WILI: SISTEMA DE ANÁLISIS DE ILUMINACIÓN ADAPTATIVO PARA APLICACIONES DE REALIDAD AUMENTADA

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA

EN INFORMÁTICA

PROYECTO FIN DE CARRERA

WILi: Sistema de Análisis de Iluminación Adaptativo para Aplicaciones de Realidad Aumentada

Jorge Barco Moreno

Septiembre, 2011

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Tecnologías y Sistemas de Información

PROYECTO FIN DE CARRERA

WILi: Sistema de Análisis de Iluminación Adaptativo para Aplicaciones de Realidad Aumentada

Autor: Jorge Barco Moreno

Director: Dr. Carlos González Morcillo

Septiembre, 2011

Jorge Barco Moreno

Ciudad Real – Spain

E-mail: JorgeBarco@alu.uclm.es, jorgebarcomoreno@gmail.com
Web site: `wiliproject.quijost.com`

TRIBUNAL:

Presidente:

Vocal 1:

Vocal 2:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL 1	VOCAL 2	SECRETARIO
Fdo.:	Fdo.:	Fdo.:	Fdo.:

Resumen

La Realidad Aumentada puede definirse como un novedoso paradigma de interacción que consiste en la integración de imágenes sintéticas 3D generadas por ordenador con la imagen captada del mundo físico en tiempo real.

Este campo de la informática gráfica está experimentado un importante crecimiento en los últimos años, llegando a tener una influencia representativa en el mercado y comenzando a estar presente en la vida cotidiana.

Para maximizar la correcta integración de los objetos virtuales en el mundo real es necesario emplear métodos de síntesis de imagen realista que simulen adecuadamente el modelo de interacción física de la luz. El principal problema asociado con estos algoritmos de *rendering* realista es su elevado coste computacional.

El presente Proyecto de Fin de Carrera llamado *WILi* (*Where Is Light?*) surge con el objetivo principal de aplicar métodos de síntesis de imagen realista y cálculo de iluminación en aplicaciones de Realidad Aumentada. A diferencia de otras aproximaciones, *WILi* emplea métodos de precálculo de texturas hiperrealistas para obtener en cada instante la configuración que mejor se ajusta a las condiciones de iluminación del mundo físico. Para ello, obtiene las características básicas de los focos de iluminación principales del mundo real para integrar posteriormente los modelos virtuales de un modo más realista. Con el fin de superar los requisitos de interactividad de las aplicaciones de Realidad Aumentada, las etapas de análisis, de iluminación y de despliegue del modelo se realizan en tiempo real.

La arquitectura modular de *WILi* permite emplear diferentes algoritmos y métodos de síntesis de imagen realista para su adaptación a diversos proyectos de Realidad Aumentada. El módulo de análisis de iluminación podrá ser mejorado (sin cambios en el resto de componentes del sistema) y soportar un mayor número de propiedades de los focos de iluminación (color, intensidad, borrosidad, etc.). Para finalizar, la capa de representación permite desplegar modelos tridimensionales con mapeado *UV* de diferente nivel de detalle.

Abstract

Augmented Reality can be defined as a new integration paradigm where 3D sintetic images created by computer are integrated with the physic world image catch in real time.

This computer graphics field is undergoing an important growth recently, reaching a big influence in markets and starting to take part of our daily lifes.

To maximize the successful integration of virtual objects in the real world is necessary to use methods for realistic image synthesis that simulates adequately the physical light interaction model. The main problem associated with these realistic rendering algorithms is the high computational cost.

This End of Degree called *WILI* (*Where Is Light?*) uses hyper-realistic textures precalculus methods to get the best setup according to physical world light. In order to do this, it gets the the basic features of the main light sources in real-world to integrate the virtual models in a more realistic way. Besides, the analysis stages, lighting and deployment model are done in real time.

The *WILi's* modular architecture allows using different algorithms and methods for realistic image synthesis to be used in various Augmented Reality projects. The analysis module would be improved (without changes in other system components) and will support a greater number of light properties (color, intensity, blur, etc.). Finally, the representation layer allows deploying 3D models with UV mapping using different detail levels.

Índice general

Resumen IX

Abstract XI

Índice general XIII

Índice de cuadros XVII

Índice de figuras XIX

Índice de listados XXI

1. Introducción 1

1.1. Realidad aumentada 1

1.2. Impacto socio-económico 6

1.3. Realismo en aplicaciones de Realidad Aumentada 7

1.4. Problemática 8

1.5. Estructura del documento 9

2. Antecedentes 11

2.1. Fundamentos matemáticos 11

2.1.1. Teoría vectorial 12

2.1.2. Teoría matricial 17

2.2. Síntesis de imagen realista 21

2.2.1. Modelos de iluminación 21

2.2.2. Proyección de texturas 23

2.2.3. Modos de proyección de sombras 24
2.2.4. Bibliotecas para síntesis de imagen interactiva 27
2.3. Visión por computador 31
2.3.1. Definición de contornos 31
2.3.2. Gestión de histogramas 34
2.4. Suites de gráficos 3D 37
2.4.1. Características 37
2.4.2. 3ds Max 39
2.4.3. LightWave 39
2.4.4. Maya 40
2.4.5. Rhino 3D 40
2.4.6. Blender 41
2.4.7. Comparativa 42
2.4.8. Conclusión 44

3. Objetivos **45**

4. Método de trabajo **47**
4.1. Metodología de trabajo 47
4.2. Herramientas 48
4.2.1. Lenguajes de programación 49
4.2.2. Hardware 49
4.2.3. Software 49

5. Arquitectura **53**
5.1. Descripción general 53
5.2. Subsistema de precálculo de iluminación 55
5.2.1. Módulo Baker 55
5.2.2. Módulo de entrada/salida 63
5.3. Subsistema de análisis 67
5.3.1. Módulo de tratamiento gráfico 67

5.3.2. Módulo de visión por computador 71
5.3.3. Módulo de gestión de texturas 76
5.3.4. Módulo controlador 78
5.3.5. Módulo de depuración 79
5.4. Subsistema de representación 84
5.4.1. Módulo de entidades 2D 84
5.4.2. Módulo de entidades 3D 85

6. Evolución, Resultados y Costes **89**
6.1. Evolución del proyecto 89
6.1.1. Infraestructura para la metodología 89
6.1.2. Concepto del software 90
6.1.3. Análisis preliminar de requisitos 90
6.1.4. Diseño general 91
6.1.5. Estadísticas del repositorio 97
6.2. Resultados 97
6.3. Recursos y costes 99
6.3.1. Coste económico 99

7. Conclusiones y propuestas **101**
7.1. Objetivos alcanzados 101
7.2. Propuestas de trabajo futuro 103
7.3. Conclusión personal 105

A. Construcción de marcas **109**
A.1. Las marcas 109
A.2. Los materiales 109
A.3. Disposición del cubo en la marca 110

B. Constantes **113**
B.1. Subsistema de precálculo de iluminación 113
B.1.1. Constantes de los focos de luz 113

B.1.2. Constantes de rutas 114
B.1.3. Constantes de nombres de objetos 114
B.1.4. Constantes de la esfera 114
B.1.5. Otras constantes 115
B.1.6. Constantes del fichero Baker.py 116
B.2. Subsistema de análisis 116
B.2.1. Constantes del cubo 116
B.2.2. Constantes del área de proyección de la sombra 116
B.2.3. Otras constantes 116
B.2.4. Fichero WiliConstants.h 120

C. Código fuente **121**

D. GNU GENERAL PUBLIC LICENSE **123**

Bibliografía **133**

Índice de cuadros

2.1. Comparativa entre *OpenGL* y *Direct3D*. 30
2.2. Comparativa de soporte entre SSOO y Licencias entre suites de gráficos 3D. 42
2.3. Comparativa de características soportadas entre suites de gráficos 3D. . . . 43

5.1. Mejoras y ampliaciones realizadas al importador del formato Orej 64

6.1. Estadísticas del repositorio. 98
6.2. Tiempo de cómputo de acciones dependientes de la geometría. 98
6.3. Tiempo de cómputo de acciones independientes de la geometría. 99
6.4. Desglose económico del coste de *WILi*. 100

B.1. Constantes de los focos de luz 113
B.2. Constantes de rutas 114
B.3. Constantes de nombres de objeto 114
B.4. Constantes de la esfera 115
B.5. Otras constantes del subsistema de precálculo de iluminación 115
B.6. Constantes del cubo 117
B.7. Constantes del área de proyección de la sombra 118
B.8. Otras constantes del subsistema de análisis 119

Índice de figuras

1.1. Aplicaciones basadas en Realidad Aumentada 2
1.2. Comparativa de búsquedas entre *Augmented Reality* y *Virtual Reality* 6

2.1. Mapa conceptual de *WILi*. 12
2.2. Proyección de un vector $\vec{u}$ sobre $\vec{v}$. 14
2.3. Distancia mínima entre dos rectas en $\Re^3$. 15
2.4. Ejemplo de aplicación de un mapa UV 24
2.5. Posición de un objeto en función de su sombra 25
2.6. Proceso de generación de sombras y componentes de la misma 26
2.7. Comparación de código al dibujar un triángulo con *OpenGL* y *Direct3D* . . 29
2.8. Binarización de una imagen 32
2.9. Algoritmo de Douglas-Peucker 34
2.10. Operaciones aritméticas con histogramas 35

4.1. Metodología basada en prototipado evolutivo. 48

5.1. Esquema modular de *WILi*. 54
5.2. Consecuencias en la elección del tamaño del plano 58
5.3. Proceso de precálculo de texturas 58
5.4. Texturas generadas por el módulo *Baker* 59
5.5. Filtros aplicados a la imagen 70
5.6. Elección de vértices según la orientación de la sombra 74
5.7. Búsqueda de vértices de la sombra 75
5.8. . Cálculo del foco a partir de los vértices detectados. 76

5.9. Algoritmo de búsqueda de la mejor textura 77

5.10. Diagrama de flujo del subsistema de análisis. 82

5.11. Imprecisión en la correspondencia 3D/2D 83

5.12. Captura de las ventanas de depuración. 83

6.1. Gráfica de *commits* realizados y evolución de *WILi* 97

6.2. Ejemplo de salida de WILi con una escena real 100

A.1. Medidas para la construcción de la marca 111

B.1. Constantes que representan elementos de la marca 119

Índice de listados

5.1. Clase Object 66
5.2. Obtención del contorno de la sombra poligonizado 73

Agradecimientos

Quisiera en este espacio agradecer a todas las personas que de una manera u otra han contribuido a que pueda presentar mi Proyecto de Fin de Carrera.

En primer lugar, querría agradecer a mis padres y a mi hermana todo el esfuerzo ejemplar que han realizado año tras año para que pueda ser hoy lo que siempre quise ser. Gracias por todos los buenos consejos, por los ánimos y por ayudarme en mis momentos de indecisión. También agradecer a mis familiares que siempre han confiado en mi.

Gracias a Silvia por su paciencia, por la seguridad y fuerza que me aporta día tras día y por su apoyo incondicional. Gracias por hacerme ver las cosas de una manera más sencilla.

Gracias a toda la gente de informática que he conocido y con la que he pasado muy buenos momentos, en especial a César, Paco y Tafa.

Gracias a mis compañeros del laboratorio de investigación Oreto, siempre dispuestos a ayudarme cuando lo he necesitado.

Por último, querría destacar en este espacio a Carlos González Morcillo por su atención desinteresada y la profesionalidad que ha manifestado durante la dirección del presente proyecto. Gracias por todo el trabajo y esfuerzo realizado.

CAPÍTULO 1

INTRODUCCIÓN

LA *Realidad Aumentada* consiste en la integración de elementos virtuales sobre la imagen obtenida del mundo real. Este paradigma de interacción está creciendo de forma significativa, aumentando el número de aplicaciones que emplean este tipo de técnicas.

En este capítulo se introducen las características que deben cumplir las aplicaciones para ser consideradas de Realidad Aumentada, así como una clasificación de las mismas según la temática en las que son aplicadas. Por último se describirá la problemática relacionada con la integración realista de la imagen virtual, objetivo principal del presente Proyecto de Fin de Carrera.

1.1. Realidad aumentada

La *Realidad Aumentada* puede definirse como un paradigma de interacción que compone información sintética generada por computador con imágenes obtenidas del mundo real. Según R.Azuma, para que una aplicación sea considerada de Realidad Aumentada, han de cumplirse las siguientes tres características [A+97]:

- *Tiene que combinar objetos reales y virtuales en un entorno real*. Esto significa que la imagen resultante debe ser el resultado de una captura de la realidad a la que se le superpone, al menos, un objeto virtual.

