat(const cv::Mat& mat) {
9    std::vector<unsigned char> mat_buff;
10   std::vector<int> params;

12   // Compress the image, since UDP packet <= 65536 bytes
13   params.push_back(CV_IMWRITE_JPEG_QUALITY);
14   params.push_back(80);
15   cv::imencode(".jpg", mat, mat_buff, params);

17   int length = mat_buff.size();
```

```
18    unsigned char packet_size[sizeof(int)];
19    memcpy(packet_size, &length, sizeof(int));

21    _buff.insert(_buff.end(), &packet_size[0],
22                 &packet_size[sizeof(int)]);
23    _buff.insert(_buff.end(), mat_buff.begin(), mat_buff.end());
24  }
```

Listado 6.29: Métodos utilizados para construir el *buffer* de datos.

El Listado 6.30 por otra parte, muestra el bucle de envió de datos que ocupa a la aplicación.

```
1   ... // Initializing stuff
2   cv::Mat currentFrame;
3   TaskDelegation* td = NULL;
4   if(argc > 1) {
5     td = new TaskDelegation();
6     td->setEndpoint(argv[1], argv[2]); // [1] -> ip | [2] -> port
7   }

9   while(1) {
10    ... // Gets camera frame and store it in the currentFrame object

12    if(td) {
13      td->addMatrix(projection_matrix);  // Adds a matrix (float[16])
14      td->addMatrix(modelview_matrix);  // Adds another matrix
15      td->addCvMat(currentFrame);  // Adds a cv::Mat
16      td->send();  // Sends the built buffer through the socket
17    }

19    ...
20  }

22  ...
```

Listado 6.30: Envío de las matrices y fotogramas al servidor implementado en Blender.

Una vez enviada la información, se procede a su procesado en el servidor, mediante un *script* Python ejecutado por la aplicación construida en Blender.

El Listado 6.31 muestra el código del servidor que permanece a la espera de recibir la información del cliente. Por otra parte, El Listado 6.32 muestra el código de las funciones que procesan las matrices y fotogramas recibidos.

```
1   def listen(cont):
2     global sock

4     obj = cont.owner

6     while 1:
7       data, addr = sock.recvfrom(32768)

9       ...
```

```
11      raw_matrix = data[0:64]
12      buff.append("\n> Projection Matrix:\n")
13      buff.append(getPrintableMatrix(parseMatrix(raw_matrix)))

15      raw_matrix = data[64:128]
16      buff.append("\n> ModelView Matrix:\n")
17      buff.append(getPrintableMatrix(parseMatrix(raw_matrix)))

19      cv_mat_length = data[128:132]
20      mat_length, = struct.unpack("<i", cv_mat_length)
21      raw_cv_mat = data[132:len(data)]
22      parseCvMat(raw_cv_mat)

24      ...
```

Listado 6.31: Recepción de la información enviada por el cliente.

```
1   def parseMatrix(raw_matrix):
2     m = struct.unpack("<16f", raw_matrix) # Unpack matrix from C memcpy
3     m = ["{:.2f}".format(i) for i in m]   # 2 decimals per element
4     m = [[m[0],  m[4],  m[8],  m[12]],    # Set to row-major order
5          [m[1],  m[5],  m[9],  m[13]],
6          [m[2],  m[6],  m[10], m[14]],
7          [m[3],  m[7],  m[11], m[15]]]