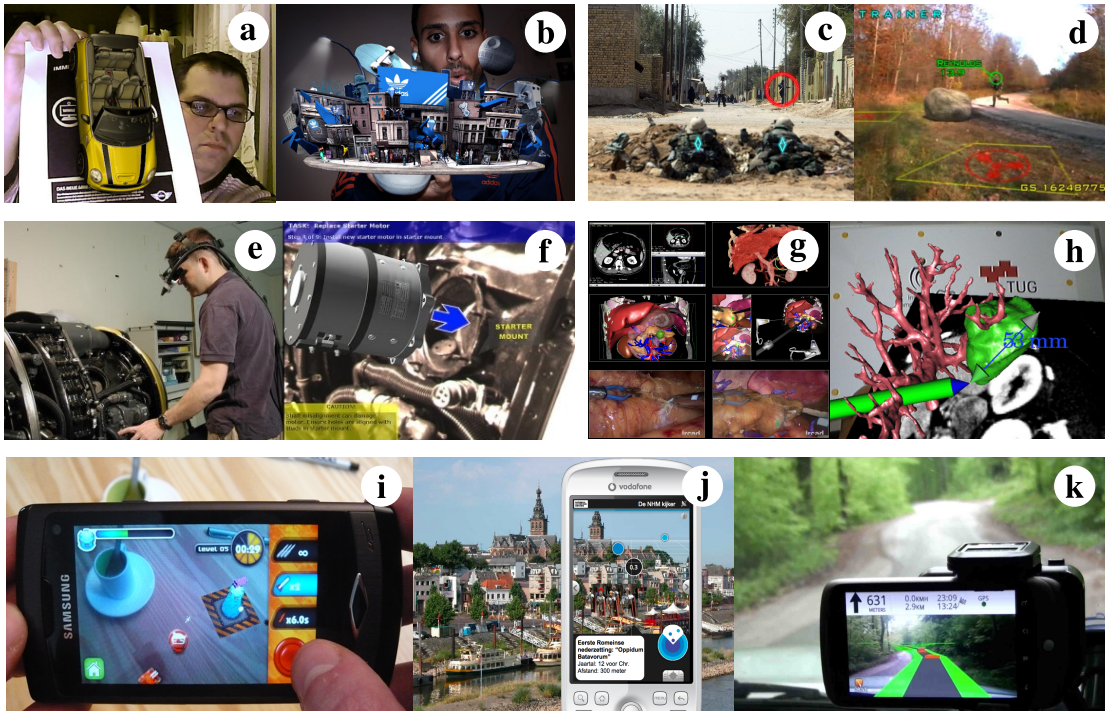

Figura 1.1: Aplicaciones basadas en Realidad Aumentada. a) Utilización en publicidad web de Mini. b) Edición especial de zapatillas Adidas AR. c) y d) Uso de la RA en aplicaciones militares. e) y f) Aplicación de la RA en el mantenimiento de maquinaria. g) y h) RA aplicada en medicina. i) Aplicación *AR Defender*. j) *Layar*. k) *Wikitude Drive*.

- *La combinación ha de realizarse en tiempo real*. Este factor es determinante y el causante de que cualquier técnica de postprocesado de imágenes no se considere Realidad Aumentada.

- *Debe alinear y componer objetos virtuales con la realidad en 3D*. Aunque el resultado sea una imagen bidimensional, la alineación y composición de los elementos virtuales deben hacerse en base a un mundo tridimensional.

En un primer momento, Azuma clasificaba las aplicaciones de Realidad Aumentada en aplicaciones médicas, aplicaciones para soporte en la fabricación y reparación, aplicaciones militares y aplicaciones de entretenimiento [A+97].Sin embargo, el avance de la informática que rápidamente incorpora nuevas disciplinas, además de contar con el soporte de mejor hardware , hace que en tan solo cuatro años se tenga que redefinir esta clasificación en [ABB+01] incluyendo nuevos conjuntos más generales: *aplicaciones móviles*, *aplicaciones colaborativas* y *aplicaciones comerciales*. En la figura 1.1 puede verse una figura compuesta por imágenes de distintas aplicaciones basadas en Realidad Aumentada. A continuación se describe cada una de ellas:

- *Aplicaciones médicas*.

  La *Realidad Aumentada* va ganando terreno en diversos campos de la medicina. Los sistemas de apoyo en las operaciones de cirugía van poco a poco extendiéndose, apoyándose sobretodo en la ventaja de tener en tiempo real modelos virtuales superpuestos adaptados al paciente que ayudan al cirujano a tomar decisiones, hacer comparaciones o simplemente actuar con mayor precisión [Gei05].

  También se emplea la Realidad Aumentada para ver algún órgano de un paciente en su contexto sin aplicar ninguna incisión, reconstruyendo a partir de sensores como ultrasonidos o resonancias magnéticas modelos virtuales que los representan [RBBS06].

- *Aplicaciones para soporte en la fabricación y reparación de maquinaria compleja*.

  La ventaja de estas aplicaciones, frente a un manual de texto con imágenes, es que en tiempo real el software va indicando al técnico cada paso que tiene que realizar y cómo lo debe hacer. Para ello, sobre la imagen real se superponen animaciones de elementos virtuales o incluso algún vídeo informativo [SD03].

  Un ejemplo de este tipo de aplicaciones es el proyecto ARVIKA, que emplea la Realidad Aumentada para el desarrollo, producción y servicio en las industrias aeroespacial y de automoción [FJS01].

- *Aplicaciones militares*.

  La Realidad Aumentada se emplea ampliamente en el ámbito militar. En aviación, se superponen datos en 3D en el HMD (*Head Mounted Display*) del piloto para visualizar información ampliada sobre objetivos militares.

  En despliegues terrestres de tropas, la Realidad Aumentada ofrece al soldado información relevante en un entorno desconocido para él, indicándole recorridos a realizar, información de los edificios cercanos e incluso la visualización de información que está fuera de su campo de visión. Un ejemplo de la aplicación de la Realidad Aumentada es el proyecto BARS del Laboratorio de Investigación Naval de Estados Unidos [LRJ+02].

- *Aplicaciones de entretenimiento*.

  Cada vez son más las aplicaciones de Realidad Aumentada aplicadas al entretenimiento. Algunas aplicaciones presentan unos requisitos hardware muy específicos y costosos, pero según [TS10], las aplicaciones de este tipo presentan una tendencia hacia sistemas más baratos como los basados en marcas y cámaras.

  A continuación se describen algunas de estas aplicaciones:

  - *ARQuake*: Permite jugar al famoso juego Quake cambiando el entorno virtual por uno alineado con la realidad. ARQuake fué desarrollado en el año 2000 por

la Facultad de Ciencias de la Computación e Información de la Universidad del Sur de Australia y supuso uno de los primeros juegos basados en la Realidad Aumentada para jugar al aire libre [TCD+00].

Para poder jugar a ARQuake, es necesario contrar con material hardware específico costoso: Para reconocer el movimiento se emplean sensores inerciales, una brújula y GPS. Para poder visualizar los objetos de la realidad con los virtuales superpuestos se emplea un HMD[PT03].

- *Invizimals*: Juego creado por la empresa Novarama y comercializada por Sony para su dispositivo PSP en el que el usuario captura animales virtuales de especies inexistentes y los usan para combatir contra otros usuarios. El juego tuvo una gran aceptación entre los usuarios con un volumen de ventas en España de más de 70.000 unidades.

- *AR Defender*: Se trata de un juego comercial lanzado por Apple para su dispositivo iPhone. Emplea una marca como método de tracking para situar una torre virtual. El objetivo del juego es defender la torre moviendo la cámara y disparando con varias armas a los enemigos que intentan destruirla.

- *Aplicaciones móviles*.

  Uno de los mercados emergentes que mayor crecimiento presenta es el de la Realidad Aumentada en móviles. La gran mayoría de los terminales móviles llevan incorporada una cámara y algunos de ellos tienen giróscopos, acelerómetros, GPS e incluso brújulas digitales, lo que los convierten en dispositivos perfectos para la ejecución de aplicaciones de realidad aumentada [CP11]. Usando estos sensores como método de tracking se evita, en algunos casos, el empleo de técnicas basadas en el reconocimiento de marcas para obtener una posición en el mundo real.

  Según [Joh10] hoy en día hay un gran número de aplicaciones de Realidad Aumentada para móviles. A continuación se describen algunas de ellas:

  - *Pocket Universe: Virtual Sky Astronomy*. Esta aplicación de Craic Design utiliza la Realidad Aumentada para mostrar, entre otras cosas, información sobre los cuerpos celestes que obtiene a través de la cámara. De esta forma muestra el cielo con información sobre las estrellas que detecta. Esta aplicación es compatible con iPhone, iPod Touch y iPad.

  - *Nearest Tube*. Desarrollada por AcrossAir para iPhone3G, esta aplicación muestra sobre la imagen capturada por la cámara flechas que indican la dirección de las bocas de metro más cercanas diferenciándolas por líneas y las paradas de autobús.

- *Layar*. Según [Kar11], Layar es un navegador lanzado en el 2009 basado en Realidad Aumentada[1] que añade información adicional siguiendo un sistema de capas. Estas capas varían desde información ofrecida mediante búsquedas en Google hasta información encontrada en Wikipedia sobre la posición del terminal e información de las tiendas cercanas.
- *Wikitude*. Se trata del primer navegador basado en Realidad Aumentada creado por la compañia Mobilizy. Su lanzamiento tuvo lugar cuando Google introdujo por primera vez Android en un dispositivo móvil en el 2008 . Este navegador escanea el entorno empleando la cámara y los sensores del terminal móvil como el GPS , el acelerómetro o la brújula para mostrar contenido geo-referenciado en el mismo lugar que los objetos de la realidad. Wikitude superpone a la imagen captada por la cámara información sobre determinados puntos de interés de acuerdo a las preferencias y necesidades del usuario.

- *Aplicaciones colaborativas*.

  En las aplicaciones colaborativas basadas en Realidad Aumentada destacan aquellas de soporte para reuniones. Con estos sistemas se pueden emplear objetos virtuales que son manejados directamente por los usuarios [BHF+99].

  Otra aplicación de la Realidad Aumentada en este campo es el soporte a la colaboración remota, superponiendo imágenes de otros usuarios al medio real para hacer más realistas las videoconferencias [BK02].

- *Aplicaciones comerciales*.

  Recientemente la Realidad Aumentada se está empleando para ofertar anuncios en vídeo sobre eventos deportivos y mostrando anuncios estáticos en medio de la escena real, ofreciendo a los anunciantes nuevas formas de presentar sus productos. Marcas como Adidas han apostado por emplear la realidad aumentada como reclamo para sus clientes, incorporando en uno de sus modelos de zapatillas marcas de Realidad Aumentada.

  Numerosas marcas comerciales relacionadas con el mundo del automóvil como BMV o Toyota han llevado también la Realidad Aumentada al mundo de la publicidad, permitiendo que los posibles compradores puedan visualizar sus modelos empleando marcas y aplicaciones web [BB11].

[1] Según la clasificación de Azuma, Layar no sería una aplicación de Realidad Aumentada, ya que superpone información adicional en un plano virtual incumpliendo la tercera característica que indica que la alineación de los objetos sintéticos debe realizarse en el espacio 3D.

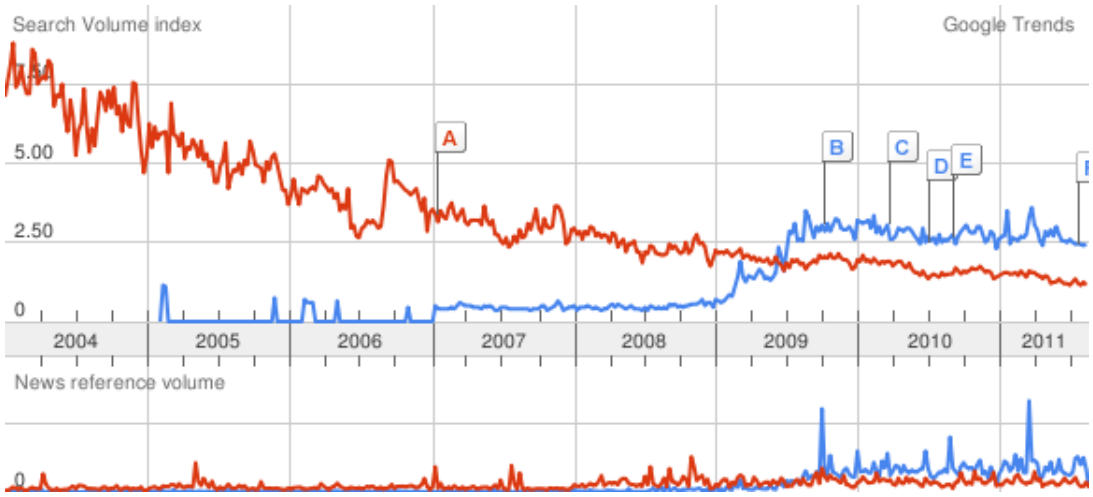

Figura 1.2: Comparativa de búsquedas entre *Augmented Reality* y *Virtual Reality* según *Google Trends*. En rojo aparece el volumen de búsquedas bajo el término de Realidad Virtual. Se observa como el interés por la Realidad Aumentada, en azul, ha ido creciendo en los últimos años, desbancando en Junio del 2009 a la Realidad Virtual en el número de búsquedas y presentando una gráfica creciente.