9     return m

11  def parseCvMat(raw_cv_mat):
12    f = open("frame.jpg",'wb')
13    f.write(raw_cv_mat)           # Writes the received image to disk
14    f.close()                     # in order to display it after
```

Listado 6.32: Procesado de la información recibida.

Una vez recibida y procesada la información, solo queda mostrarla mediante Blender y permitir la interacción del usuario, la cual se realiza mediante las teclas de movimiento WASD y el botón izquierdo del ratón.

A continuación se muestran algunas capturas de la aplicación desarrollada.

Figura 6.14: Vista general del sistema. Hasta que el usuario no interactúa con el portátil, no se inicia el servidor.

Figura 6.15: La unidad de detección y despliegue, Raspberry Pi, junto al proyector.

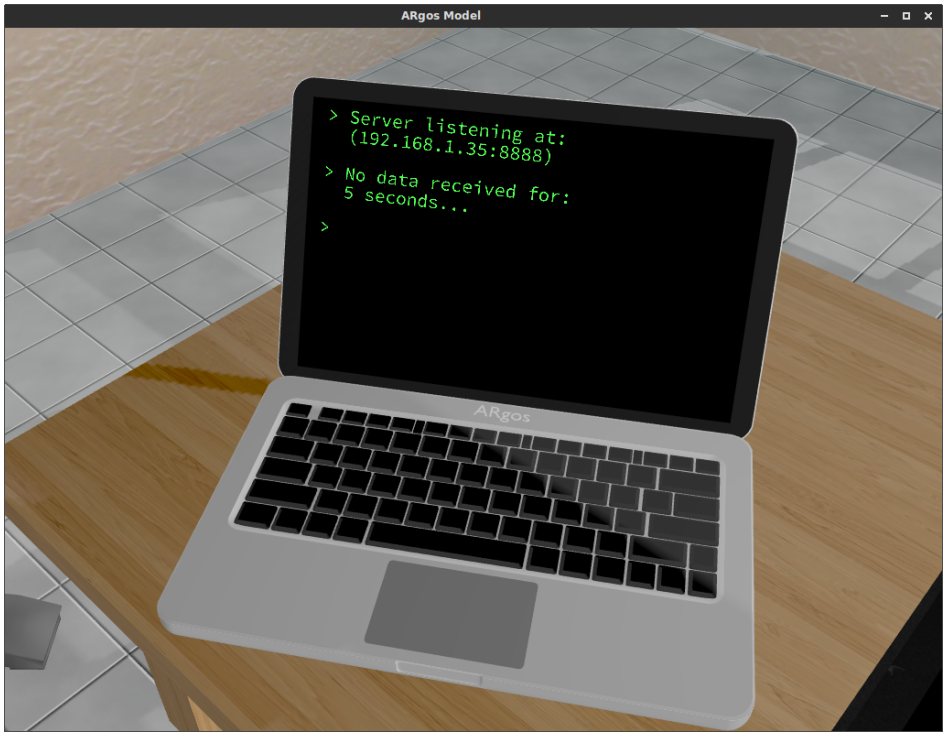

Figura 6.16: Servidor en ejecución y esperando información del cliente.

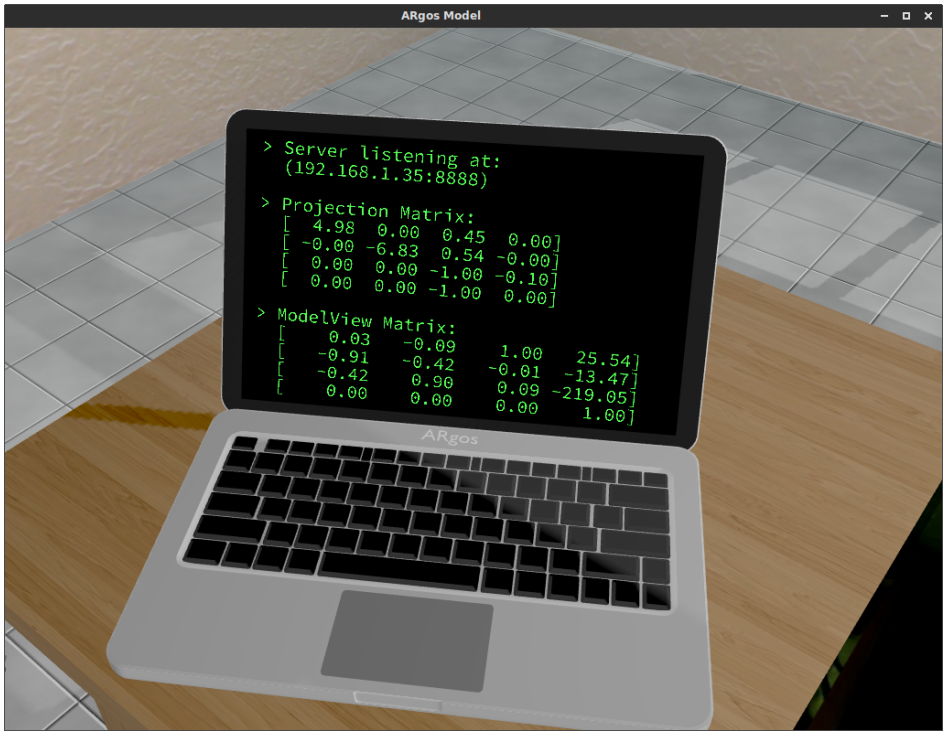

Figura 6.17: Servidor recibiendo y mostrando las matrices de modelo, vista y proyección del documento.

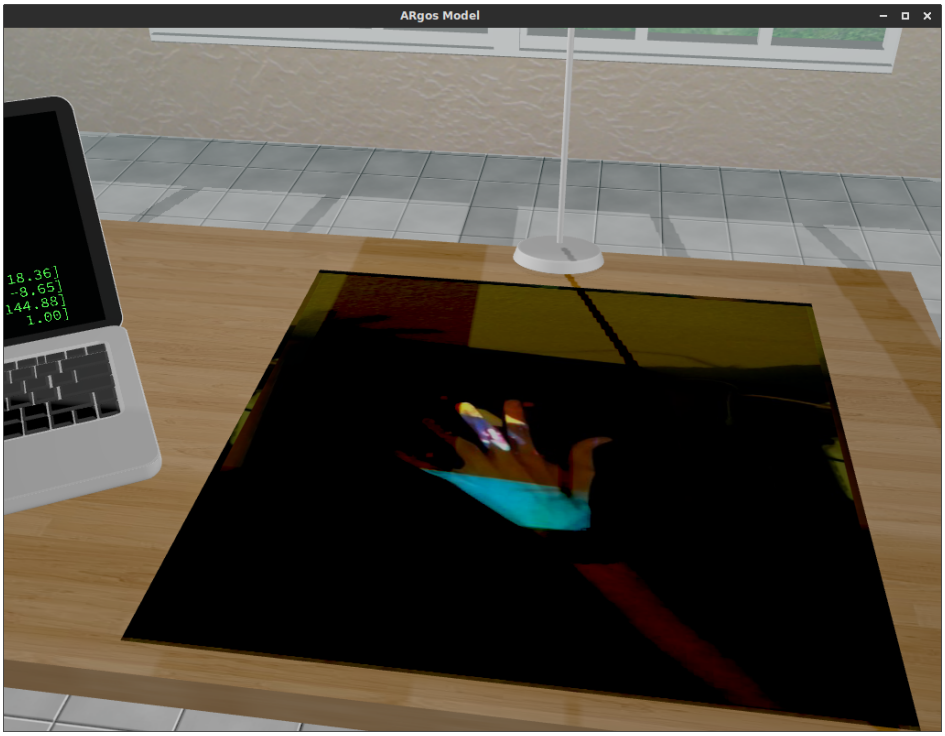

Figura 6.18: Los fotogramas recibidos son proyectados sobre la *cartulina* negra simulada.

6.4 Recursos y costes

En esta sección se introducirán los recursos y costes empleados por BelfegAR. Presentaremos estadísticas del proyecto y algunas medidas de rendimiento que se han aplicado junto a sus resultados. Respecto al apartado de costes, se hará una estimación de éstos de acuerdo a las condiciones de mercado actuales.

6.4.1 Estadísticas

La Tabla 6.1 muestra el número de líneas de código de BelfegAR. Se han omitido sin embargo las líneas de código pertenecientes a las herramientas externas desarrolladas y a los *scripts* ARS creados, los cuales sumarían en torno a 2000 líneas más de código. Para contabilizar dichas líneas, se ha empleado la aplicación CLI *cloc*.

Lenguaje	**Archivos**	**Espacios en blanco**	**Comentarios**	**Líneas de código**
C++	30	942	472	3480
Cabeceras C/C++	34	601	1546	1392
GLSL	12	0	0	99
make	2	29	0	93
Total:	**66**	**1572**	**2018**	**4965**

Cuadro 6.1: Número de líneas del código fuente de BelfegAR, entre el cliente y el servidor.

6.4.2 Coste económico

El desarrollo del presente Trabajo de Fin de Grado abarcó desde el día 2 de Diciembre de 2013, hasta el 25 de Agosto de 2014, aproximadamente. Esto supone un tiempo de desarrollo de 38 semanas, con una dedicación media de 4,5 horas diarias (5 días a la semana), el cual se traduce en 855 horas de trabajo. No se ha contabilizado sin embargo el tiempo dedicado a la elaboración del presente documento.

Se ha supuesto un precio de *30 €/hora* tomando como referencia la media de cobro actual de mercado en este sector [2].

La Tabla 6.2 muestra un desglose de gastos aproximados para el desarrollo de BelfegAR. La tabla incluye todo el hardware necesario para la puesta en marcha del sistema, así como el sueldo del programador calculado teniendo en cuenta lo anterior. Se muestra también el coste en total del proyecto.

El coste obtenido es meramente informativo, ya que a la hora de calcular un presupuesto se deben de tener en cuenta otros muchos factores.

[2] Información obtenida de `https://freelance.infojobs.net/freelancers/programador/informatica`.

Recurso	Cantidad	Coste
Sueldo programador (855 horas)	1	25.650 €
Computador de trabajo	1	499 €
Raspberry Pi (Modelo B)	1	40 €
Cámara Raspberry Pi	1	24,95 €
Proyector Optoma PK320	1	325 €
MicroSDHC Trascend 32GB	1	14,75 €
Altavoz WaveShare LM386	1	10 €
Webcam Logitech HD C270	1	26 €
Total		26.589,7 €

Cuadro 6.2: Desglose de costes de BelfegAR.

6.4.3 Profiling

El *profiling* o medida de rendimiento realizada a BelfegAR consiste en la medición de los tiempos que el sistema tarda en realizar las tres operaciones más críticas del sistema:

- Captura de imágenes desde la cámara.
- Envío/recepción de datos por red.
- Renderizado gráfico.

Para cada una de las partes se han realizado dos mediciones. En la primera, no se había aplicado *overclock* al sistema, mientras que en la segunda se había subido la frecuencia de la CPU hasta 900 MHz, desde los 800 MHz base.

Para el primer caso y tras 100 iteraciones del bucle principal se ha obtenido un tiempo de 54,12 milisegundos para la captura de imágenes desde la cámara, 266,98 milisegundos en el envío y recepción de datos por red y 10,15 milisegundos en renderizar los componentes gráficos. Los resultados de cada una de las iteraciones pueden observarse en la Figura 6.19.

Para el segundo caso y en las mismas condiciones se ha obtenido un tiempo de 46,65 milisegundos para la captura de imágenes, 238,28 milisegundos en el envío y recepción de datos por red y 8,61 milisegundos en renderizar los componentes gráficos. En este caso, los resultados pueden ser observados en la Figura 6.20.

Como podemos observar, *overclockear* la frecuencia del procesador en solo 100 MHz, supone una **mejora media del rendimiento del 15 %**. También concluimos de esto que el tiempo gastado en el envío y recepción de datos por red es excesivo y se debería profundizar por tanto en los subsistemas de delegación de tareas y comunicación. Aun así, la mayor cantidad de ancho de banda es consumido por el envío y recepción de los fotogramas de la cámara, los cuales son comprimidos únicamente un 20 %, por lo que una inminente mejora consiste en aumentar el ratio de comprensión de los fotogramas.

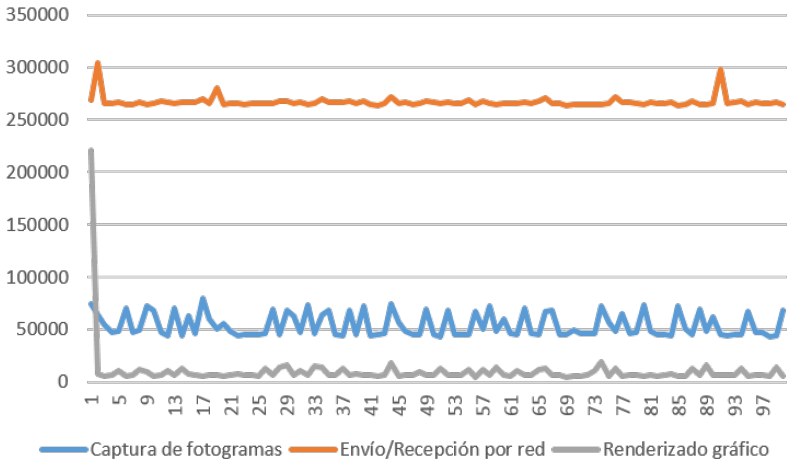

Figura 6.19: Resultados de 100 iteraciones con el procesador a 800MHz. El eje Y indica el tiempo en nanosegundos para las iteraciones del eje X.

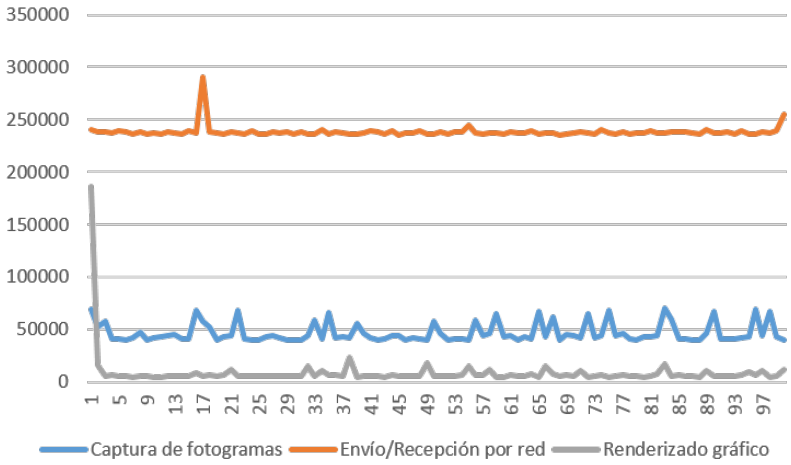

Figura 6.20: Resultados de 100 iteraciones con el procesador *overclockeado* a 900MHz. El eje Y indica el tiempo en nanosegundos para las iteraciones del eje X.

6.5 Caso de uso concreto

En este apartado se introducirá un caso de uso concreto del sistema ARgos funcionando, mediante fotos del sistema real funcionando. Además, se detallará cada imagen para dar a conocer el estado en qué se encuentra el sistema en ese momento.

6.5.1 Prerrequisitos

La Figura 6.21 muestra los componentes hardware empleados por la unidad de detección y despliegue. Previo inicio del sistema, el usuario cuenta con dos facturas que utilizará en esta demostración. Además, cada factura lleva asociada un *script* que será ejecutado una vez el sistema detecte la factura correspondiente.

6.5.2 Carga del sistema

Durante la carga del sistema, el registro de eventos muestra por la salida especificada los sucesos correspondientes al cliente y el servidor durante esa etapa (véase Figura 6.22 y Figura 6.23, respectivamente). Para el cliente (y detallando el registro correspondiente a BelfegAR), el registro muestra que la conexión con el servidor se ha realizado correctamente, la inicialización del contexto OpenGL ES y la carga de imágenes que se usarán más tarde por el sistema. Para el caso del servidor, el registro muestra la carga en memoria de los *scripts* y la IP y el puerto donde está escuchando conexiones de clientes.

Arrancado el sistema, el proyector muestra una imagen cargada desde disco por BelfegAR con información sobre un supuesto usuario ficticio (véase Figura 6.24).

6.5.3 Bucle principal

Pasado un tiempo (por defecto 10 segundos), el sistema avanza hacia el próximo estado en que entra en el bucle principal de detección de documentos, delegación de tareas y renderizado. Este estado es fácilmente reconocible por proyectar las cuatro esquinas del área de trabajo sobre la superficie (véase Figura 6.25).

En este caso, el usuario coloca la primera factura sobre la superficie de trabajo comprendida entre las cuatro esquinas. Tras esto, puede verse como el sistema renderiza sobre el documento el componente gráfico indicado en el *script* (véase Figura 6.26).

Después, el usuario coloca la segunda factura. Para este caso, el sistema renderiza otro componente gráfico correspondiente a esa factura. (véase Figura 6.27).

Para esta factura además, está definido que se inicie una videollamada al darle la vuelta al documento. La Figura 6.28 demuestra como el usuario le ha dado la vuelta al documento e inmediatamente la Figura 6.29 muestra como se han empezado a recibir fotogramas del servidor y son renderizados sobre el propio documento junto a una imagen de fondo. Además, la Figura 6.30 muestra los fotogramas recibidos en el servidor.

Volviendo al registro de eventos, la Figura 6.31 y la Figura 6.32 muestran respectivamente los registros de eventos del cliente y del servidor mientras se ejecuta el bucle principal del sistema.

Finalmente, la Figura 6.33 muestra el cierre del cliente de manera limpia liberando la memoria del sistema y cerrando los subsistemas adecuados.

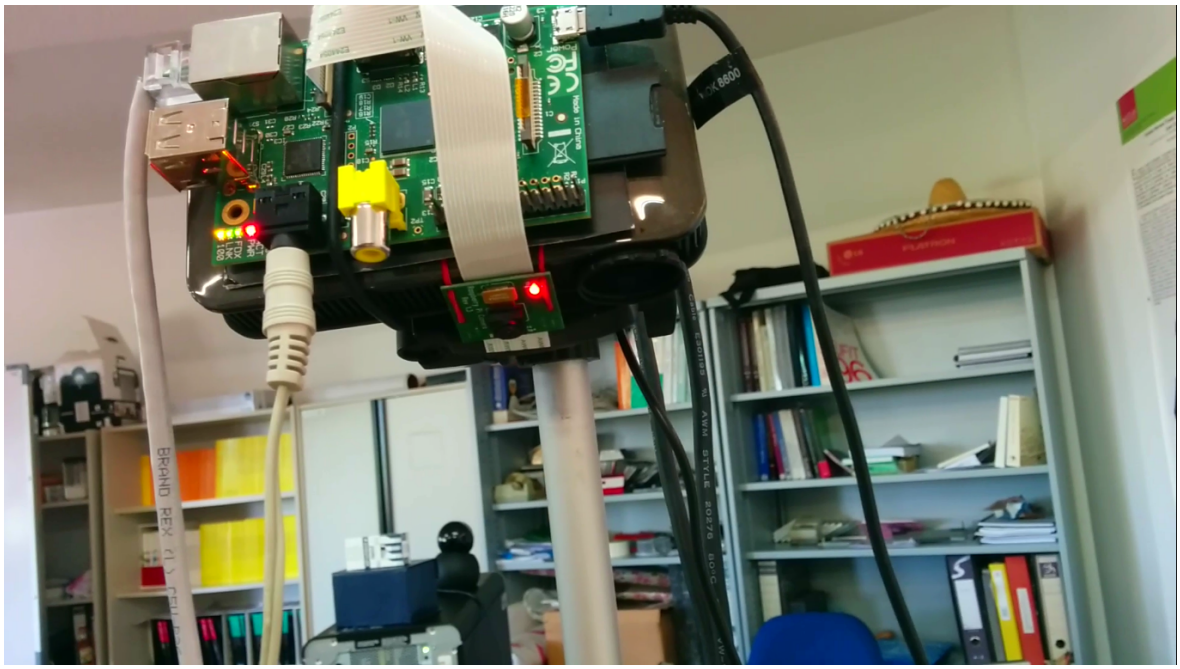

Figura 6.21: Hardware de ARgos.

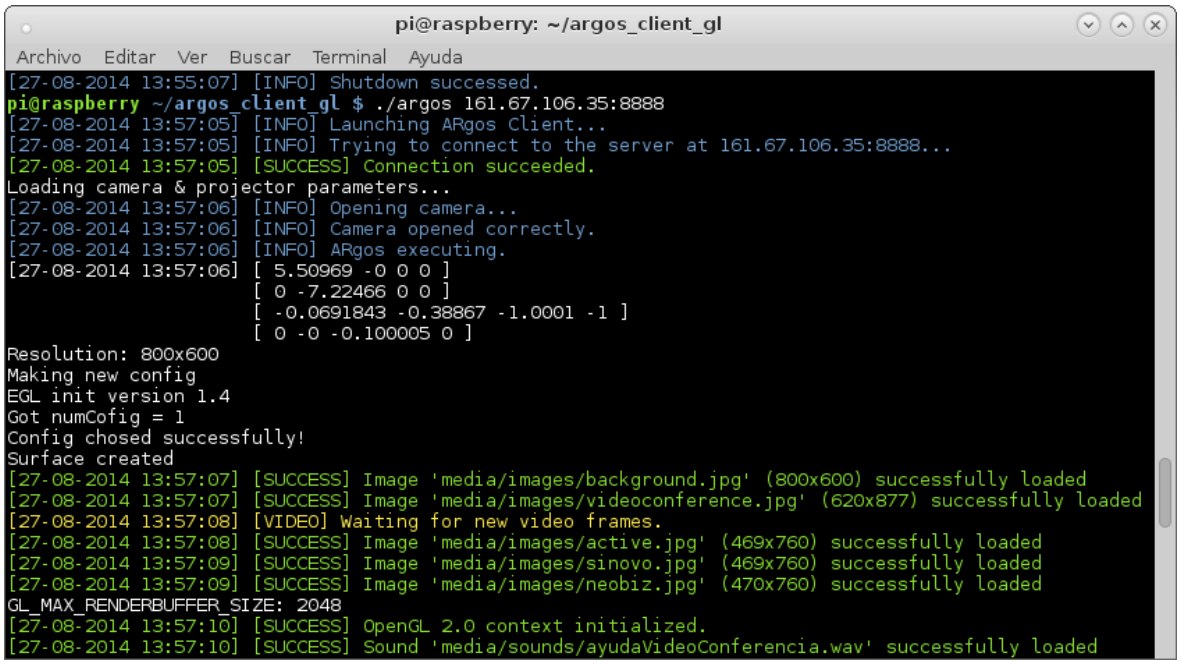

Figura 6.22: Inicialización del cliente.

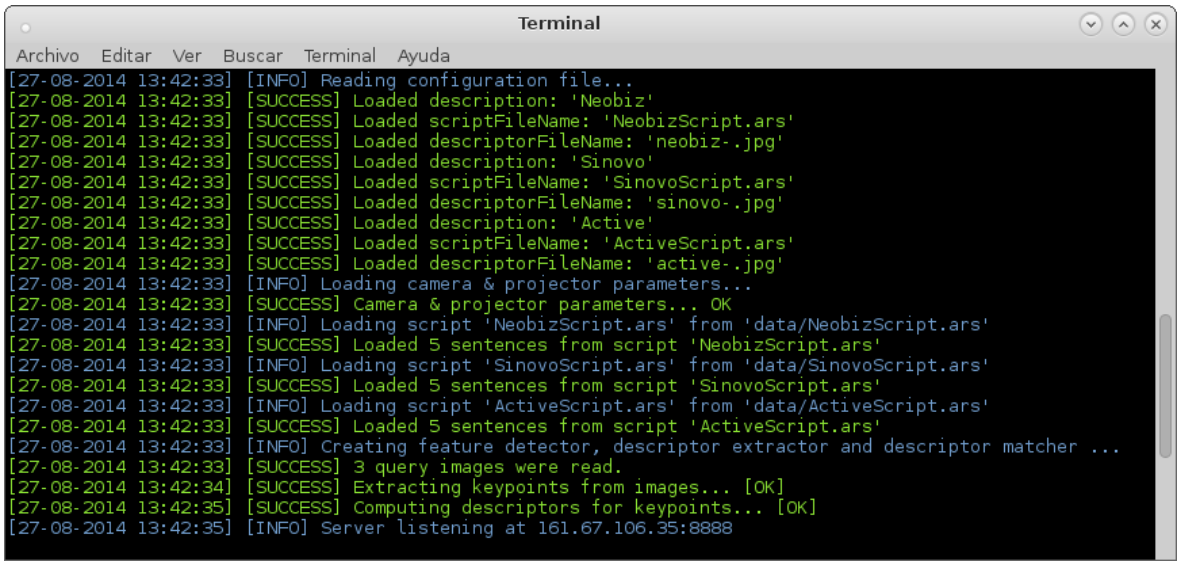

Figura 6.23: Inicialización del servidor.

Figura 6.24: Arranque de ARgos.

Figura 6.25: ARgos esperando por documentos.

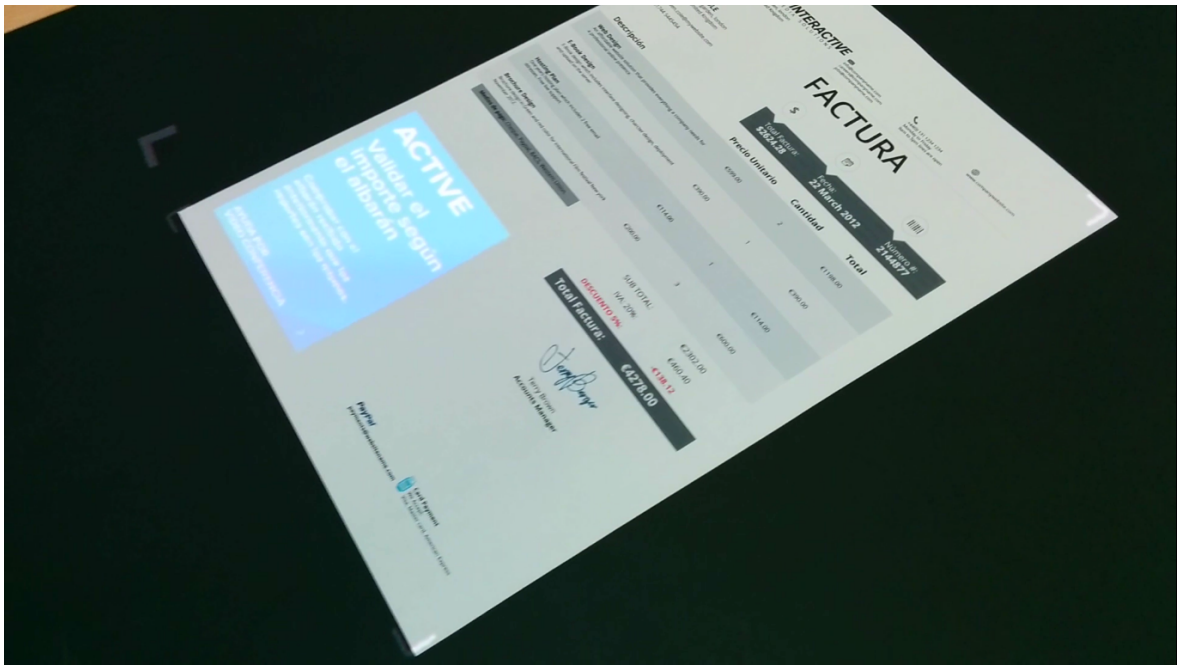

Figura 6.26: Primera factura detectada y componente gráfico desplegado.

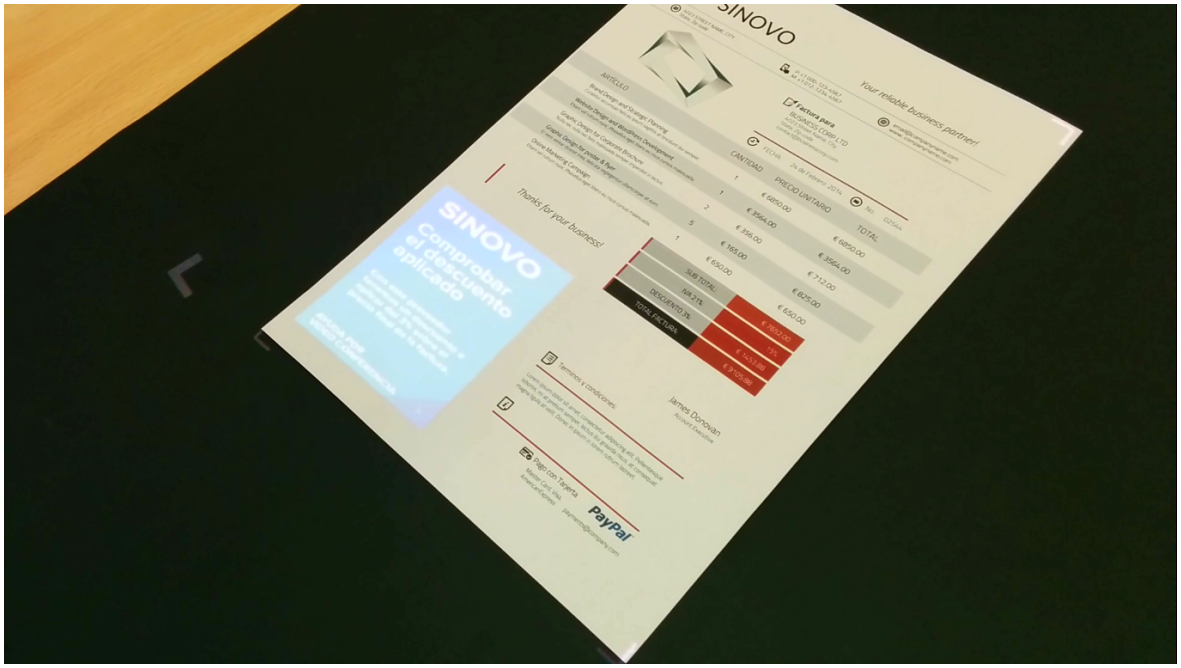

Figura 6.27: Segunda factura detectada y componente gráfico desplegado.

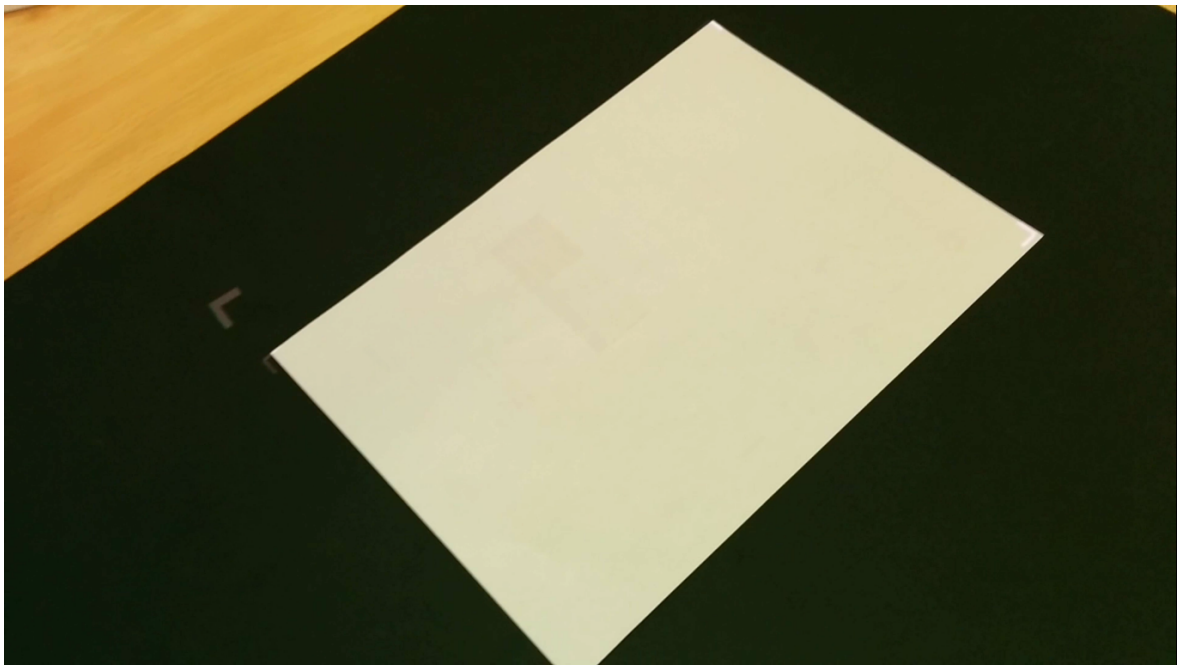

Figura 6.28: Factura dada la vuelta para iniciar la videollamada.

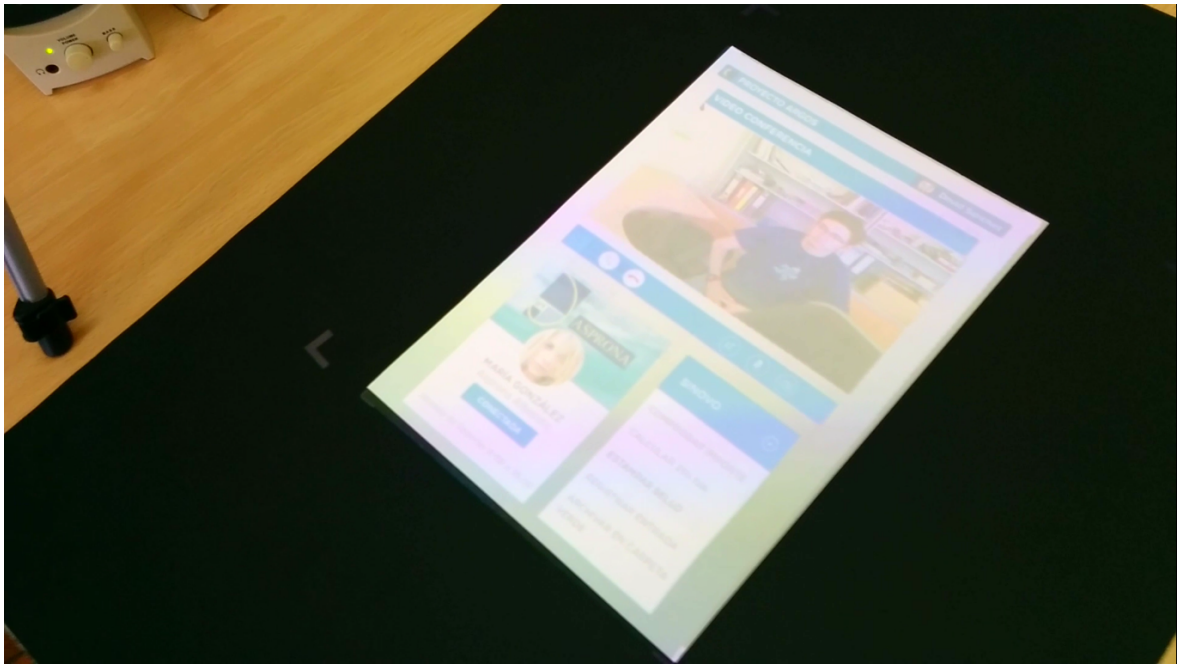

Figura 6.29: Videollamada iniciada sobre la superficie del documento.

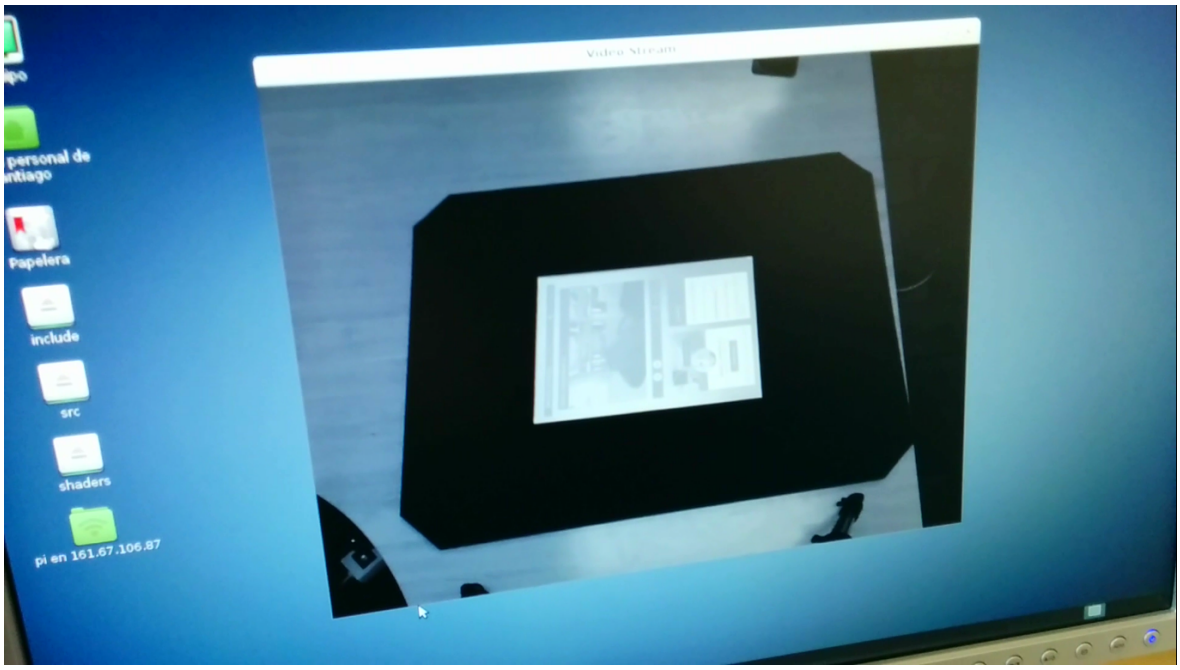

Figura 6.30: Videollamada en la parte del servidor.

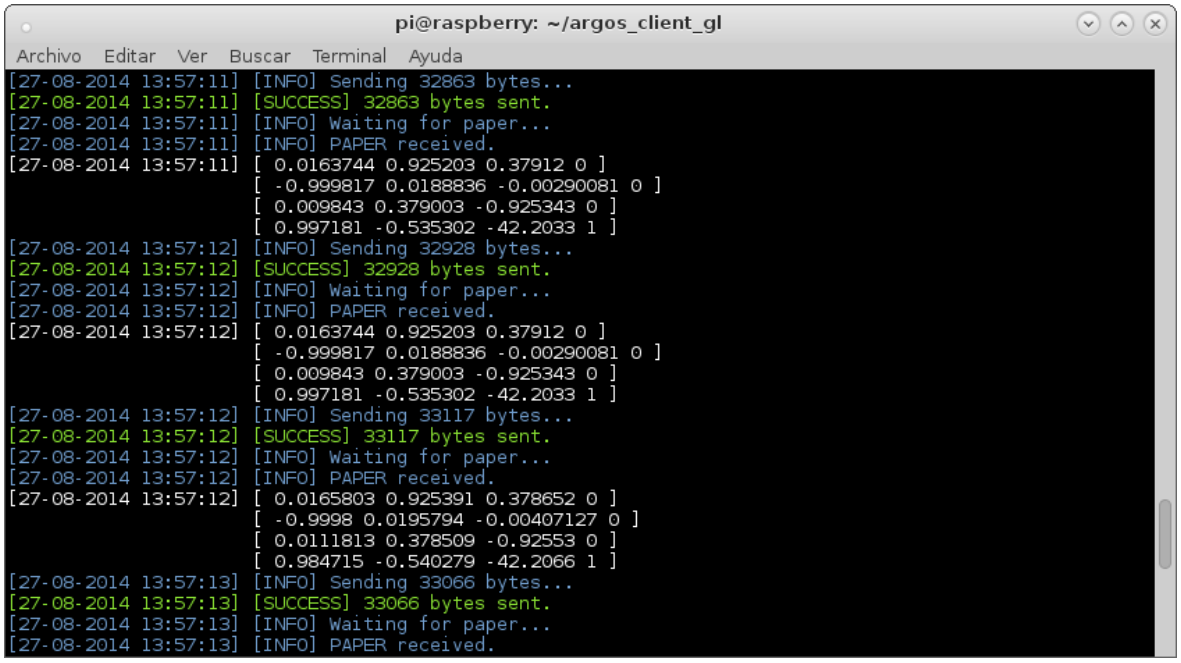

Figura 6.31: Registro de eventos del cliente mientras se ejecuta el bucle principal del sistema.

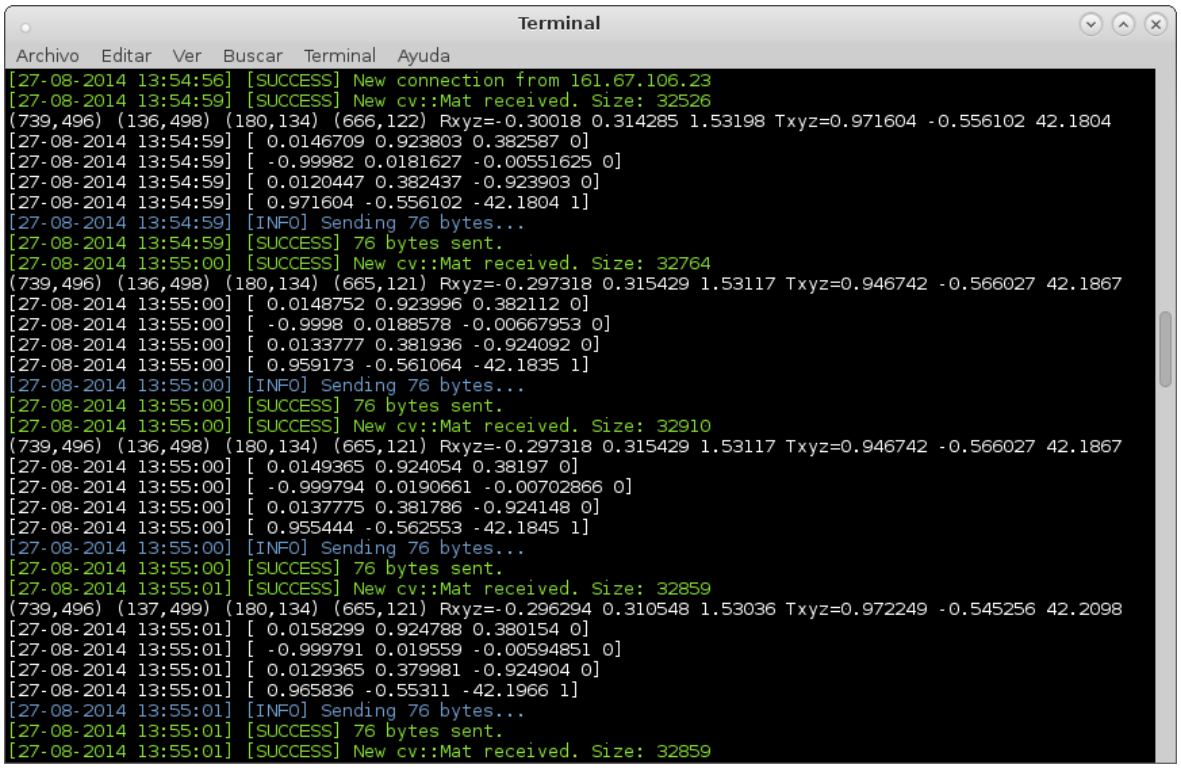

Figura 6.32: Registro de eventos del servidor mientras se ejecuta el bucle principal del sistema.

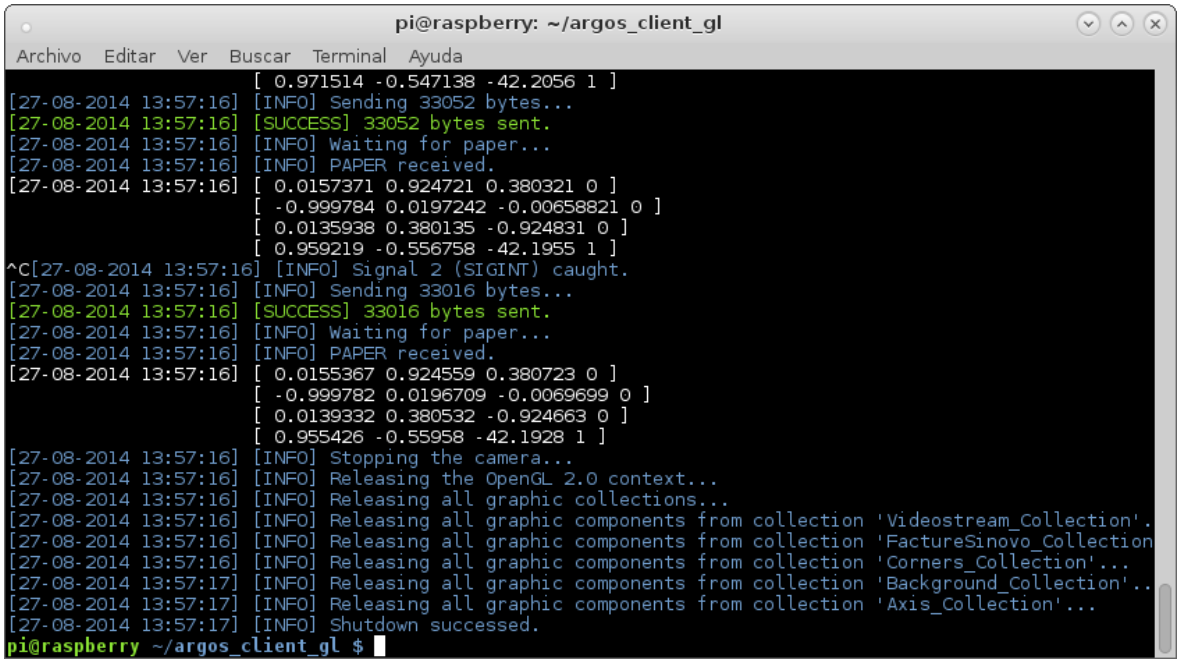

Figura 6.33: Cierre de subsistemas y liberación de la memoria del cliente.

Capítulo 7

Conclusiones

En el desarrollo de BelfegAR se han resuelto satisfactoriamente los requisitos principales establecidos en el Capítulo 6. En este capítulo se detallarán las conclusiones obtenidas del desarrollo de este proyecto y se propondrán posibles líneas de trabajo futuras a desarrollar.

7.1 Objetivos cumplidos

De acuerdo a los objetivos establecidos en el Capítulo 2 todos han sido cumplidos correctamente.

El objetivo principal de construir una plataforma para facilitar el despliegue de gráficos 3D en dispositivos empotrados de bajo coste para el soporte de gestión documental con realidad aumentada ha sido realizado gracias al cumplimiento de todos los objetivos específicos.

El **desarrollo de componentes gráficos** se ha llevado a cabo satisfactoriamente desarrollando un *framework* propio que proporciona un amplio conjunto de *widgets* de despliegue 3D de alto nivel que se basa en algunas bibliotecas externas como *freetypeGlesRpi* para el renderizado de fuentes y SOIL para la carga de imágenes. Se ha diferenciado entre componentes gráficos simples, fáciles de comprender para el usuario, y componentes gráficos más complejos formados como composición de los primeros. Además, el planteamiento modular que se ha seguido en su desarrollo asegura una fácil escalabilidad en caso de querer añadir nuevos componentes. Así mismo, se han superado satisfactoriamente los retos de emplear OpenGL ES realizando ciertas optimizaciones como el uso de listas de vértices e índices para su dibujado y el uso de la API EGL para obtener una superficie de renderizado acelerada por hardware.

Los **subsistemas de delegación de tareas y comunicación** se han implementado mediante una arquitectura de red basada en *sockets* TCP para mantener la comunicación entre el cliente y el servidor y *sockets* UDP orientados a garantizar la eficiencia en la comunicación para durante las videollamadas. La implementación de *sockets* ha sido llevada a cabo empleando la biblioteca *Boost.Asio*.

La **reproducción de audio** es llevada a cabo de forma centralizada y realizando la carga de los archivos desde disco. Todos los subobjetivos para el subsistema de audio han sido

cumplidos satisfactoriamente. La implementación del subsistema ha sido basada, casi completamente, alrededor de la biblioteca *SDL_Mixer* y la limitación física impuesta por la unidad de procesamiento para la reproducción de audio se ha solventado instalando un altavoz interno.

El **subsistema de *scripts*** basa su desarrollo en la definición del lenguaje propio ARS, el cual permite definir y asignar nuevos *scripts* de comportamiento que serán cargados durante la detección del propio documento. El propio lenguaje ARS está orientado a la facilidad de uso para el usuario, combinando un lenguaje de sencilla sintaxis con la potencia necesaria definida en los requisitos funcionales del proyecto ARgos, manteniendo además una simplicidad que no sería posible realizando la integración de un completo lenguaje de *scripting* como Python o Lua.

La **estabilidad del sistema** queda patente mediante un sistema de manejo de excepciones y valores de retorno que permiten recuperar el sistema ante cualquier tipo de error no crítico. La arquitectura de red también se beneficia de esta aproximación, reconectando al cliente en caso de que se pierda la conexión con el servidor. Además, el registro de eventos proporciona un informe al usuario de cómo ha funcionado el sistema durante su ejecución y le notifica los errores que hayan ocurrido.

El proyecto desarrollado basa su implementación en varios **patrones de diseño**, como «Singleton», «Builder» y «Adapter», además de algunos *idioms* de C++ para asegurar la eficiencia, mantenibilidad y futura adaptación a otros proyectos desarrollados con la misma base tecnológica. Además, el uso del nuevo estándar C++11 proporciona una implementación moderna y fácilmente escalable del sistema. El uso de un diseño orientado al *idiom* «Adquirir Recursos es Inicializar» (RAII de su término en Inglés Resource Acquisition Is Initialization) garantiza la reserva y liberalización de recursos, utilizado en BelfegAR mediante la apertura de ficheros, punteros inteligentes y sincronización multihilo.

Todas las **bibliotecas y herramientas** externas empleadas para el desarrollo de BelfegAR poseen una **licencia libre** y son **multiplataforma**, por lo que la migración a otros sistemas y arquitecturas supondría mínimos cambios en el código fuente del proyecto.

Gracias a lo anterior, los usuarios ven superadas sus posibles necesidades especiales mediante el uso de soporte externo, sonidos y el despliegue de componentes gráficos a modo de guía.

7.2 Lineas de trabajo futuro

Como se dijo al principio de este documento, BelfegAR junto a GrayAR, pertenecen a un proyecto mayor envergadura llamado ARgos, de la Cátedra Indra-UCLM. Dicho proyecto finaliza su ejecución en Noviembre de 2014. Hasta entonces, se deben de cumplir algunos requisitos que quedan fuera del alcance de este Trabajo de Fin de Grado.

Entre los requisitos propuestos se encuentran:

- **Mejora del subsistema de *scripts*:** Actualmente el subsistema no cuenta con ningún tipo de lógica; sólo se limita a ejecutar secuencialmente las instrucciones que encuentra en los *scripts*, y la lógica viene contenida en la propia función C++ equivalente a esa sentencia de *script*.

  Como mejora se pretende añadir una lógica mínima al subsistema, que permita decidir ejecutar una sentencia o no dependiendo del valor de retorno de esa función. Se trata por tanto de añadir nuevas características al subsistema que permitan hacer lo anterior. El coste temporal de dicha implementación sería de 1 semana aproximadamente.

  Si los requisitos del sistema aumentaran hasta el punto de que los cambios realizados al subsistema de *scripts* para cumplirlos fueran demasiado grandes, se optaría finalmente por utilizar un lenguaje de *scripting* completo como Python, empleando para ello la biblioteca `Boost.Python`. En este caso se tardaría hasta 3 semanas en realizar dicha implementación, ya que habría que migrar todos los *scripts* realizados en ARS a Python, y generar un nuevo manual de usuario.

  El impacto sobre GrayAR sería mínimo, ya que simplemente se tendría que cambiar la implementación de la carga de los *scripts* desde archivos XML, a los requisitos de la biblioteca `Boost.Python`.

- **Componentes orientados a datos:** Los componentes gráficos más complejos deberán ser creados mediante archivos externos interpretados por el motor, de tal forma que el usuario disponga de la posibilidad de crearlos indicando sus colores, formas, tamaño, textos, y posiciones. Se realizará una implementación mediante archivos XML ubicados y procesados en el cliente.

  Los archivos de componentes serán almacenados al comienzo de la ejecución en memoria para su posterior creación, provocando de esta forma que el sistema no se vea afectado durante su uso.

  El coste temporal de implementación sería de 1 semana; solo habría que añadir al gestor gráfico un método que interprete dichos archivos XML y construya los nuevos componentes gráficos como `RenderToTextureComponent` en base a los datos obtenidos.

  Un posible ejemplo de componente gráfico complejo definido en un archivo externo se puede ver en el Listado 7.1.