Otra aplicación de la realidad virtual es el uso de probadores virtuales a modo de espejo en el que el usuario puede probarse ropa o calzado de forma virtual. Para ello, esta aplicación calcula las transformaciones necesarias para que el producto concuerde con la zona del cuerpo donde va a ser superpuesto [ERF07].

1.2. Impacto socio-económico

La Realidad Aumentada está experimentando en los últimos años un crecimiento muy fuerte. Si se compara la Realidad Aumentada con la Realidad Virtual en el número de búsquedas en internet, vemos que la primera presenta un interés mayor y en crecimiento por parte de los usuarios (ver Figura 1.2). Hay multitud de factores que contribuyen a este crecimiento, algunos de ellos basados en que el mercado de la telefonía móvil. Las empresas que desarrollan terminales móviles apuestan por los *smartphones*. Estos dispositivos poseen hardware específico como la brújula electrónica, sensores inerciales, cámara, GPS con un coste relativamente bajo. Estos elementos, junto con una conexión a Internet pueden determinar una posición en el espacio y aportar información adicional.

Según la consultora de telefonía móvil *Juniper*, los beneficios generados por aplicaciones de Realidad Aumentada ascenderán a los 150 millones de dólares en los próximos cuatro años [Hol11]. Otra consultora americana, *ABI Research* llega a decir que los beneficios podrían aumentar de los 6 millones de dólares que generaron en 2009 hasta los 350 millones de dólares en 2014, lo que significa un crecimiento económico anual de un 97 % [Mad09]. Estas cifras indican que hay un mercado en auge alrededor de la Realidad Aumentada, teniendo un impacto económico de importantes dimensiones.

1.3. Realismo en aplicaciones de Realidad Aumentada

Tal y como indica Madsen [ML10], para que una aplicación consiga integrar de forma realista un objeto virtual en una escena real debe enfrentarse a tres retos:

- *El tracking de la cámara*. Este problema define la dificultad de conseguir alinear de forma correcta la cámara virtual con la cámara real, adaptando los objetos virtuales al sistema de coordenadas de la escena.
- *Manejo de oclusiones*. Este problema describe la difícil tarea de determinar que partes de algunos objetos virtuales están más cerca de la cámara y por tanto tapan partes de otros objetos.
- *Consistencia de la iluminación*. Como se lleva tratando en todo el documento, para una correcta integración hay que tener un amplio conocimiento de la iluminación de la escena para poder aplicarla a objetos virtuales y conseguir así un mayor realismo.

El primer problema, el tracking de la cámara, tiene alternativas desarrolladas, aunque sigue siendo un tema de investigación y evolución. Algunas alternativas están basadas en el reconocimiento de marcas u otro tipo de objetos conocidos. Otros métodos usan sensores que determinen características como la posición o la inclinación en los distintos ejes.

El segundo problema, el manejo de la oclusiones, también presenta algunas soluciones ya definidas como el uso de buffers de profundidad o técnicas basadas en el trazado de rayos.

El tercer problema es el que pretende resolver este Proyecto de Fin de Carrera. En los últimos años se ha comenzado a investigar la integración de objetos virtuales con métodos alternativos a el uso de perspectiva. Integrar de forma adecuada un objeto requiere que este presente una sombra y una iluminación acordes con la iluminación real. Siguiendo este propósito se puede hacer una clasificación de las técnicas de realismo según el método de registro que empleen y el entorno donde las aplicaciones se ejecuten:

- *Registro de la posición mediante hardware en entornos exteriores*. Este tipo de registro es el que se realiza utilizando sensores como GPS, acelerómetros, brújulas o giróscopos. Utilizando toda la información que le proveen estos sensores y conociendo la fecha y hora es capaz de determinar la posición del Sol y simular la iluminación que ejerce sobre objetos virtuales [EZ11]. Este tipo de iluminación es la más adecuada para aplicaciones ejecutadas en smartphones.
- *Registro de la posición mediante software en entornos interiores*. El registro se realiza mediante marcas o detección de patrones. Este tipo de aplicaciones se basan en el análisis de los elementos lumínicos de la escena destacando:

- Análisis de la sombra. Estas aplicaciones hacen un análisis de las sombras de una escena utilizando visión por computador [HHD00] para determinar la posición del foco de luz .

- Análisis de elementos lumínicos. Estas aplicaciones realizan un estudio de algún objeto conocido en el que se reflejan los focos de la escena. Empleando diversos algoritmos son capaces de determinar la posición de dichos focos [NS10].

1.4. Problemática

Como se describió en la sección 1.1, Azuma indica que la primera característica que debe cumplir una aplicación de Realidad Aumentada es que integre objetos virtuales en un entorno real. Para maximizar la experiencia de interacción, esta composición debe realizarse con el mayor grado de realismo posible.

Tal y como señala Ferweda [Fer03], el realismo gráfico en síntesis de imagen por computador se basa principalmente en la proyección en perspectiva (teniendo en cuenta los parámetros intrínsecos y extrínsecos de la cámara virtual) y en los modelos de simulación de la iluminación.

La proyección en perspectiva realista se resuelve de forma directa en aplicaciones de Realidad Aumentada al tratar con el problema del *registro*. Empleando métodos de tracking se realiza un seguimiento de elementos conocidos del mundo real para obtener posteriormente el posicionamiento de la cámara real. De esta manera, para que la representación en perspectiva de los objetos del mundo virtual se alineen correctamente con la escena del mundo real, bastará con ajustar los parámetros asociados a la cámara virtual.

Las aplicaciones de Realidad Aumentada existentes tratan con el problema de la proyección en perspectiva. Sin embargo, como se ha comentado anteriormente, para maximizar el realismo gráfico es necesario una simulación adecuada de la iluminación. Este aspecto requiere definir métodos que permitan analizar las características principales de la iluminación del mundo real para aplicarla posteriormente a los objetos virtuales. El resultado, en términos de sombras, intensidad de la iluminación y color complementará los aspectos de realismo que no son cubiertos con un modelo exclusivamente de proyección [TPPP08].

El presente Proyecto de Fin de Carrera pretende dar solución al segundo principio de realismo que habitualmente no es abordado en aplicaciones de Realidad Aumentada. En *WILi* se realiza un análisis de la fuente de luz principal del mundo real para posteriormente aplicar las propiedades de iluminación a los objetos del mundo virtual. Para permitir la integración realista en tiempo real (segunda característica requerida en aplicaciones de Realidad Aumentada según Azuma), el modelo de proyección realista de sombras difusas es precalculado de forma que se selecciona de un repositorio de texturas la más adecuada en tiempo de ejecu-

ción. Para cumplir el último requisito, *WILi* emplea una aproximación de registro absoluto basado en marcas, que proporciona un posicionamiento de precisión en el espacio 3D.

1.5. Estructura del documento

Este Proyecto de Fin de Carrera ha sido escrito siguiendo los principios expuestos en la normativa aprobada por la Junta del Centro de la Escuela Superior de Informática de Ciudad Real (Universidad de Castilla-La Mancha) el 8 de Noviembre de 2007 [dI07].

El contenido se reparte en los siguientes capítulos:

- **Capítulo 2. Antecedentes**

  En este capítulo se describen, agrupados en áreas, algunos de los conceptos más importantes que han sido necesarios para poder abordar el Proyecto de Fin de Carrera.

- **Capítulo 3. Objetivos**

  En este capítulo se definen los objetivos que se han pretendido abarcar en *WILi*.

- **Capítulo 4. Método de trabajo**

  Aborda la metodología de trabajo seguida, además de especificar el hardware y el software empleado en su desarrollo.

- **Capítulo 5. Arquitectura**

  En este capítulo se describen los detalles técnicos relativos al diseño e implementación de *WILi* siguiendo una descripción modular con un nivel de detalle incremental.

- **Capítulo 6. Evolución y costes**

  En este capítulo se describen con detalle las iteraciones que han sido necesarias para el desarrollo de *WILi*, además de realizar un informe sobre los costes de dicho desarrollo.

- **Capítulo 7. Conclusiones y propuestas**

  En este capítulo se da una visión general del Proyecto de Fin de Carrera, tanto a nivel técnico como personal, demás de completarlo con unas líneas de trabajo futuras.

CAPÍTULO 2

ANTECEDENTES

EL diseño e implementación de *WILi* requiere unos conocimientos sólidos y específicos en diversas ramas de la informática. Informática gráfica, una importante base matemática y visión por computador constituyen los ejes centrales sobre los que se apoya *WILi*.

En la figura 2.1 puede verse un esquema que recoge las distintas áreas investigadas para desarrollar el Proyecto de Fin de Carrera.

En este capítulo se abordarán los distintos campos y materias estudiadas que forman la base teórica del presente proyecto. Además se analizarán las bibliotecas y software existente que fueron examinados y comparados para su posterior utilización en el desarrollo de *WILi*.

2.1. Fundamentos matemáticos

La informática gráfica requiere de unos conocimientos matemáticos específicos en el álgebra lineal. En esta sección se definirán los principales conceptos basados en teoría vectorial, tanto en $\Re^2$ como en $\Re^3$ necesarios para realizar, entre otras operaciones, cálculos de distancias, proyecciones y ángulos (ver sección 2.1.1).

Por otro lado se revisarán determinados conceptos de la teoría matricial, ya que por cuestiones de eficiencia las transformaciones de los objetos virtuales son representadas mediante

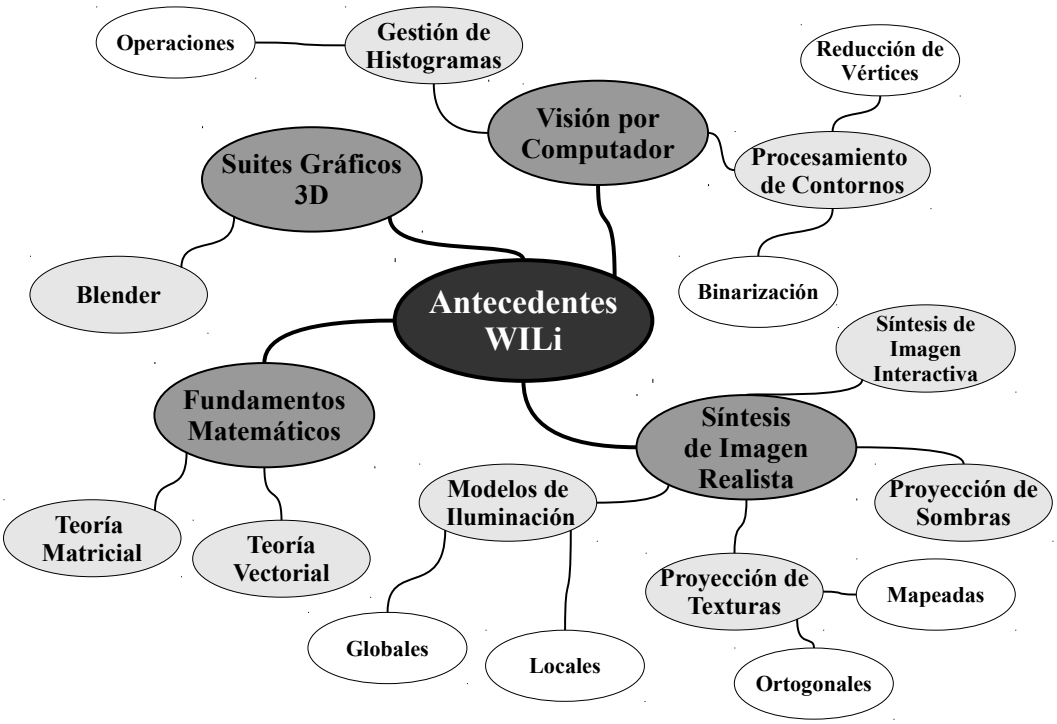

Figura 2.1: Mapa conceptual de *WILi*.

matrices. En la sección 2.1.2 se describirán con detalle las operaciones de cuerpo rígido (rotación, escalado y traslación) de objetos tridimensionales.

2.1.1. Teoría vectorial

Un vector es un segmento de recta dirigido en el espacio que presenta:

- *Módulo*: Determina la longitud del vector.
- *Origen*: Es el punto sobre el que actúa el vector.
- *Dirección*: Dada por la recta que lo contiene
- *Sentido*: Indica hacia que lado de la recta actúa el vector.