```
<ComplexGraphicComponent name="Button1">
  <Rectangle x="10" y="20" w="20" h="15" colour="CCCCCC" />
  <Text x="12" y="22" colour="000000" text="Press me!" />
</ComplexGraphicComponent>
```

Listado 7.1: Posible implementación de un componente gráfico complejo definido en un XML.

El impacto de esta mejora sobre GrayAR sería prácticamente nulo, ya que GrayAR no posee ninguna competencia relacionada con la representación gráfica del sistema.

- **Decodificación de vídeo por hardware:** Debido a limitaciones de tiempo y a la escasa documentación existente, la decodificación de vídeo en BelfegAR es realizada por software, afectando dramáticamente al rendimiento general de la aplicación durante la reproducción de vídeos.

  Como mejora, se estudiará el uso de la API OpenMAX que permite la decodificación de vídeo por hardware. Una vez implementada la decodificación de los fotogramas, éstos deberán ser copiados a una textura de OpenGL para poder aplicar las transformaciones del espacio 3D.

  Sin embargo como hemos dicho, la documentación existente es muy escasa y la interfaz de OpenMAX no es demasiado amigable de cara al programador. Aun así, este requisito se consolida como uno de los más prioritarios de cara al futuro.

  El coste temporal de esta mejora podría ser de hasta 3 semanas, debido a la necesidad de estudio de la implementación de la propia biblioteca a bajo nivel.

  El impacto de esta mejora sobre GrayAR vuelve a ser inexistente, ya que no interfiere con ninguna parte de GrayAR de forma directa.

- **Interfaz de usuario:** Se creará una interfaz de usuario web, que permita la creación visual de componentes gráficos complejos, la elaboración de *scripts*, la configuración de ARgos de manera sencilla y el envío de archivos como vídeos o imágenes a la unidad de procesamiento de forma automática para el usuario. La aplicación se desarrollará empleando el marco de trabajo Django por estar basado en Python y tener una base de conocimientos del lenguaje.

  El coste temporal de esta implementación sería de hasta 1 mes, ya que habría que realizar un estudio del marco de trabajo de Django para poder implementar el sistema web.

  El impacto sobre GrayAR se resumiría en la generación de archivos de configuración XML a través del sistema web que serían cargados por GrayAR. Esto sin embargo se haría de forma transparente por lo que no supondría ningún cambio en la base de GrayAR.

Capítulo 8

Conclusions

In the BelfegAR development it has been resolved the main requirements stated in Chapter 6 successfully. In this chapter it will be detailed the conclusions obtained in the development of this project as well as it will be proposed some future working lines to be developed.

8.1 Accomplished objectives

According to the objectives established in Chapter 3 all of them have been accomplished correctly.

The main goal of building a platform to facilitate the deployment of 3D graphics in low-cost embedded devices to support document management with augmented reality has been made through the accomplishment of all the subgoals.

The **development of graphic components** has been carried out successfully building a framework which provides a wide range of widgets for 3D deployment in a high level and based on some external libraries such as *freetypeGlesRpi* for font rendering and SOIL for image loading. It was differentiated between simple graphic components, easy to understand for the user, and more complex graphic components built from a composition of the former. Furthermore, the modular approach followed for its development assures a suitable scalability in case of adding new components. Hence, the challenges of using OpenGL ES have been accomplished doing some optimizations such as using vertex arrays and indexes for rendering and using the EGL API to obtain a hardware accelerated rendering surface.

The **communication and tasks delegation subsystems** have been implemented through a network architecture based on **TPC!** *sockets* to hold the communication between client and server, and UDP *sockets* oriented to guarantee the efficiency the communication during videoconferences. The *sockets* implementation has been carried out using the *Boost.Asio* library.

Audio playback is carried out in a centralised way and loading files from disk. All the subgoals for the audio subsystem have been accomplished successfully. The subsystem implementation has been based, almost completely, around the *SDL_Mixer* library and the physical limitation for audio playback has been resolved installing an on-board speaker.

The **scripting subsystem** bases its development building a new language named ARS which allows to define and assigning new behavioural scripts that will be loaded during document detection. The ARS language is well-formed for easy-using to the user, mixing a simple syntax language and the needed power defined in the functional requirements by the ARgos project; maintaining a simplicity that would not be possible by integrating a whole scripting language like Python or Lua.

The **system stability** is proved through an exception handling system and return values that allows system recovery to face any kind of non-critical error. The network architecture also benefits from this approach, reconnecting the client in case of losing the connection with the server. Furthermore, the events log provides a report to the user about how the system has been running and it notifies the errors that happened.

The developed project bases its implementation in a variety of **design patterns** such as «Singleton», «Builder», and «Adapter», and some C++ *idioms* to assure the efficiency, maintainability, and future projects adaptations developed with the same technology basis. Moreover, the using of the new C++11 standard provides a modern implementation and easily scalable system. The using of a design oriented to Resource Acquisition Is Initialization (RAII) guarantees the acquisition and release of resources in BelfegAR through opening files, smart pointers, and multithreaded synchronisation.

All the external **libraries and tools** used in the development of BelfegAR are **crossplatform** and **free software**, so the transition to other systems and architectures would suppose minimal changes in the source code.

Thanks to this, users get over their possible special needs through external support, sounds, and graphic components deployment as a guide.

8.2 Future working lines

As it was stated previously in this document, BelfegAR and GrayAR, belong to a bigger project named ARgos, from *Cátedra Indra-UCLM*. This project finishes its execution in November, 2014. Until then, it must be accomplished some requirements that lie out this Final Degree Project.

Among the proposed requirements there are:

- **Scripts subsystem improvement:** Currently the subsystem does not count with any kind of logic; it only limits itself to run sentences sequentially found in the scripts, and the logic is stored in the own C++ function analogous to the script sentence.

  As an improvement it is pretended to add a minimal logic to the subsystem, that allows to decide whether running a sentence or not, depending on the return value of that function. This way, new features should be added to accomplish the previous requirement. The time cost of this implementation would be about 1 week.

If the requirements of the system got increased until the changes made to the scripts subsystem to accomplish them were too big, finally it would choose using a whole scripting language like Python, using the `Boost.Python` library to carry it out. In this case the implementation would take 3 weeks to be done, because all the scripts should be translated from ARS to Python, and make a new user manual.

The impact over GrayAR would be minimal, because it simply would has to change the script loading implementation from XML, to the `Boost.Python` needs.

- **Data-driven components:** The complex graphic components will have to be built from external files read by the engine, so the user can make them specifying their colours, shapes, size, texts, and locations. It will be done an implementation through XML files located and processed in the client.

  The components files will be loaded at the beginning of the execution in memory for their later creation, causing, this way, the system will not be affected during their using.

  The implementation time cost would be of 1 week; it would only has to add to the graphic manager a new method that reads these XML files and build up the new graphic components as `RenderToTextureComponent` based on the read data.

  A possible example of a complex graphic component defined in an external file can be seen in the Listing 8.1.

```
1 <ComplexGraphicComponent name="Button1">
2   <Rectangle x="10" y="20" w="20" h="15" colour="CCCCCC" />
3   <Text x="12" y="22" colour="000000" text="Press me!" />
4 </ComplexGraphicComponent>
```

Listado 8.1: Possible implementation of a complex graphic component defined in an XML.

  The impact of this improvement on GrayAR would be almost null, because GrayAR does not own any competence related with graphical representation.

- **Hardware video decoding:** Due to time limits and the lack of existing documentation, video decoding in BelfegAR is carried out at software level, affecting dramatically to the overall performance of the application during video playback.

  As an improvement, it will be studied the using of the OpenMAX API that allows to video decode at hardware level. Once frame decoding is implemented, these would be copied to an OpenGL texture to later apply 3D transformations.

  However, as it was said, the lack existing documentation and the OpenMAX API is not very friendly for the developer. Even so, this requirement becomes established as one of the first importance for the future.

  The time cost of this improvement would be up to 3 weeks, due to the need of study the library implementation at low level.

The impact of this improvement on GrayAR would be again almost null, because it does not interfere directly with any part of GrayAR.

- **User interface:** It will be developed a web user interface, that allows drag-and-drop to make complex graphics components, script making, easily setup of ARgos, and sending files like video and images to the processing unit automatically for the user. The application will be developed using the Django framework for being based on Python and having a language knowledge base.

  The time cost of this implementation would be up to 1 month, because it would have to do a research of the Django framework to implement the web system.

  The impact on GrayAR would be summarised in generating XML configuration files through the web system that would be loaded by GrayAR. However, it would be done transparently so there would be no change in its basis.

ANEXOS

Anexo A

Características Raspberry Pi

SoC:	Broadcom BCM2835 (CPU + GPU + DSP + SDRAM + puerto USB)
CPU:	ARM 1176JZF-S a 700 MHz (familia ARM11)
Juego de instrucciones:	RISC de 32 bits
GPU:	Broadcom VideoCore IV, OpenGL ES 2.0, MPEG-2 y VC-1 (con licencia), 1080p30 H.264/MPEG-4 AVC
Memoria (SDRAM):	512 MiB (compartidos con la GPU)
Puertos USB 2.0:	2 (vía hub USB integrado)
Entradas de vídeo:	Conector MIPI CSI que permite instalar un módulo de cámara desarrollado por la RPF
Salidas de vídeo:	Conector RCA (PAL y NTSC), HDMI (rev1.3 y 1.4), Interfaz DSI para panel LCD
Salidas de audio:	Conector de 3.5 mm, HDMI
Almacenamiento integrado:	SD / MMC / ranura para SDIO
Conectividad de red:	10/100 Ethernet (RJ-45) via hub USB
Periféricos de bajo nivel:	8 x GPIO, SPI, I2C, UART
Reloj en tiempo real:	Ninguno
Consumo energético:	700 mA, (3.5 W)
Fuente de alimentación:	5 V vía Micro USB o GPIO header
Dimensiones:	85.60mm x 53.98mm (3.370 x 2.125 inch)
Sistemas operativos soportados:	GNU/Linux: Debian (Raspbian), Fedora (Pidora), Arch Linux (Arch Linux ARM), Slackware Linux. RISC OS

Cuadro A.1: Características del modelo (B) de Raspberry Pi empleado en el sistema [Wik].

Anexo B

Compilación cruzada con Scratchbox 2

En este anexo se explicará en detalle como instalar y configurar una solución de compilación cruzada basada en el entorno *Scratchbox 2*, que nos permita prescindir de una plataforma *Raspberry Pi* real y aprovechar de esta forma los recursos superiores de nuestra máquina de desarrollo a la hora de realizar la compilación.

B.1 Instalación del entorno de compilación cruzada

El entorno de compilación cruzada requiere de algunas dependencias para su correcto funcionamiento. Cabe destacar, que las instrucciones indicadas a continuación están orientadas a distribuciones GNU/Linux basadas en Debian.

B.1.1 Instalación de dependencias

La lista de paquetes de los que depende el entorno es la siguiente:

```
git-core
build-essential
autoconf
fakeroot
realpath
qemu-user
zlib1g-dev
libglib2.0-dev
gcc-arm-linux-gnueabi
g++-arm-linux-gnueabi
gcc-multilib
libpixman-1-dev
```

Dichas dependencias pueden ser instaladas empleando la aplicación `apt-get` de la terminal. Antes de ello, se debe añadir el repositorio que contiene los binarios de GCC para compilar contra la arquitectura ARM.

El Listado B.1 muestra los comandos que se deben ejecutar por terminal para la correcta instalación de todas las dependencias. Nota: los comandos que empiecen por un $ deben ser ejecutados como usuario, mientras que los que empiecen por # deberán ser ejecutados como superusuario.