A continuación se verán algunas de las operaciones más importantes realizadas con vectores.

Cálculo de un vector a partir de dos puntos

Dados dos puntos en $\Re^3$ $A(a_x, a_y, a_z)$ y $B(b_x, b_y, b_z)$, se puede definir el vector $\overrightarrow{AB}$ como:

$$\overrightarrow{AB} = \{b_x - a_x, b_y - a_y, b_z - a_z\} \tag{2.1}$$

Módulo de un vector

El módulo de un vector $\vec{u}$ corresponde a su longitud y viene dado por la siguiente expresión:

$$|\vec{u}| = \sqrt{u_1^2 + u_2^2 + u_3^2} \tag{2.2}$$

, donde u_1, u_2, u_3 son las componentes del vector $\vec{u}$.

Producto escalar

Dado un vector $\vec{u}$ y un vector $\vec{v}$ que forman un ángulo θ, el producto escalar denotado por $\vec{u} \cdot \vec{v}$ es el módulo del primero, $|\vec{u}|$, por el módulo del segundo,$|\vec{v}|$, por el coseno del ángulo que forman.

$$\vec{u} \cdot \vec{v} = |\vec{u}||\vec{v}| \cos\theta \tag{2.3}$$

También puede expresarse como la suma del producto entre los componentes del vector:

$$\vec{u} \cdot \vec{v} = \vec{u}_1\vec{v}_1 + \vec{u}_2\vec{v}_2 \tag{2.4}$$

Una propiedad que cumple el producto escalar es que, si los dos vectores son perpendiculares entre si, el producto escalar de los mismos es cero:

$$\vec{u} \perp \vec{v} \rightarrow \vec{u} \cdot \vec{v} = 0$$

En cualquier caso el producto escalar de dos vectores tiene como resultado un escalar.

Proyección de un vector sobre otro

Siguiendo la definición de [Rid96] y teniendo en cuenta la Figura 2.2, la proyección de un vector $\vec{u}$ sobre un vector $\vec{v}$ es un vector $\vec{w}$ tal que si $\overrightarrow{AB}$ representa a $\vec{u}$ y $\overrightarrow{AC}$ representa a $\vec{v}$, entonces $\vec{w}$ se representa por un segmento de recta dirigido $\overrightarrow{AD}$ que está sobre la recta determinada por $\overrightarrow{AC}$, siendo $\overrightarrow{BD}$ perpendicular a esa recta.

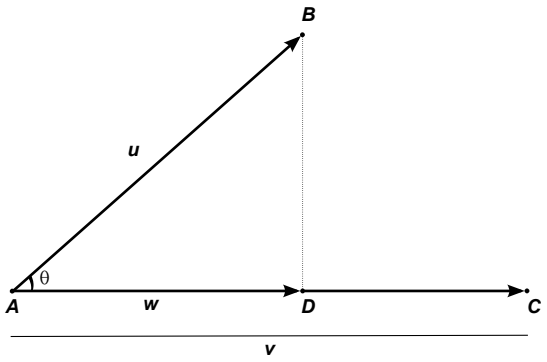

Figura 2.2: Proyección de un vector $\vec{u}$ sobre $\vec{v}$.

Por trigonometría,

$$\cos\theta = \frac{|w|}{|u|} \rightarrow \quad |w| = |u|\cos\theta;$$

$$|w| = |u|\cos\theta\frac{|v|}{|v|} \rightarrow |w| = \frac{|u||v|\cos\theta}{|v|};$$

El numerador de puede ser simplificado empleando la ecuación 2.3 que corresponde al producto escalar con el siguiente resultado:

$$|w| = \frac{\vec{u}\cdot\vec{v}}{|v|} \tag{2.5}$$

Por tanto se puede definir $\vec{w}$, ya que se conoce su módulo (ecuación 2.5) y su dirección está determinada por $\vec{v}$, porque la proyección se hace sobre ese vector.

$$\vec{w} = \left(\frac{\vec{u}\cdot\vec{v}}{|v|}\right)\frac{\vec{v}}{|v|} \tag{2.6}$$

Distancia entre dos puntos

Dado un punto A y un punto B, la distancia entre ambos puntos puede ser calculada como el módulo del vector $\overrightarrow{AB}$.

$$dist(A,B) = |\overrightarrow{AB}| \tag{2.7}$$

Distancia de un punto a una recta

Supóngase un punto $A = (a_x, a_y)$ y una recta dada por la ecuación paramétrica $L(t) = P - t\vec{d}$.

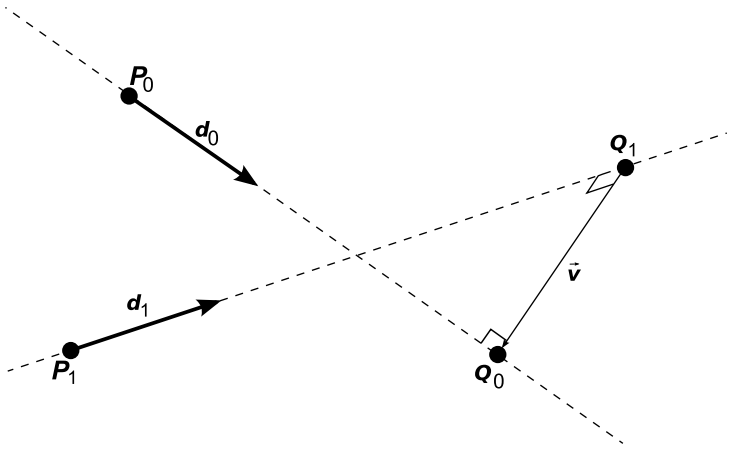

Figura 2.3: Distancia mínima entre dos rectas en $\Re^3$.

La distancia del punto A a la recta $L(t)$ corresponde al módulo del vector cuya dirección viene determinada por el punto.

Distancia mínima entre dos rectas

Para esta operación se usará la demostración de [SE03].

Dadas dos rectas $L_0(s) = P_0 + s\vec{d_0}$ y $L_1(t) = P_1 + s\vec{d_1}$, se toman dos puntos $Q_0 = P_0 + s_c\vec{d_0}$ y $Q_1 = P_1 + t_c\vec{d_1}$ de forma que la distancia entre los mismos sea la mínima distancia entre las dos rectas anteriores. Se determina también el vector $\vec{v}$ como $\vec{v} = Q_0 - Q_1$ (ver Figura 2.3).

Hay que destacar que $\vec{v}$ es perpendicular a L_0 y a L_1 cuando $\vec{v}$ es minimizado, por lo que:

$$\vec{d_0} \cdot \vec{v} = 0 \tag{2.8}$$

$$\vec{d_1} \cdot \vec{v} = 0 \tag{2.9}$$

, deben satisfacerse. Esto será útil para calcular s_c y t_c.

Expandiendo la definición de $\vec{v}$:

$$\vec{v} = Q_0 - Q_1 = P_0 + s_c\vec{d_0} - P_1 + t_c\vec{d_1}$$

, y sustituyendo con las ecuaciones 2.8 y 2.9 se obtiene:

$$(\vec{d_0} \cdot \vec{d_0})s_c - (\vec{d_0} \cdot \vec{d_1})t_c = -\vec{d_0} \cdot (P_0 - P_1)$$

$$(\vec{d_1} \cdot \vec{d_0})s_c - (\vec{d_1} \cdot \vec{d_1})t_c = -\vec{d_1} \cdot (P_0 - P_1)$$

Para simplificar el problema, se define como $a = \vec{d_0} \cdot \vec{d_0}$, $b = -\vec{d0} \cdot \vec{d1}$, $c = \vec{d1} \cdot \vec{d1}$, $d = \vec{d_0} \cdot (P_0 - P_1)$, $e = \vec{d1} \cdot (P_0 - P_1)$ y $f = (P_0 - P_1) \cdot (P_0 - P_1)$. Esto deja una ecuación con dos incógnitas cuya solución es:

$$s_c = \frac{be - cd}{ac - b^2}$$

$$t_c = \frac{bd - ae}{ac - b^2}$$

Si el denominador $ac - b^2 < \epsilon$, donde ϵ es un número que tiende a 0, entonces L_0 y L_1 son paralelas. En este caso se toma t_c como 0, reduciendo el cálculo de s_c a $s_c = -d/a$.

Una vez calculados los valores de los puntos más cercanos, se puede calcular la distancia entre L_0 y L_1.

$$|L_0(s_c) - L_1(t_c)| = (P_0 - P_1) + \frac{(be - cd)\vec{d_0} - (bd - ae)\vec{d_1}}{ac - b^2} \tag{2.10}$$

En el caso de que la distancia entre las dos rectas sea nula, se podrá determinar que las rectas son secantes.

Pertenencia de un punto a una recta

Para determinar si un punto P pertenece a una recta se pueden usar, al menos dos planteamientos distintos:

- *Cálculo por distancia a la recta*. Un punto pertenece a una recta si la distancia de ese punto a la recta es 0.
- *Cálculo por ecuación general de la recta*

  Dado un punto en $\Re^2$ $P = (p_x, p_y)$, P pertenece a la recta definida por una ecuación general $Ax + By + C = 0$ si cumple

$$Ap_x + Bp_y + C = 0$$

Ángulo que forman dos rectas

Tomando la ecuación 2.3 se puede calcular el ángulo que forman dos rectas despejando θ:

$$\vec{u} \cdot \vec{v} = |\vec{u}||\vec{v}| \cos\theta \rightarrow \theta = \arccos\left(\frac{\vec{u} \cdot \vec{v}}{|\vec{u}||\vec{v}|}\right)$$

, aplicando la ecuación 2.4, podemos sustituir $\vec{u} \cdot \vec{v}$, obteniendo la ecuación resultante:

$$\theta = \arccos\left(\frac{\vec{u}_1\vec{v}_1 + \vec{u}_2\vec{v}_2}{|\vec{u}||\vec{v}|}\right) \tag{2.11}$$

2.1.2. Teoría matricial

Una matriz es un conjunto de elementos ordenados en filas y columnas. En informática gráfica, las matrices son empleadas, entre otras cosas, para expresar las transformaciones aplicables a objetos tridimensionales.

Las transformaciones aplicadas a los objetos virtuales están representadas por una matriz resultante llamada *matriz de transformación neta*. Dicha matriz es el resultado de componer, usando la multiplicación, todas las operaciones aplicables a un objeto expresadas mediante matrices.

A continuación se definen las principales operaciones de transformación y su representación matricial asociada.

Suma de matrices

Sean dos matrices A y B del mismo tamaño de la forma:

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

, se define la suma, denotado como $A + B$ a la operación:

$$A + B = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \cdots & a_{2n} + b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \cdots & a_{mn} + b_{mn} \end{bmatrix} \tag{2.12}$$

Multiplicación de matrices

Sea una matriz C que representa una fila de una matriz y una matriz D que representa una columna de la forma:

$$C = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix} \quad \text{y} \quad D = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix}$$

El resultado de multiplicar una fila por una columna es:

$$C * D = \left[\; c_1 * d_1 + c_2 * d_2 + \cdots + c_n * d_n \; \right]$$

Una vez descrita esta operación básica y, denotando una fila n de una matriz como X_{fn} y una columna n como X_{cn}, puede definirse la multiplicación de dos matrices A y B del mismo tipo que las expresadas en la suma de matrices como:

$$A * B = \begin{bmatrix} A_{f1} * B_{c1} & A_{f1} * B_{c2} & \cdots & A_{f1} * B_{cn} \\ A_{f2} * B_{c1} & A_{f2} * B_{c2} & \cdots & A_{f2} * B_{cn} \\ \vdots & \vdots & \ddots & \vdots \\ A_{fn} * B_{c1} & A_{fn} * B_{c2} & \cdots & A_{fn} * B_{cn} \end{bmatrix} \tag{2.13}$$

Escalado de un vértice

La operación de escalado puede representarse con la siguiente matriz homogénea:

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

De forma genérica, se puede calcular el resultado de aplicar una operación de traslación a un punto en $\Re^3$, $P = (p_x, p_y, p_z)$:

$$\begin{bmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \Rightarrow \begin{cases} p'_x = p_x * S_x \\ p'_y = p_y * S_y \\ p'_z = p_z * S_z \end{cases}$$

, obteniendo, por tanto un nuevo punto $P' = (p'_x, p'_y, p'_z) = (p_x * S_x, p_y * S_y, p_z * S_z)$.

Traslación de un vértice

La operación de traslación puede representarse con la siguiente matriz homogénea:

$$T = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

De forma genérica, se puede calcular el resultado de aplicar una operación de traslación a un punto en $\Re^3$, $P = (p_x, p_y, p_z)$:

$$\begin{bmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \Rightarrow \begin{cases} p'_x = p_x + T_x \\ p'_y = p_y + T_y \\ p'_z = p_z + T_z \end{cases}$$

El nuevo punto con las operaciones de traslación viene definido por $P' = (p'_x, p'_y, p'_z) = (p_x + T_x, p_y + T_y, p_z + T_z)$.

Rotación de un vértice

Las matrices de transformación de las rotaciones dependen del eje en el que quiera aplicarse:

- **Rotación en el eje x**

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\operatorname{sen}\theta & 0 \\ 0 & \operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

De forma genérica, se puede calcular el resultado de aplicar una operación de rotación en el eje x a un punto en $\Re^3$, $P = (p_x, p_y, p_z)$:

$$\begin{bmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\operatorname{sen}\theta & 0 \\ 0 & \operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \Rightarrow \begin{cases} p'_x = p_x \\ p'_y = p_y \cos\theta - p_z \operatorname{sen}\theta \\ p'_z = p_y \operatorname{sen}\theta + p_z \cos\theta \end{cases}$$

El nuevo punto con las operaciones de rotación en el eje x viene definido por $P' = (p'_x, p'_y, p'_z) = (p_x, p_y \cos\theta - p_z \operatorname{sen}\theta, p_y \operatorname{sen}\theta + p_z \cos\theta)$.

- **Rotación en el eje y**

$$R_y = \begin{bmatrix} \cos\theta & 0 & \operatorname{sen}\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\operatorname{sen}\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

De forma genérica, se puede calcular el resultado de aplicar una operación de rotación en el eje y a un punto en $\Re^3$, $P = (p_x, p_y, p_z)$:

$$\begin{bmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \operatorname{sen}\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\operatorname{sen}\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \Rightarrow \begin{cases} p'_x = p_x \cos\theta + p_z \operatorname{sen}\theta \\ p'_y = p_y \\ p'_z = -p_x \operatorname{sen}\theta + p_z \cos\theta \end{cases}$$

El nuevo punto con las operaciones de rotación en el eje y viene definido por $P' = (p'_x, p'_y, p'_z) = (p_x \cos\theta + p_z \operatorname{sen}\theta, p_y, -p_x \operatorname{sen}\theta + p_z \cos\theta)$.

- **Rotación en el eje z**

$$R_z = \begin{bmatrix} \cos\theta & -\operatorname{sen}\theta & 0 & 0 \\ \operatorname{sen}\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

De forma genérica, se puede calcular el resultado de aplicar una operación de rotación en el eje z a un punto en $\Re^3$, $P = (p_x, p_y, p_z)$:

$$\begin{bmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\operatorname{sen}\theta & 0 & 0 \\ \operatorname{sen}\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \Rightarrow \begin{cases} p'_x = p_x \cos\theta - p_y \operatorname{sen}\theta \\ p'_y = p_x \operatorname{sen}\theta + p_y \cos\theta \\ p'_z = p_z \end{cases}$$

El nuevo punto con las operaciones de rotación en el eje z viene definido por $P' = (p'_x, p'_y, p'_z) = (p_x \cos\theta - p_y \operatorname{sen}\theta, p_x \operatorname{sen}\theta + p_y \cos\theta, p_z)$.

Estas operaciones se emplean, entre otras cosas, en la adaptación de los objetos virtuales para simular la visión en perspectiva. Sin embargo como se determinó en la sección 1.4, hay otros parámetros como la iluminación o el uso de texturas que consiguen aumentar el realismo de las escenas sintetizadas. En la siguiente sección se definen algunos de los elementos y técnicas empleados para este propósito.

2.2. Síntesis de imagen realista

En la actualidad, la síntesis de imagen realista se basa en el uso de algoritmos y técnicas que emulan el comportamiento de las partículas de luz en la escena. Algunas de estas técnicas como el *ray tracing* emplean algoritmos que simulan el rebote de las partículas de luz en las superficies que forman la escena, obteniendo el color asociado a cada *píxel* que forma el plano imagen.

Sin embargo, el principal problema que presentan estas técnicas de *render* realista es la enorme cantidad de tiempo necesario para realizar cada simulación. Con los medios hardware disponibles hoy en día, los algoritmos de simulación realista de iluminación no pueden ser ejecutados en tiempo real debido a su elevado coste computacional.

Puesto que la segunda característica de Azuma indica que la integración del objeto virtual debe hacerse en tiempo real, estas técnicas no pueden emplearse directamente sobre el objeto en la escena generando una imagen en tiempo real. En su lugar, métodos como el *texture baking* consiguen generar texturas que representen la iluminación atribuida a un objeto en una escena sintética de forma que en tiempo real puedan desplegarse dichas texturas sobre el objeto.

2.2.1. Modelos de iluminación

En esta sección se describen algunos de los principios básicos de iluminación que sirven de base teórica para el desarrollo de *WILi*.

En el renderizado de la imagen, la luz adquiere un papel fundamental en la simulación de la iluminación que permitirá dar una apariencia tridimensional a la escena. Sin embargo, es necesario establecer un compromiso entre el tiempo empleado por los distintos algoritmos en calcular la escena y entre la calidad de la imagen obtenida.

Atendiendo a la simulación de la interacción de la luz sobre las superficies, los modelos de iluminación pueden clasificarse en dos tipos:

- Modelos de iluminación local.
- Modelos de iluminación global.

A continuación se describe cada uno de ellos, además de mostrar algunos de los algoritmos más significativos de cada tipo.

Modelos de iluminación local

Los modelos de iluminación local corresponden a los métodos más sencillos computacionalmente y pueden ser empleados para simular la iluminación de una escena en tiempo real.

Estos modelos se caracterizan por emplear la simulación del primer rebote de la luz sobre una superficie.

A pesar de poder generar la iluminación de la escena en tiempo real, estas técnicas no consiguen el mismo realismo que los siguientes modelos de iluminación global.

Modelos de iluminación global

Los modelos de iluminación global se caracterizan por simular las interacciones de luz entre superficies.

Estos modelos de iluminación son más realistas que los modelos de iluminación local, pero también son más pesados computacionalmente, invirtiendo más cantidad de tiempo en terminar el cálculo de la iluminación.

Algunos de los modelos más usados son:

- *Ray tracing*. También conocido como trazado de rayos, este modelo sólo tiene en cuenta los rayos que llegan al punto de vista. Para poder estimar estos rayos, el modelo sigue el proceso inverso, es decir, parte del punto de vista generando rayos hacia los objetos estudiando los rebotes provocados ante el choque de un rayo con un objeto de la escena. De esta forma consigue simular escenas foto-realistas.

- *Radiosity*. Se trata de un modo de iluminación independiente del punto de vista consistente en el estudio de la interacción difusa de la luz entre objetos de la escena subsivididos en pequeños polígonos llamados *patchs* (o parches). La cantidad de subdivisiones contribuye a la calidad de la escena generada a la vez que aumenta el tiempo de cómputo.

- *Otros*. Otras aproximaciones como *pathtracing* (simple o bidireccional), o los métodos de transporte de luz de metrópolis (MLT) se basan en aproximaciones de MonteCarlo para simular los rebotes de luz en superficies, mediante la definición de funciones específicas para cada tipo de material.

A pesar de no poder emplear estos algoritmos en el cálculo de la iluminación en tiempo real, sí es posible aplicar los resultados de los mismos empleando técnicas como el *texture baking*.

El *texture baking* utiliza texturas para mapear la luz que recibe un objeto. El desplegado de una textura sobre un objeto no presenta una carga de cómputo a tener en cuenta y el objeto quedaría iluminado de forma más realista.

Gran cantidad de aplicaciones basadas en gráficos 3D como los videojuegos, hacen un ligero cálculo de la iluminación para generar sombras de elementos dinámicos en las escenas.

El resto de la iluminación de las mismas se basan en la asignación de texturas precalculadas.

Estas técnicas también son empleadas en la Realidad Aumentada, ya que uno de los requisitos fundamentales de estas aplicaciones es que la composición de los objetos virtuales y la escena se realice en tiempo real (ver 1.1).

De esta manera, los objetos ganan en realismo por presentar unas propiedades lumínicas precisas acordes con el entorno en el que se encuentran sin hacer grandes inversiones en cálculos complejos.

En esta sección se han estudiado los distintos modos de iluminación, tanto locales como globales. Éstos requieren de técnicas basadas en mapeado de texturas para poder simular el uso de técnicas de iluminación global en aplicaciones de tiempo real. A continuación, en la siguiente sección, se tratarán las texturas y se definirá cómo se mapean a partir de objetos.

2.2.2. Proyección de texturas

Cuando se trabaja en proyectos de informática gráfica, es muy común hablar de materiales y texturas. Un material define una serie de propiedades que son constantes a lo largo de toda la superficie, mientras que una textura es una imagen que se despliega sobre el mismo para simular, superficies heterogéneas, algo más cercano a la realidad.

Texturizar por tanto un objeto consiste en desplegar una imagen bidimensional para que su superficie adquiera una determinada apariencia.

El tiempo de cómputo empleado para realizar operaciones sobre un objeto virtual es directamente proporcional al número de vértices que intervienen en dichas operaciones. Este motivo hace que actualmente en multitud de videojuegos los objetos virtuales estén modelados con un número de vértices mínimo. Sin embargo, con este número reducido de vértices no se puede representar con gran realismo dichos objetos. Es aquí donde el texturizado brinda la opción de mitigar la falta de sensación de realismo, pudiendo ampliar la geometría de forma visual, sin aumentar los vértices[AMHH08].

La aplicación de texturas (2D) sobre un objeto (3D) requiere del uso de métodos de proyección . Existen dos tipos de métodos de proyección de texturas:

- *Ortogonales*: Se proyectan de forma automática utilizando las coordenadas del objeto 3D. En el caso de modelos con geometrías complicadas, la proyección ortogonal no es válida para desplegar texturas que presenten dibujos específicos para cada cara del mismo.

- *Mapeadas*: Este tipo de texturas establece una equivalencia entre los vértices del modelo y unas coordenadas de la textura, de manera que se despliegue como si la textura

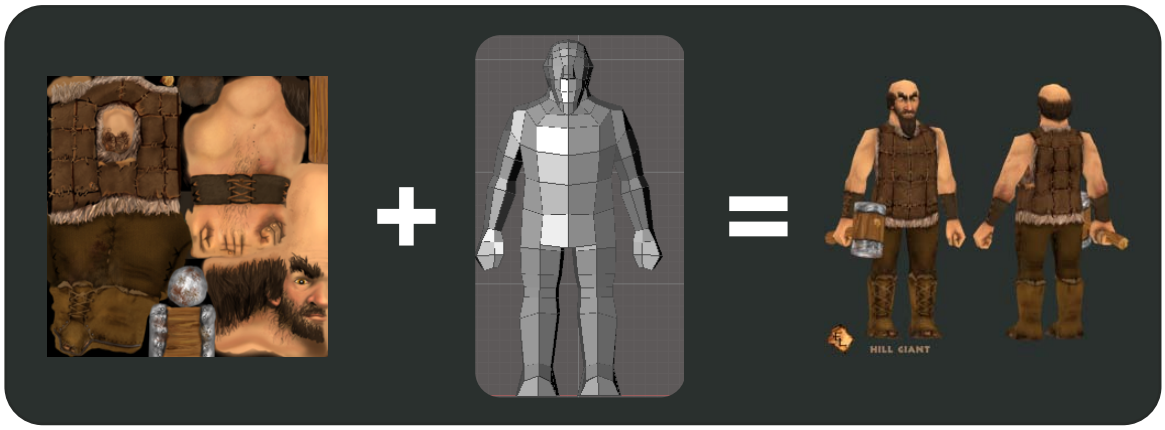

Figura 2.4: Ejemplo de aplicación de un mapa UV. En la imagen se observa cómo la textura se despliega envolviendo al objeto siguiendo una coordenada *uv* para cada vértice del mismo.

actuase como una piel del modelo, haciendo coincidir cada vértice del mismo con una coordenada bidimensional de la textura.

El nombre *uv mapping* proviene del empleo de coordenadas bidimensionales para definir la relación entre vértices y la textura. Las coordenadas *uv* vienen definidas por una coordenada *u*, en el eje x y una coordenada *v*, en el eje y. En la Figura 2.4 puede verse como se despliega una textura en un modelo siguiendo esta técnica.

Una de las características que pueden almacenarse en las texturas mapeadas con coordenadas *uv* es la información referente a las sombras precalculadas. Es habitual en videojuegos precalcular la proyección de sombras estáticas y almacenarlas en texturas. Para realizar esta etapa, es necesario conocer los métodos de proyección de sombras más utilizados en informática gráfica. La siguiente sección introduce los principales conceptos relacionados con la proyección de sombras

2.2.3. Modos de proyección de sombras

Como indica [TPPP08] , la sombra no es simplemente un elemento que agudiza el realismo de una escena modelada. La sombra aporta a la vista información como la profundidad de los objetos en un entorno dado o la distancia real que presentan unos con otros. Este tipo de información facilita la visión estereoscópica del usuario a partir de una imagen plana. Sin esta información pueden surgir ambigüedades sobre el tamaño, posición, profundidad y posición relativa con otros objetos.