```
# echo "deb http://www.emdebian.org/debian/ squeeze main" \
    >> /etc/apt/sources.list.d/emdebian.list
# apt-get update

# apt-get install git-core build-essential autoconf fakeroot \
    realpath qemu-user zlib1g-dev libglib2.0-dev \
    gcc-arm-linux-gnueabi g++-arm-linux-gnueabi gcc-multilib \
    libpixman-1-dev
```

Listado B.1: Instalación de dependencias.

B.1.2 Instalación de Scratchbox 2

La instalación del conjunto de herramientas de *Scratchbox 2* se detalla en el Listado B.2.

```
$ mkdir $HOME/RaspberryPi && cd $HOME/RaspberryPi
$ git clone git://gitorious.org/scratchbox2/scratchbox2.git

$ cd scratchbox2
$ ./autogen.sh
$ make install prefix=$HOME/RaspberryPi/sb2
```

Listado B.2: Instalación de Scratchbox 2.

B.1.3 Instalación de QEMU

Scratchbox 2 depende de *QEMU* para emular la arquitectura ARM. La instalación de *QEMU* se detalla en el Listado B.3.

```
$ cd $HOME/RaspberryPi
$ git clone git://git.qemu.org/qemu.git

$ cd qemu
$ ./configure --prefix=$HOME/RaspberryPi/sb2 \
    --target-list=arm-linux-user
$ make
# make install
```

Listado B.3: Instalación de QEMU.

B.2 Configuración del entorno

Una vez instalado el entorno solo queda realizar las configuraciones pertinentes.

B.2.1 Configuración local

Para que todo funcione correctamente, se deben realizar algunos ajustes para poder trabajar de forma satisfactoria con nuestro sistema. El Listado B.4 muestra los comandos a ejecutar para la correcta configuración del sistema.

```
$ echo 'export PATH="$HOME/RaspberryPi/sb2/bin:$PATH"' \
    >> $HOME/.bashrc
$ source $HOME/.bashrc

$ cd $HOME/RaspberryPi
$ mkdir wheezy && mkdir devel
$ wget http://downloads.raspberrypi.org/raspbian_latest
$ unzip raspbian_latest

$ export OFFSET=\
    $(printf "%d * 512\n" $(/sbin/fdisk -lu *-raspbian.img | \
    grep Linux | awk '{print $2}') | bc)
# mount -o loop,offset=$OFFSET $HOME/RaspberryPi/*-raspbian.img \
    $HOME/RaspberryPi/wheezy
# chown $USER:$GROUPS -R wheezy
# cp -a $HOME/RaspberryPi/wheezy/{bin,etc,home,lib,media,mnt,opt,\
    root,run,sbin,selinux,srv,sys,tmp,usr,var} $HOME/RaspberryPi/devel

$ cd $HOME/RaspberryPi/devel
$ sb2-init RaspDev /usr/bin/arm-linux-gnueabi-gcc
$ sed -i -e 's/^/#/' $HOME/RaspberryPi/devel/etc/ld.so.preload

# umount $HOME/RaspberryPi/wheezy
$ rm -rf $HOME/RaspberryPi/wheezy
$ rm -rf $HOME/RaspberryPi/*-raspbian.img
```

Listado B.4: Configuración local del sistema.

B.2.2 Configuración de la Raspberry Pi virtual

Finalmente, conseguimos disponer de una plataforma Raspberry Pi virtual sobre la que trabajar de forma emulada.

Solo necesitamos conocer dos comandos para trabajar con nuestro nuevo sistema. Dichos comandos pueden ser usados de forma aislada, lanzando de esta forma la ejecución de programas de forma interactiva, o pasándole dichos programas como parámetros durante la ejecución de los comandos:

- `sb2 -e` permite ejecutar programas de forma emulada por la Raspberry Pi virtual.
- `sb2 -eR` permite ejecutar programas como **super usuario**, de forma emulada por la Raspberry Pi virtual.

A partir de aquí y desde el propio programa de configuración que incluye la distribución *Raspbian* que se ha instalado de forma virtual para simular una Raspberry Pi, se recomienda actualizar dicho configurador (`Advanced Options` → `Update`) y ajustar la localización (`Internationalisation Options` → `Change Locale`). Por último, es también recomendable actualizar la lista de repositorios y el propio sistema en si mediante `apt-get`. El Listado B.5 indica los comandos a utilizar.

```
$ sb2 -eR raspi-config
$ sb2 -eR apt-get update
$ sb2 -eR apt-get upgrade
```

Listado B.5: Actualización del nuevo sistema Raspberry Pi emulado.

Una vez actualizado el sistema y siguiendo la misma mecánica, se puede instalar cualquier paquete mediante `apt-get` o compilar cualquier programa mediante `make` como si nos encontráramos en una Raspberry Pi real. Después de compilar y una vez generado el ejecutable, solo debe de ser copiado a la Raspberry Pi física para su ejecución, mediante el programa `scp` por ejemplo.

B.3 Script de instalación automático

Aunque las instrucciones anteriores funcionan correctamente, se proporciona además un *script* `bash` para instalar todo el entorno de forma automática. El *script* debería de funcionar correctamente en cualquier distribución GNU/Linux basada en Debian. Si se presenta algún error durante el proceso, se recomienda entonces intentar la instalación manual.

El Listado B.6 contiene el código fuente del *script* de instalación.

```
1  #!/usr/bin/env bash
2
3  yellow='\E[1;33m'
4  blue='\E[1;34m'
5
6  echo -e "$blue"
7
8  echo "#################################################################"
9  echo "## Este script intentará instalar todo el entorno de          ##"
10 echo "## compilación cruzada usando Scratchbox 2 para proporcionar ##"
11 echo "## una configuración simulada de una Raspberry Pi virtual    ##"
12 echo "## ---------------------------------------------------------  ##"
13 echo "## Autor: Santiago Sánchez Sobrino                           ##"
14 echo "## Fecha: 07/03/2014                                         ##"
15 echo "## NOTA: Este script se encuentra en fase experimental,      ##"
16 echo "##       por lo que si tras un primer uso falla, se          ##"
17 echo "##       recomienda la instalación manual.                   ##"
18 echo "#################################################################"
19
20 echo -e "$yellow"
21
22 if [ $# -eq 0 ]; then
23     echo "Ejecutar como: ./install.sh <install_dir>"
24     exit 1
25 fi
26
27 if [ -d "$1" ]; then
```

```
28                  echo "El directorio '$1' ya existe. Elija otro."
29                  exit 1
30  fi

32  progressfilt ()
33  {
34      local flag=false c count cr=$'\r' nl=$'\n'
35      while IFS='' read -d '' -rn 1 c
36      do
37          if $flag
38          then
39              printf '%c' "$c"
40          else
41              if [[ $c != $cr && $c != $nl ]]
42              then
43                  count=0
44              else
45                  ((count++))
46                  if ((count > 1))
47                  then
48                      flag=true
49                  fi
50              fi
51          fi
52      done
53  }

55  max_steps=13
56  step=1
57  target=${1%/}

59  echo "[$step/$max_steps] > Actualizando lista de paquetes..."
60  sudo apt-get update -q=2
61  ((step++))

63  echo "[$step/$max_steps] > Instalando paquetes..."
64  sudo apt-get install -y git-core build-essential autoconf fakeroot
       realpath qemu-user zlib1g-dev libglib2.0-dev gcc-arm-linux-gnueabi
        g++-arm-linux-gnueabi gcc-multilib libpixman-1-dev >/dev/null
65  ((step++))

67  echo "[$step/$max_steps] > Creando directorio de instalación en '
       $target'..."
68  mkdir $target && cd $target
69  ((step++))

71  echo "[$step/$max_steps] > Descargando Scratchbox 2..."
```

```
72 git clone -q git://gitorious.org/scratchbox2/scratchbox2.git
73 ((step++))

75 echo "[$step/$max_steps] > Instalando Scratchbox 2 en '$target/sb2'...
       "
76 cd scratchbox2 && ./autogen.sh >/dev/null
77 make install prefix=$target/sb2 >/dev/null 2>&1
78 cd $target
79 ((step++))

81 echo "[$step/$max_steps] > Descargando qemu..."
82 git clone -q git://git.qemu.org/qemu.git
83 ((step++))

85 echo "[$step/$max_steps] > Instalando qemu..."
86 cd qemu && ./configure --prefix=$target/sb2 --target-list=arm-linux-
       user >/dev/null 2>&1
87 make >/dev/null 2>&1 && sudo make install >/dev/null 2>&1
88 ((step++))

90 echo "[$step/$max_steps] > Configurando variables de entorno..."
91 echo 'export PATH="$target/sb2/bin:$PATH"' >> $HOME/.bashrc
92 source $HOME/.bashrc
93 ((step++))

95 echo "[$step/$max_steps] > Descargando imagen de Raspbian..."
96 mkdir $target/wheezy && mkdir $target/devel
97 mkdir $target/temp && cd $target/temp
98 wget --progress=bar:force http://downloads.raspberrypi.org/
       raspbian_latest 2>&1 | progressfilt
99 ((step++))

101 echo "[$step/$max_steps] > Descomprimiento Raspbian..."
102 cd $target/temp
103 unzip -qq raspbian_latest
104 ((step++))

106 echo "[$step/$max_steps] > Copiando archivos de Raspbian a '$target/
       devel'..."
107 cd $target
108 OFFSET=$(printf "%d * 512\n" $(/sbin/fdisk -lu $target/temp/*-raspbian
       .img | grep Linux | awk '{print $2}') | bc)
109 sudo mount -o loop,offset=$OFFSET $target/temp/*-raspbian.img $target/
       wheezy
110 sudo chown $USER:$GROUPS -R wheezy
111 cp -a $target/wheezy/{bin,etc,home,lib,media,mnt,opt,root,run,sbin,
       selinux,srv,sys,tmp,usr,var} $target/devel
```

```
112 ((step++))

114 echo "[$step/$max_steps] > Configurando el entorno de Scratchbox 2..."
115 cd $target/devel
116 sb2-init RaspDev /usr/bin/arm-linux-gnueabi-gcc >/dev/null 2>&1
117 sed -i -e 's/^/#/' $target/devel/etc/ld.so.preload
118 sudo umount $target/wheezy
119 rm -rf $target/wheezy
120 rm -rf $target/temp
121 ((step++))

123 echo "[$step/$max_steps] Terminado."
```

Listado B.6: *Script* automatizado de instalación del entorno de compilación cruzada.

Anexo C

Código fuente

C.1 Cliente

Para la aplicación cliente, es decir, aquella que se ejecuta en la Raspberry Pi, proyecto presenta la siguiente estructura de directorios

- **include:** Contiene los archivo de cabecera o *headers* C++ del proyecto.
- **libs:** Contiene las biblioteca externas `freetypeGlesRpi` y `glm`.
- **data:** Contiene los recursos del sistema divididos en subdirectorios:
  - **calibrations:** Contiene los archivos de calibración `calibrationCamera.xml`, `calibrationProjector.xml` y `CameraProjectorExtrinsics.xml`.
  - **fonts:** Contiene las fuentes *true-type* a usar por el sistema.
  - **images:** Contiene todas las imágenes del sistema.
  - **sounds:** Contiene todos los archivos de audio del sistema.
  - **videos:** Contiene todos los archivos de vídeo del sistema.
- **shaders:** Contiene todos los programas *vertex shader* y *fragment shader* usados por OpenGL ES 2.0.
- **src:** Contiene los archivos de código fuente C++ del proyecto.
- El directorio raíz contiene el *Makefile* necesario para la construcción del proyecto.

C.2 Servidor

Por otra parte, la aplicación servidora o aquella que se ejecuta en el computador externo presenta la siguiente estructura de directorios:

- **data:** Contiene el archivo de configuración principal `config.xml` y el resto de recursos del sistema divididos en subdirectorios:
  - **calibrations:** Contiene los archivos de calibración `calibrationCamera.xml`, `calibrationProjector.xml` y `CameraProjectorExtrinsics.xml`.
  - **scripts:** Contiene los archivos de *script* ARS del sistema.
  - **templates:** Contiene todas las plantillas de documentos que debe detectar el sistema.
- **include:** Contiene los archivo de cabecera o *headers* C++ del proyecto.
- **src:** Contiene los archivos de código fuente C++ del proyecto.
- El directorio raíz contiene el *Makefile* necesario para la construcción del proyecto.

Anexo D

Contenido del CD

En este anexo se detalla el contenido del cederrón que acompaña a este documento.

- **SRC:** Fuentes completos con la estructura detallada en el Anexo C.
- **Doc:** Contiene la documentación del proyecto, el PDF generado y los fuentes LaTeXpara su construcción.
- **Exec:** Directorios con los diferentes ejecutables generados para el cliente, el servidor y las utilidades externas.
- **Pruebas:** Primeros prototipos y programas que no forman parte de la distribución final.
- **Doxygen:** Documentación completa del sistema, tanto en HTML como en PDF.

Anexo E

Generación de archivos de audio hablado o TTS

Para la obtención de archivos de audio hablados o TTS, se ha creado un pequeño *script* en *bash* que se comunica con el servidor de Google y realiza una petición a su servicio de traducción en línea, obteniendo de esta forma la descarga del archivo de sonido correspondiente.

El Listado E.1 contiene un ejemplo de uso del *script*, mientras que el Listado E.4 contiene el código fuente del propio *script*.