La Figura 2.5(a) muestra un objeto sin sombra. El espectador no es capaz de determinar si el objeto flota en el aire o, si por el contrario, está apoyado sobre el suelo. Dado que se trata de una imagen plana, la falta de sombra produce estas ambigüedades. La Figura 2.5(b) sí presenta sombra, ayudando al observador a determinar que está en el aire. De la misma manera, la sombra de la Figura 2.5(c) establece que el objeto está en el suelo.

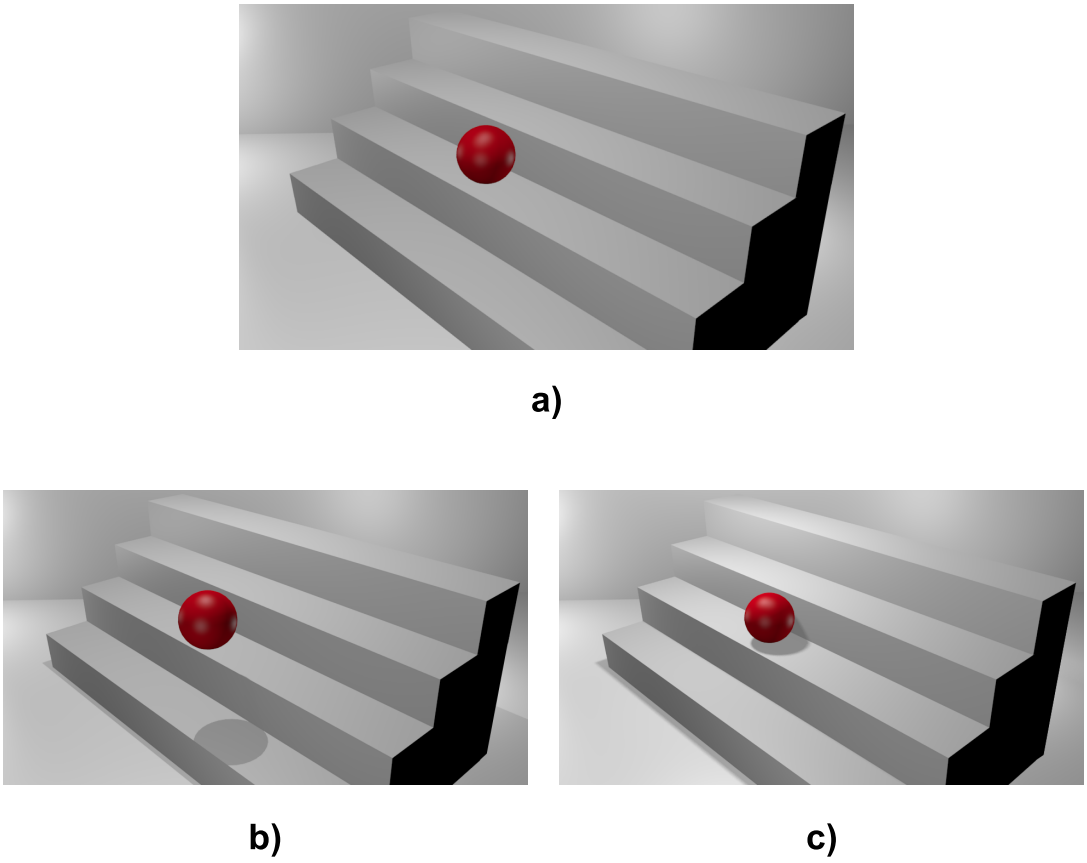

Figura 2.5: Posición de un objeto en función de su sombra. En la imagen (a) se observa cómo un objeto parece flotar en el aire sin tener una posición clara para el espectador. En la imagen(b), la sombra indica que el objeto está en el aire sobre los escalones. En la imagen (c), la sombra determina que el objeto está apoyado en el escalón.

Las sombras se generan en las superficies cuando un objeto se interpone entre un foco de luz y dicha superficie, impidiendo que los rayos de luz lleguen a ella. En la Figura 2.6(a) se puede ver cómo se genera la sombra del cubo. En primer lugar, el foco emite rayos de luz. Los rayos que no inciden con el objeto iluminan la superficie, mientras que los que inciden en el cubo causan un vacío en la iluminación de la superficie (la sombra). Como se puede apreciar, los vértices de la sombra están alineados con los vértices del objeto que la genera. De esta forma podría seguirse el proceso inverso, es decir, en lugar de calcular la sombra a partir de un objeto y un foco de luz, calcular el foco de luz a partir del objeto y la sombra que arroja. Siguiendo la Figura 2.6, si se traza una recta desde cada vértice de la sombra y que pase por el vértice del objeto que lo genera, estas rectas intersecan en un único punto, el foco.

También hay que destacar que esta técnica de trazado de rayos inverso puede emplearse siempre que la sombra esté perfectamente definida. Sin embargo, en la realidad esto no sucede. Las sombras están formadas por una parte en la que los rayos del foco de luz son

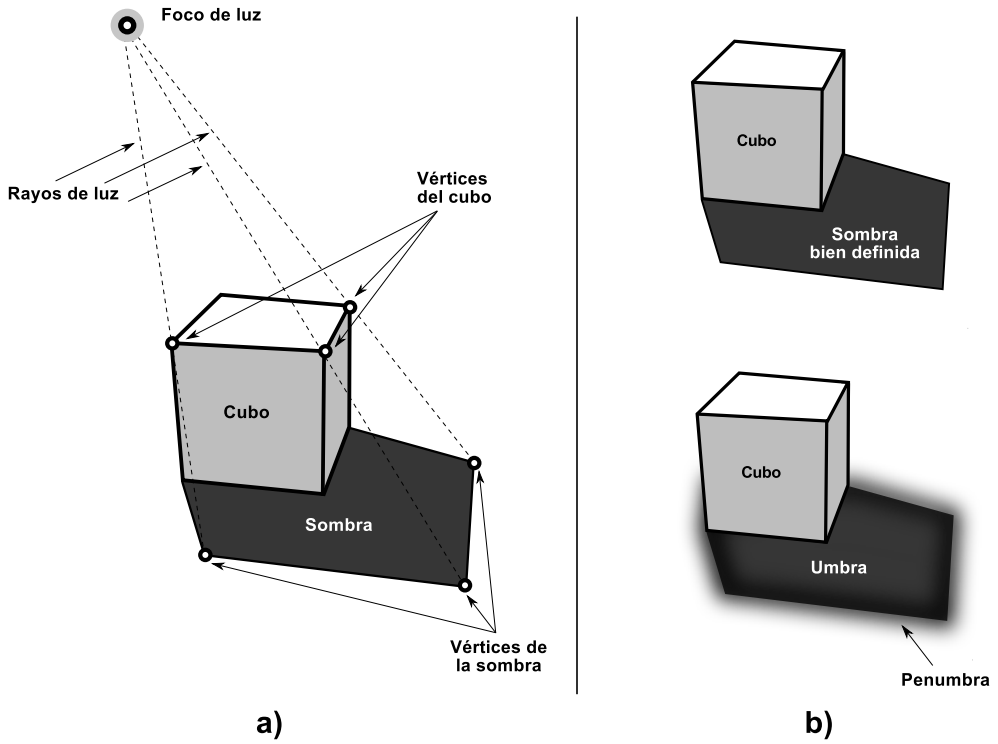

Figura 2.6: Proceso de generación de sombras y componentes de la misma. En (a) se muestra un esquema para esclarecer los aspectos que entran en juego en la generación de la sombra. En la imagen (b) superior aparece una sombra perfecta con los bordes bien marcados. En la inferior una sombra real con zonas de umbra y penumbra.

totalmente bloqueados por el objeto a la que se le denomina *umbra*. La otra parte, denominada *penumbra*, presenta una componente difusa formada por algunos rayos de la fuente de luz, es decir, el bloqueo de los rayos ha sido parcial y por tanto no genera una sombra tan marcada como la *umbra*.

Como se ve en la Figura 2.6(b), la imagen superior presenta el caso ideal en el que la sombra tiene un contorno perfectamente definido. Por el contrario, la imagen inferior tiene una sombra con bordes poco definidos como consecuencia de la existencia de zonas de penumbra.

El grado de penumbra de una sombra depende en gran medida de la fuente de luz. Aspectos como el tamaño, intensidad o la posición de la fuente no solo determinan el tamaño de la sombra, sino también la componente difusa de la misma.

A continuación se explicarán dos modelos empleados en el cálculo de sombras:

- *Ray tracing*. El cálculo de la sombra con *ray tracing* consiste en realizar una comprobación cuando el rayo emitido desde el punto de vista choca contra un objeto. Esta comprobación consiste en emitir otro rayo desde el punto de intersección hasta la fuente de luz. Si dicho rayo choca contra otro objeto, ese punto es considerado una sombra.

- *Buffer de profundidad*. Esta técnica consiste en visualizar la escena desde el foco, almacenando la información de la profundidad de los objetos en un Z-Buffer de sombras. Una vez realizado esto para establecer la sombra se utiliza la información de distancia discretizada en el *buffer* de profundidad.

En esta sección se ha descrito la importancia de la sombra en una iluminación para conseguir una imagen realista y determinar la posición de los objetos en la escena.

2.2.4. Bibliotecas para síntesis de imagen interactiva

Las bibliotecas para síntesis de imagen interactiva proporcionan de funciones para el dibujado de objetos 3D y 2D. La Realidad Aumentada mezcla los objetos virtuales con la imagen real empleando bibliotecas de este tipo.

Para determinar cuál sería la biblioteca empleada por *WILi* para realizar todas las operaciones de gráficos 3D se realizó una comparativa entre Direct3D (D3D) y *OpenGL*. En esta breve comparativa se pretende mostrar las ventajas e inconvenientes de usar una u otra biblioteca de gráficos por computador.

En primer lugar se hará una descripción de las dos bibliotecas estándar en el mercado : *OpenGL* y Direct3D.

OpenGL

Tal y como define [Shr10], *OpenGL* es una interfaz software para gráficos por hardware consistente en más de 700 comandos distintos. Se trata, además, de una interfaz independiente del hardware. *OpenGL* trabaja como una máquina de estados. Muchos de estos estados se activan en *OpenGL* con las funciones `glEnable()` y `glDisable()`.

Las ventajas de usar esta biblioteca son, según [AMHH08] son las siguientes:

- Se trata de una biblioteca muy eficiente, avalada por multitud de proyectos que abarcan la simulación, CAD, videojuegos y un largo etcétera de campos.
- Se presenta como una biblioteca ampliamente soportada.
- Biblioteca multiplataforma tanto en Hardware como en Software.
- Es una biblioteca abierta.
- Presenta un estándar en el nombrado de las funciones. Las funciones de *OpenGL* siguen un convenio en el nombrado de las mismas para identificar su funcionamiento, número de entradas y tipo de datos de forma intuitiva especificado por ARB (The OpenGL Architecture Review Board) y posteriormente por Kronos.

El convenio establecido especifica que el nombre de una función debe especificarse con el nombre de la biblioteca seguida del nombre de la función. A continuación le sigue un entero que especifica el número de entradas de la función y el tipo de datos especificado por un carácter. Algunos ejemplos son: glVertex3f, glVertex2i glColor3f.

Para que un objeto tridimensional sea representado en una pantalla con *OpenGL*, se suceden una serie de fases o etapas en las que, cada salida de la misma sirve de entrada a la siguiente hasta obtener la imagen renderizada.

Esta serie de pasos encadenados es definido como *pipeline*. A continuación se describen todas estas etapas que emplea *OpenGL* para el renderizado de objetos virtuales.

1. **Transformación de visualización**

   En primer lugar, hay que definir la *transformación de visualización*, que consiste en asignar una posición y orientación a la cámara virtual, obteniendo el punto de vista desde el que se renderizará el objeto.

2. **Transformación de modelado**

   Una vez definida la transformación de visualización, hay que aplicar la *transformación de modelado*. Estas transformaciones pueden ser de rotación, escalado o translación y son aplicadas al modelo. Como se explica en la sección 2.1.2, cada operación aplicada a un objeto viene representada por una matriz de transformación que varía según dicha operación. Las operaciones aplicadas a un objeto pueden resumirse en una única matriz de transformación, fruto de componer las matrices de transformación empleando la multiplicación.

3. **Transformación de proyección**

   El siguiente paso es definir el tipo de proyección usado para renderizar la escena. La *transformación de proyección* determina cómo los objetos van a ser proyectados sobre la pantalla. Hay dos formas básicas de proyección de objetos:

   - *Proyección en perspectiva*: Este tipo de proyección corresponde a como se visualizan los objetos en la realidad, más pequeños cuanto más alejados de la cámara estén y más grande cuanto más cercanos se encuentren.
   - *Proyección ortográfica*: La proyección ortográfica no distingue entre objetos cercanos o lejanos haciendo que su tamaño varíe en función de este parámetro, sino que muestra todos los objetos con el mismo tamaño.

4. **Transformación de pantalla**

   Una vez definidas las transformaciones anteriores, se pasa a la fase de transformación de pantalla. Tanto la transformación de proyección como la de pantalla determinan

OpenGL

```
glBegin (GL_TRIANGLES);
    glVertex (0,0,0);
    glVertex (1,1,0);
    glVertex (2,0,0);
glEnd ();
```

Direct 3D

```
v = &buffer.vertexes[0];
 v->x = 0; v->y = 0; v->z = 0;
 v++;
 v->x = 1; v->y = 1; v->z = 0;
 v++;
 v->x = 2; v->y = 0; v->z = 0;
 c = &buffer.commands;
 c->operation = DRAW_TRIANGLE;
 c->vertexes[0] = 0;
 c->vertexes[1] = 1;
 c->vertexes[2] = 2;
 IssueExecuteBuffer (buffer);
```

Figura 2.7: Comparación de código al dibujar un triángulo con *OpenGL* y *Direct3D*

como el objeto se va a visualizar en pantalla. La transformación de pantalla indica la forma del área de la pantalla sobre la que se va a mapear el objeto. Puede entenderse esta transformación como la posición y tamaño del área donde se va a mostrar el objeto tridimensional.

Tras estos pasos, se obtiene en pantalla la imagen renderizada.

Direct3D

Se trata de una API diseñada para la programación de gráficos 3D definida por *Microsoft*. Forma parte de *DirectX*, un conjunto de bibliotecas de soporte multimedia.

En la tabla 2.1[1] se muestra una comparativa de las propiedades más comunes que deben soportar.

La principal desventaja encontrada es que *WILi* está desarrollado usando un sistema operativo *GNU/Linux* y que está bajo una licencia *GPLv3*. *Direct3D* únicamente funciona en entornos de *Microsoft Windows*, por lo que no sería viable utilizarla.

Otra ventaja importante de *Opengl* frente a *Direct3D* reside en la complejidad del código empleado para generar formas básicas. En la Figura 2.7 puede verse una ejemplo en el que se muestran las primitivas usadas por *OpenGL* y las usadas por *Direct3D* para dibujar una triángulo.

En esta sección se ha realizado una comparativa entre las dos bibliotecas más empleadas para carga y dibujado de objetos virtuales en la escena. Para que la integración se realice de la forma más realista posible, debe realizarse un estudio de la iluminación, determinando así que textura representa mejor al objeto en la escena real. La rama de la visión por computador

[1]Tabla extraida de `http://www.xmission.com/~legalize/d3d-vs-opengl.html`

Característica	**OpenGL**	**Direct3D**
Sistemas operativos soportados	Windows, MacOS, BeOS, *nix, otros	Windows
Control de la API	OpenGL ARB	Microsoft
Emulación del Software y emulación de características no aceleradas	Si	No
Mecanismos de extensión	Si	Si
Código fuente disponible	Si	No
Modelado		
Primitivas de curvas paramétricas	Si	Si
Primitivas de superficies paramétricas	Si	Si
Listas de dibujado jerárquico	Si	No
Rendering		
Iluminación a doble cara (normales dobles)	Si	No
Atributos de ancho de línea	Si	No
Pixel Shaders programables	Sí	Si
Niebla de distancia	No	Si
Bump Mapping	No	Si
Frame Buffer		
Acceso al Z-Buffer de forma independiente al Hardware	Si	No
Antialiasing	Si	Si
Motion Blur	Si	Si
Buffers de acumulación	Si	No

Cuadro 2.1: Comparativa entre *OpenGL* y *Direct3D*.

ha ido evolucionando a lo largo de los años, siendo *OpenCv* la biblioteca estándar en el análisis de imágenes. A continuación se comentarán determinados aspectos y técnicas empleadas en la visión por computador.

2.3. Visión por computador

Tal y como especifica G.Bradski en [BK08], la visión por computador consiste en la transformación de los datos de una imagen o fuente de vídeo en una nueva representación. Por tanto la finalidad de la visión por computador consiste en la interpretación de imágenes en formato digital por parte de un programa para conseguir algún objetivo determinado.

El campo de la visión por computador comienza cuando tecnológicamente se pueden captar y manipular las imágenes como matrices [BO01] y desde entonces no ha dejado de crecer el número de aplicaciones en las que esta disciplina está presente como en las aplicaciones de reconocimiento de sujetos, interpretación de la orientación de piezas, lectura de caracteres[2], interpretación de imágenes médicas o seguimiento de trayectorias de objetos.

Como ya se explicó en la sección 1.4, en Wili se realiza un análisis de la fuente de luz principal del mundo real para posteriormente aplicar las propiedades de iluminación a los objetos del mundo virtual. El empleo de las técnicas propias de la visión por computador ayudan a conseguir este objetivo. Empleando diversos algoritmos es posible determinar, a partir de una captura con una cámara digital, la posición donde se encuentra el foco que ilumina la escena.

Para ello se emplea *OpenCv (Open Computer Vision)* que se presenta como la biblioteca más extendida para la realización de software de visión por computador.

OpenCv fue desarrollada por Intel Research en 1999. Actualmente presenta una licencia BSD y es multiplataforma. Esta biblioteca presenta diversas interfaces para poder desarrollar aplicaciones basadas en visión por computador en varios lenguajes de programación como *AnsiC* y *C++*, *Python* y recientemente *Java*.

En las siguientes secciones se describirán las funciones y operaciones más empleadas en el análisis de imágenes mediante visión por computador para la detección de contornos, pasando después a definir el que probablemente sea el recurso por excelencia en el estudio de imágenes, el histograma.

2.3.1. Definición de contornos

Tal y como indica [BK08], "Un contorno es una lista de puntos que representa, de una forma u otra una curva en una imagen".

[2]La lectura de caracteres está muy extendida, sobretodo en sistemas de registro de matrículas de vehículos

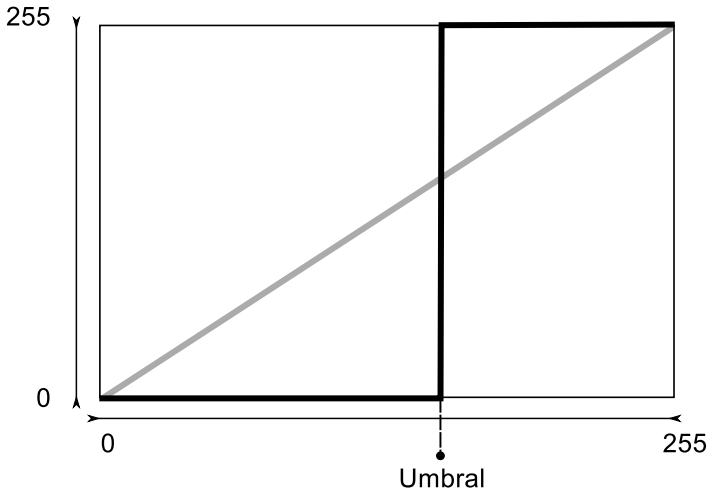

Figura 2.8: Binarización de una imagen. En este caso, la imagen está en escala de grises, ya que tiene 256 colores. Tras la binarización, los valores anteriores al umbral establecido (representado con una línea negra) tendrán el valor 0 y los valores superiores al umbral tendrán un valor de 255.

Siguiendo esta definición, un contorno consiste en una estructura de datos basada en secuencias de puntos. Esto permite definir formas en una imagen mediante objetos, obteniendo una representación lógica de las mismas. Cada punto representa una coordenada en el espacio, por lo que los contornos pueden ser empleados para realizar un estudio sobre la longitud del perímetro de las formas e incluso el área que ocupan.

Gracias a los contornos se puede obtener la posición de la forma en la imagen, hacer un recuento de formas que aparecen en la misma o establecer que formas están dentro de otras.

La biblioteca *OpenCv* soporta el trabajo con contornos a partir de una imagen binarizada.

Binarización de una imagen

Una de las principales tareas a la hora de realizar una aplicación basada en visión por computador es discernir entre la información que es relevante para el propósito de la aplicación y la información irrelevante. Una de las técnicas empleadas para conseguir una imagen limpia que contenga solo información significativa es la *binarización*.

La binarización de una imagen consiste en generar otra que presente únicamente dos colores, normalmente uno correspondiente al máximo del canal y el otro correspondiente al mínimo. De esta forma las partes de la imagen con uno de los colores representará la información importante de la misma, mientras que el resto representará la información irrelevante.

En visión por computador, normalmente se suele trabajar con imágenes en escala de grises, aunque también hay casos en los que se binariza la imagen separándola por canales. En el

caso de tratar con imágenes en escala de grises, los valores de los píxeles de la imagen pueden oscilar entre 0 y 255 si son representados con 8 bits. Esto hace, por tanto, que los dos colores por excelencia sean 0 y 255.

Para poder generar la imagen binarizada hay que determinar un *umbral* o frontera a partir del cual todo valor de la imagen por encima de ese umbral tendrá el valor de uno de los colores (normalmente el blanco) y todo valor que esté por debajo tendrá el valor del otro color (normalmente el negro), tal y como muestra la Figura 2.8.

La clave para que la binarización aporte buenos resultados reside en la elección de un umbral adecuado para la imagen a tratar. Esto, en la mayoría de los casos, requiere de un estudio exhaustivo de la imagen para lograr el valor más adecuado del umbral. En algunos casos, este cálculo puede representar una de las tareas más relevantes , condicionando incluso los resultados de etapas posteriores en el procesamiento de la imagen.

En algunos casos, después de binarizar la imagen y obtener un contorno, conviene reducir el número de puntos que lo definen, sobretodo si hay que realizar búsquedas de puntos dentro del mismo. A esta técnica de reducción de puntos se le llama *poligonización*

Poligonización de contornos

Una vez obtenido el contorno que define una forma, será interesante obtener los puntos más importantes que la definen, es decir, un número reducido de vértices mediante el cual pueda construirse con el mínimo error posible.

Para realizar esta poligonización del contorno, *OpenCv* emplea una reducción de puntos basada en el algoritmo de aproximación de Douglas-Peucker.

Los pasos seguidos por el algoritmo pueden verse en la Figura 2.9. En primer lugar, el algoritmo tiene como entrada un contorno , tal y como puede observarse en la subimagen (b) de la Figura 2.9.

Tomando dicho contorno, el algoritmo trata de buscar los dos puntos externos más distantes y los une mediante una línea, como puede apreciarse en la subimagen (c) de la Figura 2.9. Después, busca en el contorno el punto más lejano a dicha línea y lo añade a la lista de puntos que conforman la aproximación. El proceso se repite de forma iterativa hasta que todos los puntos del contorno están a una distancia menor que el parámetro de precisión.

Como se indicó al principio de esta sección, un mal cálculo del umbral a la hora de binarizar la imagen deriva en una mala aproximación mediante contornos del objeto que se quiere estudiar. Para evitar esto, es necesario realizar un estudio previo de la imagen. En este tipo de tareas, el recurso más empleado suele ser el histograma.

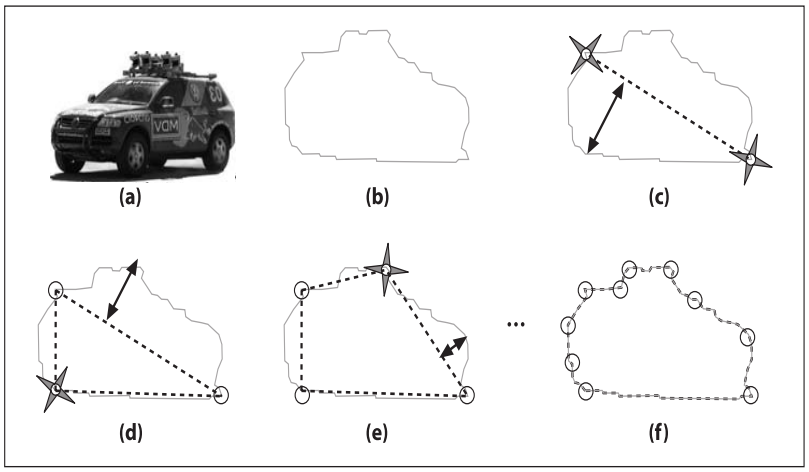

Figura 2.9: Algoritmo de Douglas-Peucker. La imagen original, (a), es aproximada por un contorno, (b). El algoritmo busca los dos puntos más distantes, (c). A continuación, busca puntos del contorno y los añade a la lista de aproximación si están a una distancia menor que la que marca el parámetro de precisión, (d). El proceso se repite de forma iterativa, (e) y (f).Imagen extraída de [BK08]

2.3.2. Gestión de histogramas

Un histograma representa una distribución de frecuencias. En el análisis de una imagen, el histograma es uno de los elementos más utilizados para determinar la exposición de la imagen, el rango de frecuencias que presenta, cuál es la frecuencia más repetida e incluso realizar operaciones sobre una imagen como puede ser el cálculo de la media de las frecuencias que presenta.