```
$ ./tts.sh es "Hola Mundo!" hola_mundo.wav
```

Listado E.1: Ejemplo de ejecución del *script*.

El ejemplo genera un archivo `hola_mundo.wav` con el texto «Hola Mundo!» pronunciado por una voz humana.

Un incorrecto uso del *script* proporciona la salida del Listado E.2. Si se utiliza la opción `-h`, se explica al usuario la manera de usar el *script* (Listado E.3).

```
$ ./tts.sh
Ejecutar como: ./tts.sh [-h] <lang> <text> <filename>
```

Listado E.2: Ejemplo de ejecución incorrecta del *script*.

```
$ ./tts.sh -h
Ejecutar como: ./tts.sh [-h] <lang> <text> <filename>
<lang>
  Idioma de la voz soportada por el motor de Google.
  Algunas son: es (Español), de (Alemán), en (Inglés).

<text>
  El texto que queremos convertir a voz.

<filename>
  El nombre del archivo resultante, con la extensión.
```

Listado E.3: Ejemplo de ayuda del *script*.

Una vez generado el archivo, éste debe de ser copiado a la unidad de detección y despliegue (directorio `media/sounds`) para poder ser usado por BelfegAR. Después, simplemente

usando el gestor de audio directamente en C++ o a través del subsistema de *scripts*, el sistema permite su posterior reproducción.

Se ha decidido no mantener está funcionalidad en línea dentro del propio código de BelfegAR, para no obligar a la unidad de detección y despliegue a mantener una conexión a Internet. De esta forma, los usuarios pueden mantener una base de sonidos fuera de línea para ser reutilizados por el sistema.

```
1  #!/usr/bin/env bash
2
3  execLine() {
4      echo "Ejecutar como: ./tts.sh [-h] <lang> <text> <filename>"
5  }
6
7  if [ $# -eq 0 ]; then
8      execLine
9      exit 1
10 fi
11
12 if [ $# -lt 3 ]; then
13     if [ $1 = "-h" ]; then
14         execLine
15         echo "<lang>"
16         echo "  Idioma de la voz soportada por el motor de Google."
17         echo "  Algunas son: es (Español), de (Alemán), en (Inglés)."
18         echo ""
19         echo "<text>"
20         echo "  El texto que queremos convertir a voz."
21         echo ""
22         echo "<filename>"
23         echo "  El nombre del archivo resultante, con la extensión."
24         exit 1
25     fi
26     execLine
27     exit 1
28 fi
29
30 lang=$1
31 text=$2
32 filename=$3
33 request="http://translate.google.com/translate_tts?ie=UTF-8&tl=$lang&q
       =$text"
34
35 wget -q -U Mozilla -O $filename $request
```

Listado E.4: Código fuente del *script* de descarga de archivos de audio como TTS.

Anexo F

Manual de usuario

El presente anexo constituye un manual de usuario, que abarca desde la instalación del sistema hasta su uso.

F.1 Instalación del cliente

Suponiendo que contamos con una Raspberry Pi con la distribución Raspbian correctamente instalada, la instalación de la aplicación cliente se divide en dos fases; instalación de dependencias y compilación del sistema.

F.1.1 Dependencias

El sistema requiere las siguientes dependencias para su funcionamiento:

```
libx11-dev
libfreetype6-dev
libsoil-dev
libsdl-1.2-dev
libboost-all-dev
libsdl-mixer-1.2-dev
libopencv-dev
```

Dichas dependencias pueden ser instaladas mediante el uso del comando `apt-get`:

```
# apt-get install libx11-dev libfreetype6-dev libsoil-dev \
    libsdl-1.2-dev libboost-all-dev libsdl-mixer-1.2-dev \
    libopencv-dev
```

Para utilizar la cámara CSI se requiere además la biblioteca «RaspiCam» [1]. Una vez descargada se procede a su compilación e instalación:

```
$ tar xvzf raspicamxx.tgz
$ cd raspicamxx
$ mkdir build
$ cd build
$ cmake ..
$ make
# make install
# ldconfig
```

[1] `http://www.uco.es/investiga/grupos/ava/node/40`

F.1.2 Compilación

Una vez instaladas las dependencias, se procede a compilar el sistema. Para ello, simplemente se ejecuta desde el directorio raíz de la aplicación:

```
$ make
```

F.2 Instalación del servidor

La instalación del servidor sigue el mismo proceso de antes, solo que puede ser llevada a cabo en cualquier computador.

F.2.1 Dependencias

El sistema requiere las siguientes dependencias para su funcionamiento:

```
build-essential
cmake
git
libgtk2.0-dev
pkg-config
libavcodec-dev
libavformat-dev
libswscale-dev
python-dev
python-numpy
libtbb2
libtbb-dev
libjpeg-dev
libpng-dev
libtiff-dev
libjasper-dev
libdc1394-22-dev
```

Dichas dependencias pueden ser instaladas mediante el uso del comando `apt-get`:

```
# apt-get install build-essential cmake git libgtk2.0-dev \
    pkg-config libavcodec-dev libavformat-dev libswscale-dev \
    python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev \
    libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev
```

Después, se pasa a compilar la biblioteca OpenCV:

```
$ git clone https://github.com/Itseez/opencv.git
$ cd opencv
$ mkdir release
$ cd release
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
  -D CMAKE_INSTALL_PREFIX=/usr/local ..
$ make
# make install
```

F.2.2 Compilación

Una vez instaladas las dependencias, se procede a compilar el sistema. Para ello, simplemente se ejecuta desde el directorio raíz de la aplicación:

```
$ make
```

F.3 Guía de uso

En este apartado, se introduce el funcionamiento del sistema de tal forma que sirva de guía al usuario.

F.3.1 Instalación de recursos

Los recursos como imágenes, sonidos, vídeos y fuentes, deben de respetar las rutas de directorios existentes:

- Los archivos de imagen deben de ir en el directorio `media/images` del cliente.
- Los archivos de sonido deben de ir en el directorio `media/sounds` del cliente.
- Los archivos de vídeo deben de ir en el directorio `media/videos` del cliente.
- Los archivos de fuentes *true-type* deben de ir en el directorio `media/fonts` del cliente.
- Los archivos de *script* ARS y configuración XML deben de ir en el directorio `data` del servidor.

F.3.2 Calibración del sistema

El proceso de calibrado del sistema se ha creado como una utilidad independiente del sistema principal de ARgos. Una vez calibrado el sistema, la aplicación proporciona los ficheros XML, con los parámetros intrínsecos y extrínsecos, que necesita ARgos para el calculo del registro.

El proceso de calibrado de la cámara esta basado esencialmente por el enfoque de Zhang [Zha99]. Se utiliza un patrón tipo tablero de ajedrez, en la que se alternan cuadrados blancos y negros, de dimensiones conocidas. El patrón se imprime y se pega sobre una superficie plana rígida. Se debe dejar una zona despejada a continuación del tablero de ajedrez, ya que en esta zona se proyectará el patrón que utiliza el proyector para su calibrado.

A continuación, colocamos la superficie bajo la cámara, en distintas orientaciones y distancias, para obtener una serie de imágenes en los que se encuentre visible el patrón.

Cuando el proceso disponga de al menos 20 imágenes válidas, realizará los cálculos para obtener los parámetros intrínsecos de cámara y los grabará en el fichero `calibrationCamera.xml`

El siguiente paso consiste en calibrar el proyector. Se proyecta un patrón de círculos asi-

métrico, en una posición fija. Colocar la superficie de tal forma que el patrón proyectado quede junto al del tablero de ajedrez. Volver a mover el patrón por espacio de proyección hasta que el sistema adquiera 5 imágenes. En ese momento, la calibración pasa al modo dinámico, y los puntos proyectados se generan en función de la posición que tenga tablero de ajedrez.

Tras capturar 15 imágenes con los puntos dinámicos, se realizan los cálculos de los parámetros intrínsecos del proyector, y finalmente, de los parámetros extrínsecos (matrices de rotación y traslación) entre la cámara y el proyector.

Los parámetros de calibrado se almacenan en los ficheros `calibrationProjector.xml` y `cameraProjectorExtrinsics.xml`.

Mientras que la cámara y el proyector mantengan su posición y rotación entre ellos, no es necesario realizar una nueva calibración y es posible mover todo el sistema.

Una vez realizada la calibración se iniciará un test para comprobar que se ha hecho correctamente. El test consiste en proyectar un círculo sobre las cuatro esquinas exteriores del patrón de tablero de ajedrez. Al mover el patrón los puntos proyectados deben siempre permanecer alineados con las esquinas.

F.3.3 Descripción de documentos

La descripción de los documentos que se puedan identificar en el sistema se realiza proporcionando una imagen en formato JPEG de una plantilla del documento.

F.3.4 Fichero de configuración

El archivo de configuración permite cambiar algunos parámetros del sistema. El archivo se encuentra en la carpeta `data` del servidor y presenta una estructura tal que:

```
<config>
  <paper_size width="21.0" height="29.7" />
  <calibration_files camera="calibrationCamera.xml"
    projector="calibrationProjector.xml"
    extrinsics="CameraProjectorExtrinsics.xml" />
  <argos_papers>
    <paper id="0" description="Neobiz"
      scriptFileName="NeobizScript.ars"
      descriptorFileName="neobiz-.jpg" />
    <paper id="1" description="Sinovo"
      scriptFileName="SinovoScript.ars"
      descriptorFileName="sinovo-.jpg" />
    <paper id="2" description="Active"
      scriptFileName="ActiveScript.ars"
      descriptorFileName="active-.jpg" />
  </argos_papers>
  <output_display type="projector" />
</config>
```

El elemento `paper_size` indica las dimensiones de los documentos sobre los que se va a trabajar. En este caso, representa el tamaño de un papel en formato A4, es decir, 21.0x29.7 cm.

El elemento `calibration_files` indica los nombres de los archivos de calibración generados en el apartado anterior.

El elemento `argos_papers` define la lista de documentos con los que va a trabajar el sistema. Estos documentos mantienen una asociación con un archivo de *script* ARS y su imagen correspondiente.

El elemento `output_display` indica al sistema si la visualización la va a realizar un proyector o un monitor para realizar los cálculos del registro en función del dispositivo.

F.3.5 Creación de *scripts*

Para la creación de *scripts* se debe crear un nuevo documento de texto plano con cualquier editor de texto soportado y escribir en él las sentencias que deseemos.

Sintaxis

Las sentencias siguen el formato:

```
<carácter_de_control> <función> <arg_1> <arg_2> <arg_n> ...
```

Donde cada elemento de la sentencia significa:

- **<caracter de control>:** Puede ser uno de los siguientes:
  - # Indica que esta sentencia será ignorada (no se ejecutará).
  - > La sentencia precedida por este carácter será ejecutada.
  - ! Una vez encontrado este carácter, el resto del *script* será ignorado (no se ejecutará).
- **<función>:** Se trata del nombre de la función a invocar.
- **<arg_1><arg_2><arg_n>...:** Los argumentos de la función anterior.

Funciones

A continuación se especifica la lista de funciones disponibles a invocar desde un *script*.

DrawImage

DESCRIPCIÓN: Dibuja una imagen desde el disco.

ARGUMENTOS:

- filename: *string*. El nombre del archivo de imagen a dibujar.
- x: *float*. La coordenada X donde situar la imagen.
- y: *float*. La coordenada Y donde situar la imagen.

- z: *float*. La coordenada Z donde situar la imagen.
- w: *float*. El ancho de la imagen (1.0 = 1:1).
- h: *float*. El alto de la imagen (1.0 = 1:1).

DrawVideo

DESCRIPCIÓN: Dibuja un vídeo desde el disco.

ARGUMENTOS:

- filename: *string*. El nombre del archivo de video a dibujar.
- x: *float*. La coordenada X donde situar la video.
- y: *float*. La coordenada Y donde situar la video.
- z: *float*. La coordenada Z donde situar la video.
- w: *float*. El ancho de la imagen (1.0 = 1:1).
- h: *float*. El alto de la imagen (1.0 = 1:1).

DrawCorners

DESCRIPCIÓN: Dibuja esquinas sobre el documento.

ARGUMENTOS:

- length: *float*. La longitud de segmento de cada esquina.
- wide: *float*. El ancho de segmento de cada esquina.
- r: *float*. La componente de color roja para las esquinas, entre 0.0 y 1.0.
- g: *float*. La componente de color verde para las esquinas, entre 0.0 y 1.0.
- b: *float*. La componente de color azul para las esquinas, entre 0.0 y 1.0.
- w: *float*. El ancho del documento. Un A4 sería 21.
- h: *float*. El alto del documento. Un A4 sería 29.7.

DrawAxis

DESCRIPCIÓN: Dibuja un eje de coordenadas.

ARGUMENTOS:

- length: *float*. La longitud de segmento de cada eje.
- wide: *float*. El ancho de segmento de cada eje.
- x: *float*. La coordenada X donde situar el eje de coordenadas.
- y: *float*. La coordenada Y donde situar el eje de coordenadas.
- z: *float*. La coordenada Z donde situar el eje de coordenadas.

InitVideostream

DESCRIPCIÓN: Inicia una videollamada.

ARGUMENTOS:

- bg_file: *string*. El nombre del archivo de imagen a cargar de fondo para la videollamada.

- w: *float*. El ancho de la videollamada.
- h: *float*. El alto de la videollamada.
- port: *int*. El número de puerto donde iniciar la videollamada.

DrawTextPanel

DESCRIPCIÓN: Dibuja un panel de texto.

ARGUMENTOS:

- r: *float*. La componente de color roja de este `TextPanel`, entre 0.0 y 1.0.
- g: *float*. La componente de color verde de este `TextPanel`, entre 0.0 y 1.0.
- b: *float*. La componente de color azul de este `TextPanel`, entre 0.0 y 1.0.
- text: *string*. El texto que contendrá este `TextPanel`.
- x: *float*. La coordenada X donde situar el `TextPanel`.
- y: *float*. La coordenada Y donde situar el `TextPanel`.
- z: *float*. La coordenada Z donde situar el `TextPanel`.

DrawHighlight

DESCRIPCIÓN: Dibuja un área de resaltado.

ARGUMENTOS:

- r: *float*. La componente de color roja para este `Highlight`, entre 0.0 y 1.0.
- g: *float*. La componente de color verde para este `Highlight`, entre 0.0 y 1.0.
- b: *float*. La componente de color azul para este `Highlight`, entre 0.0 y 1.0.
- x: *float*. La coordenada X donde situar este `Highlight`.
- y: *float*. La coordenada Y donde situar este `Highlight`.
- z: *float*. La coordenada Z donde situar este `Highlight`.
- w: *float*. El ancho de este `Highlight` (1.0 = 1:1).
- h: *float*. El alto de este `Highlight` (1.0 = 1:1).

DrawButton

DESCRIPCIÓN: Dibuja un botón.

ARGUMENTOS:

- r: *float*. La componente de color roja para este botón, entre 0.0 y 1.0.
- g: *float*. La componente de color verde para este botón, entre 0.0 y 1.0.
- b: *float*. La componente de color azul para este botón, entre 0.0 y 1.0.
- text: *string*. El texto de este botón.
- x: *float*. La coordenada X donde situar este botón.
- y: *float*. La coordenada Y donde situar este botón.
- z: *float*. La coordenada Z donde situar este botón.

DrawFactureHint

DESCRIPCIÓN: Dibuja un cuadro con instrucciones para tratar el documento.
ARGUMENTOS:

- x: *float*. La coordenada X donde situar esta ayuda.
- y: *float*. La coordenada Y donde situar esta ayuda.
- z: *float*. La coordenada Z donde situar esta ayuda.
- w: *float*. El ancho de esta ayuda (1.0 = 1:1).
- h: *float*. El alto de esta ayuda (1.0 = 1:1).
- r: *float*. La componente de color roja para esta ayuda, entre 0.0 y 1.0.
- g: *float*. La componente de color verde para esta ayuda, entre 0.0 y 1.0.
- b: *float*. La componente de color azul para esta ayuda, entre 0.0 y 1.0.
- title: *string*. El texto de título de esta ayuda.
- text_block_1: *string*. El primer bloque de texto de esta ayuda.
- text_block_2: *string*. El segundo bloque de texto de esta ayuda.

PlaySound

DESCRIPCIÓN: Reproduce un sonido.
ARGUMENTOS:

- filename: *string*. El nombre del archivo de sonido a reproducir.
- loops: *int*. El número de veces a reproducir el sonido. -1 indefinidamente, 0 reproduce una vez, 1 reproduce dos veces, etc.

PlaySoundDelayed

DESCRIPCIÓN: Reproduce un sonido en bucle con un retraso de tiempo entre reproducciones indicado.
ARGUMENTOS:

- filename: *string*. El nombre del archivo de sonido a reproducir.
- delay: *float*. El número de segundos a esperar entre reproducciones.

CheckRegion

DESCRIPCIÓN: Comprueba el documento dentro de la región rectangular indicada.
ARGUMENTOS:

- x: *float*. La posición X de la región a comprobar (esquina superior izquierda).
- y: *float*. La posición Y de la región a comprobar (esquina superior izquierda).
- w: *float*. El ancho de la región a comprobar (esquina inferior derecha).
- h: *float*. El alto Y de la región a comprobar (esquina inferior derecha).

F.3.6 Ejecución

El arranque del sistema requiere ejecutar de forma separada tanto el servidor como el cliente, y en ese orden para obtener la dirección de conexión.

Ejecución del servidor. Se debe indicar el número de puerto y la interfaz de red donde escuchará el servidor:

```
$ ./argos_server <port> <iface>}
```

Ejecución del cliente. Se debe indicar la dirección IP y el puerto proporcionados por el servidor:

```
$ ./argos_client <ip>:<port>
```

Una vez ejecutado solo queda interactuar en la superficie de trabajo con los documentos definidos en los descriptores.

Anexo G

Conclusión personal

El desarrollo de BelfegAR ha supuesto el hito final de mis 4 años de estudio en la carrera. A lo largo de este tiempo, he adquirido muchos conocimientos que me han servido para aplicarlos en el momento de este desarrollo. Sin embargo, también he tenido que enfrentarme a nuevos retos al tener que utilizar tecnologías que no conocía y de rendimiento reducido, como ha sido la plataforma de despliegue. Esto me ha hecho ver la importancia aun a día de hoy de escribir programas eficientes que puedan llegar a funcionar hasta en el entorno más hostil, siempre dentro de lo posible.

El abordar un proyecto de tal magnitud me ha ayudado a realizar una planificación previa de éste, aplicando técnicas de ingeniería del software aprendidas en la carrera. Me he percatado pues, de lo importante que es realizar un buen diseño antes de llevar a cabo la implementación, para conseguir una arquitectura de software lo más robusta y escalable posible.

Gracias a este proyecto, he aprendido además muchas técnicas del propio lenguaje C++ que muy probablemente me resultarán útiles a lo largo de mi carrera profesional. También he aprendido a programar aplicaciones que hagan uso de la implementación moderna de OpenGL pese a que al principio surgieron muchas dificultades.

Finalmente, me gustaría dejar escritas una palabras sobre la carrera de Ingeniería Informática para posibles futuros lectores. La carrera puede ser a veces complicada o desesperante, pero con esfuerzo y constancia resulta asequible. La propia naturaleza de la informática permite ver resultados muy rápidamente a costes muy bajos o casi nulos, resultando en una de las carreras más gratificantes y versátiles que existen. Sin embargo requiere constancia y continuar estudiando prácticamente toda la vida, porque la informática no es algo que se termine una vez acabados los estudios; la informática es algo vivo y en constante evolución, que gana mayor importancia con el paso de los años.

Anexo H

GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format,

SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version. 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

5. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

6. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

7. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

8. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

9. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

10. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

Referencias

[AFAN13] R. Anacleto, L. Figueiredo, A. Almeida, y P. Novais. *Server to Mobile Device Communication: A Case Study*, volume 219 of *Advances in Intelligent Systems and Computing*. 2013. Cited By (since 1996):1.

[Bee13] P. Beech. Raspberry Pi Software Architecture. `http://elinux.org/Raspberry_Pi_VideoCore_APIs`, 2013.

[BK13] G. Bradski y A. Kaehler. *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, Inc., edición 2nd, 2013.

[BN84] A.D. Birrell y B.J. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2:39–59, 1984.

[dAPGdE08] Ministerio de Administraciones Públicas. Gobierno de España. Presentación sobre la tecnología y arquitectura de la red SARA. `http://www.jcyl.es/web/jcyl/pr/ds/MunicipiosDigitales/pdf;charset=UTF-8/144/334/INTEROPERABILIDAD.pdf/_?asm=jcyl`, Octubre 2008.

[ECCDPM12] R. Esteller-Curto, E. Cervera, A.P. Del Pobil, y R. Marin. Proposal of a REST-based architecture server to control a robot. En *Proceedings - 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2012*, páginas 708–710, 2012.

[EMVH05] A. Engler, D. Müller, S. Vrancken, y M. Hecklein. *Geometria Analitica*. Universidad Nacional del Litoral, 2005.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, y T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. *RFC 2616*, 1999. url: `http://tools.ietf.org/search/rfc2616`.

[Fra13] Fragment Shader. `http://www.opengl.org/wiki/Fragment_Shader`, 2013.

[FT02] R.T. Fielding y R.N. Taylor. Principled Design of the Modern Web Architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, Mayo 2002.

[Gre08] P. Green. Iterative Design. *Industrial and Operations Engineering 436*, February 2008.

[Kni07] H. Kniberg. *Scrum and XP from the Trenches: Enterprise Software Development*. 2007.

[LG10] M. Lanthaler y C. Gütl. Towards a RESTful service ecosystem: Perspectives and challenges. En *4th IEEE International Conference on Digital Ecosystems and Technologies - Conference Proceedings of IEEE-DEST 2010, DEST 2010*, páginas 209–214, 2010. Cited By (since 1996):3.

[Lut07] M. Lutz. *Learning Python*. O'Reilly Media, edición 3rd, 2007.

[Mar01] B. Marchal. Soapbox: Why I'm using SOAP. 2001. url: `http://www.ibm.com/developerworks/xml/library/x-soapbx1/index.html`.

[MGS08] A. Munshi, D. Ginsburg, y D. Shreiner. *OpenGL(R) ES 2.0 Programming Guide*. Addison-Wesley Professional, edición 1, 2008.

[Mye98] B.A. Myers. A Brief History of Human-computer Interaction Technology. *interactions*, 5(2):44–54, Marzo 1998.

[NSOJA13] J.M. Noguera, R.J. Segura, C.J. Ogáyar, y R. Joan-Arinyo. A scalable architecture for 3D map navigation on mobile devices. *Personal and Ubiquitous Computing*, 17(7):1487–1502, 2013.

[NVAV07] C. Newham, J. Vossen, C. Albing, y J. Vossen. *Bash Cookbook: Solutions and Examples for Bash Users (Cookbooks (O'Reilly))*. O'Reilly Media, 2007.

[Obj95] OBJ Specification. `http://www.martinreddy.net/gfx/3d/OBJ.spec`, 1995.

[OO02] M. Olson y U. Ogbuji. The Python Web services developer: Messaging technologies compared. 2002. url: `www.ibm.com/developerworks/webservices/library/ws-pyth9/?dwzone=webservices`.

[Pad82] M.A. Padlipsky. A Perspective on the ARPANET Reference Model. *RFC 871*, 1982. url: `http://tools.ietf.org/html/rfc871`.

[PPAB11] M. Pirnau, C. Pirnau, M. Apetrei, y G. Badea. The SOAP protocol used for building and testing Web services. En *Proceedings of the World Congress on Engineering 2011, WCE 2011*, volume 1, páginas 475–480, 2011.

[SDLa] Anuncio oficial de SDL2. http://forums.libsdl.org/viewtopic.php?p=38568&sid=06a15a765f13bf8512ec05cb7115ee39. Consultado el 19-08-2014.

[SDLb] Página oficial de la biblioteca SDL. https://www.libsdl.org/. Consultado el 19-08-2014.

[SS13] K. Schwaber y J. Sutherland. *The Scrum Guide*. 2013.

[Str13] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, edición 4th, 2013.

[Ver13] Vertex Shader. http://www.opengl.org/wiki/Vertex_Shader, 2013.

[Wik] Wikipedia. Raspberry Pi Specifications. http://en.wikipedia.org/wiki/Raspberry_Pi. Consultado el 20-08-2014.

[Wik13] 2013. Consultado el 20-08-2014. url: http://es.wikibooks.org/w/index.php?title=Matem%C3%A1ticas/Matrices/Concepto_de_Matriz&oldid=209964.

[Win71] J.M. Winett. The Definition of a Socket. *RFC 147*, 1971. url: http://tools.ietf.org/html/rfc147.

[XJY09] F. Xinyang, S. Jianjing, y F. Ying. REST: An alternative to RPC for web services architecture. En *2009 1st International Conference on Future Information Networks, ICFIN 2009*, páginas 7–10, 2009.

[XZLS08] X. Xu, L. Zhu, Y. Liu, y M. Staples. Resource-oriented architecture for business processes. En *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, páginas 395–402, 2008. Cited By (since 1996):14.

[Zha99] Z. Zhang. Flexible camera calibration by viewing a plane from unknown orientations. En *Proceedings of the Seventh IEEE International Conference on Computer Vision*, páginas 666–673, 1999.

Este documento fue editado y tipografiado con LaTeX
empleando la clase **esi-tfg** que se puede encontrar en:
`https://bitbucket.org/arco_group/esi-tfg`