A continuación se describen algunas de las operaciones realizadas sobre histogramas.

Suma de una constante

Sea $A(x, y)$ un *píxel* determinado de la imagen A y $R(x, y)$ un *píxel* de la imagen resultante R. Se describe la suma de una imagen A por una constante a como:

$$R(x, y) = A(x, y) + a$$

Esta operación es aplicada para un único canal y produce un desplazamiento del histograma hacia la derecha, obteniendo una imagen más clara. La suma puede producir un desbordamiento que debe controlarse. Por ejemplo, en el caso de que un color se represente con 8 bits, puede darse el caso de que $R(x, y) > 255$. Cuando un *píxel* adquiere valores por encima del máximo representable o por debajo del mínimo se dice que está *saturado*. Un ejemplo de la aplicación de la suma de una constante puede verse en la Figura 2.10.

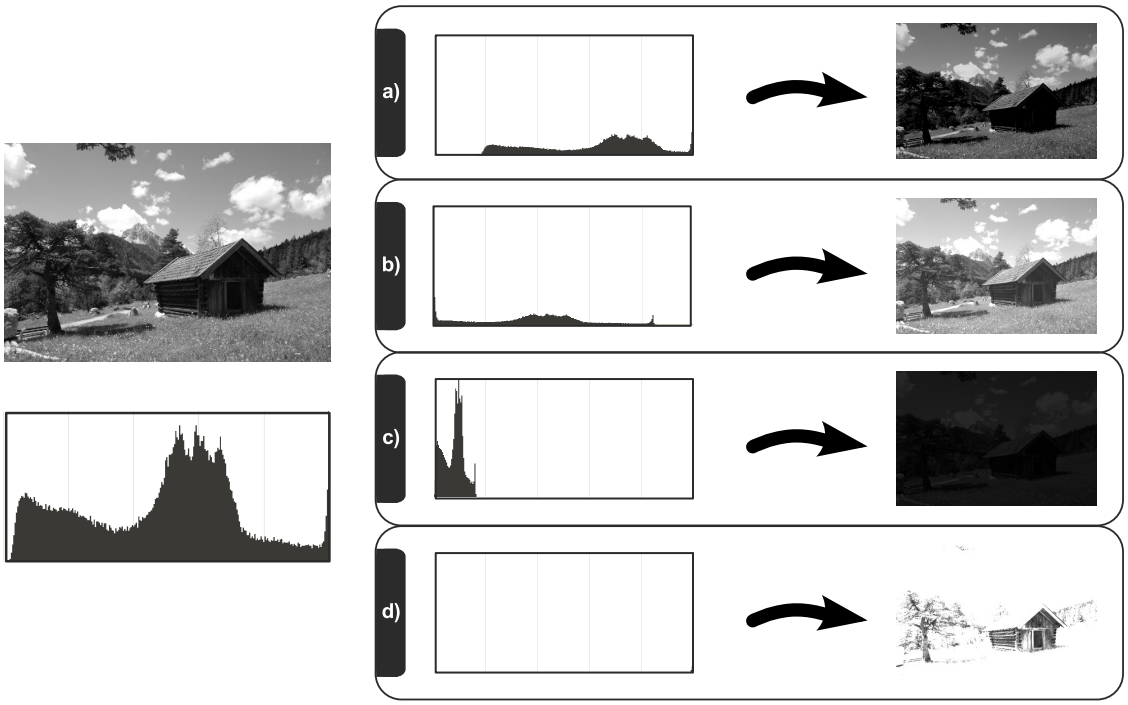

Figura 2.10: Operaciones aritméticas con histogramas.A la izquierda se observa una imagen inicial y su histograma asociado. A la derecha aparecen los resultados de aplicarle operaciones aritméticas (en todos los casos se opera con una constante $c = 40$). (a), muestra el histograma y resultado asociado de sumar una constante . (b), muestra los resultados de restar una constante. (c), es el resultado de multiplicar por una constante. Por último, (d), muestra el resultado de dividir por una constante.

Resta de una constante

Sea $A(x, y)$ un *píxel* determinado de la imagen A y $R(x, y)$ un *píxel* de la imagen resultante R. Se describe la resta de una imagen A por una constante a como:

$$R(x, y) = A(x, y) - a$$

Esta operación es aplicada para un único canal y produce un desplazamiento del histograma hacia la izquierda, obteniendo como resultado una imagen más oscura. La suma puede producir un desbordamiento que debe controlarse. Por ejemplo, en el caso de que un color se represente con 8 bits, puede darse el caso de que $R(x, y) < 0$. Un ejemplo de la aplicación de la resta de una constante puede verse en la Figura 2.10.

Multiplicación por una constante

Sea $A(x, y)$ un *píxel* determinado de la imagen A y $R(x, y)$ un *píxel* de la imagen resultante R. Se describe la multiplicación de una imagen A por una constante a como:

$$R(x, y) = A(x, y) * a$$

Esta operación transforma el histograma estirándolo hacia la derecha. Cabe destacar que, en el caso de que $a = 1$, la imagen queda inalterada. En el caso de que $a = 2$, esta operación satura la imagen a partir del valor 127 usando un canal de 256 colores . Puede verse un ejemplo de esta operación en la Figura 2.10.

División por una constante

Sea $A(x, y)$ un *píxel* determinado de la imagen A y $R(x, y)$ un *píxel* de la imagen resultante R. Se describe la división de una imagen A por una constante a como :

$$R(x, y) = \frac{A(x, y)}{a}$$

Esta operación transforma el histograma encogiéndolo. Puede verse un ejemplo de esta operación en la Figura 2.10.

Transformaciones sobre el histograma

Además de realizar operaciones aritméticas, los histogramas son utilizados para calcular y ajustar la calidad de la imagen. Esta calidad viene determinada, normalmente, por el rango de frecuencias que cubre. Un histograma que cubre todas las frecuencias corresponde a una imagen con el canal correctamente normalizado.

En ocasiones, las imágenes no presentan estas características, por lo que hay que ajustarlas siguiendo determinadas técnicas:

- *Estiramiento del histograma*. También conocido como *stretching*, este ajuste consiste en determinar el máximo y mínimo del histograma y hacerlos coincidir con el valor máximo y mínimo del canal, de forma que quede estirado cubriendo todo el rango de frecuencias. Para ello, tomando v como el valor de una determinada frecuencia en un histograma, m como la frecuencia mínima y M como la frecuencia máxima, puede definirse esta operación de estiramiento $f(v)$ como:

$$f(v) = \frac{(v - m) * 255}{(M - m)}$$

- *Ecualización del histograma*. Consiste en obtener un histograma con una distribución uniforme. Utilizando esta técnica se consigue maximizar el contraste de la imagen respetando la información estructural de la misma.

La visión por computador es empleada en la informática para el análisis de imágenes con una finalidad concreta. En *WILi* esta finalidad es obtener las coordenadas del foco principal de la escena real para poder determinar que textura integra de mejor forma al objeto virtual en el entorno. La generación de estas texturas se realiza simulando la iluminación de entornos reales en suites de gráficos 3D.

En la siguiente sección se especificará las características principales que debe tener la suite. También se realizará una comparativa para determinar que suite de gráficos 3D disponible en el mercado se ajusta más a las necesidades de proyecto.

2.4. Suites de gráficos 3D

WILi requiere de una suite de gráficos 3D para poder modelar los objetos virtuales que aparecerán en la escena 3D. Además de esto, se requiere poder generar una escena en la que se simulen distintos escenarios para poder realizar un estudio de iluminación sobre un modelo tridimensional.

Teniendo en cuenta las anteriores necesidades del sistema, se establece que la mejor alternativa disponible debe cumplir las características definidas en la siguiente sección.

2.4.1. Características

En este apartado se definirán las características que debe cumplir la suite de gráficos 3D elegida.Estas han de ser:

- *Estar basado en Software Libre*. Carecería de sentido desarrollar una aplicación libre empleando herramientas privativas. Además, se obligaría a los usuarios a tener que comprar una licencia de software privativo para poder ejecutar el software.
- *Disponer de documentación y tener soporte de una comunidad*. Las suites de gráficos 3D presentan multitud de opciones difíciles de recordar. Una buena guía de referencia puede llegar a influir incluso en el tiempo de desarrollo de la aplicación. La existencia de la comunidad permite (aunque no garantiza) que los usuarios puedan ponerse en contacto a través de foros y emails para realizar consultas y poder así resolver dudas fácilmente.
- *Empleo de pocos recursos del sistema*. La aplicación puede ejecutarse en distintas plataformas, algunas de ellas incluso con un hardware poco potente. Es por eso que el uso de los recursos del sistema debe estar muy controlado.

- *Disponer de una API para el desarrollo de scripts*. Esta característica es muy importante, ya que la interacción con la suite de modelado ha de hacerse mediante código. Los objetos que presenten deben ser accesibles mediante técnicas de *scripting*, pudiendo acceder y modificar propiedades del mismo. Ha de soportar la consulta de vértices y caras, así como poder tener accesibles diferentes acciones de edición como escalado, rotación y traslación.

- *Presentar compatibilidad entre versiones, tanto anteriores como nuevas*. Si la suite cumple esta característica, los modelos tridimensionales podrán ser siempre usados, independientemente de su la versión.

- *Ser independiente de la plataforma y del sistema operativo*. De esta forma se añade cierta flexibilidad a los requerimientos.

- *Soporte para el modelado 3D poligonal*. Para poder desplegar los objetos virtuales en la escena real, primero hay que modelarlos, por lo que esta característica ha de estar incorporada en la suite.

- *Manejo de distintos tipos de luces*. Dado que este proyecto se basa en la luz, ha de soportar distintas configuraciones de la misma, siendo básicos los focos del tipo direccional y omnidireccional.

- *Soporte de algoritmos de iluminación del tipo RayTracing*. La generación de texturas que simulen el objeto con una iluminación acorde a la realidad requiere de técnicas basadas en render realista como el trazado de rayos, que es capaz de simular los rebotes de los fotones por toda la escena de forma muy cercana a la realidad.

- *Soporte de mapeado de texturas uv*. La suite debe poder mapear texturas uv para poder desplegarla sobre el objeto cuando se dibuje en la escena.

- *Posibilidad de bakeado de texturas*. Es necesario que la suite soporte el bakeado de texturas para poder mapear la luz de determinados escenarios. Esta característica es indispensable y elimina a cualquier suite candidata que no lo soporte.

- *Posibilidad de lanzar el programa en modo background*. La intención en el uso de la suite es explotar sus funcionalidades sin tener que abrir la interfaz gráfica del mismo, por lo que será necesario que cuente con modos de ejecución en *background*.

En la actualidad existen numerosas suites de modelado 3D entre las que destacan *3ds Max* de Autodesk, *LightWave* de NewTek, *Maya* de Autodesk , *Rhino 3D* y *Blender*. A continuación se describe cada uno de ellos para terminar haciendo una comparativa entre las suites basándose en características que deben soportar.

2.4.2. 3ds Max

Desarrollado por *Autodesk*, *3ds Max* proporciona herramientas de modelado, animación, renderización y composición 3D.

Las principales características son:

- Manejo del rendimiento mediante el núcleo de gráficos.
- Soporte de texturas *uv*.
- Soporta simulaciones dinámicas de cuerpos rígidos.
- Presenta métodos de render realista.
- Posibilidad de hacer *scripting* mediante el lenguaje *MAXScript*.

La última versión disponible es la 2012, lanzada en Marzo del 2011.

2.4.3. LightWave

Se trata de una suite de gráficos 3D desarrollada por *NewTek* muy utilizada en el renderizado 3D de imágenes.

Presenta un currículum inmejorable en el mercado del cine (se ha usado en películas como *Avatar* o *Repo Men*), de la televisión (usado en series como *CSI*, *Lost* o *Star Trek*) e incluso para generar portadas de periódicos y revistas como *New York Post*, *Wall Street Journal* o *Rolling Stone*, además de usarse para desarrollar anuncios televisivos.

Las principales características que presenta son:

- Posee un motor de *rendering* que soporta el *render* realista empleando técnicas de reflexión, refracción o radiosidad, además de contar con herramientas como *Viewport Preview Renderer* (VPR), que ofrece resultados instantáneos de vistas de escenas realistas y de objetos.
- Soporta el modelado poligonal y la subdivisión de superficies.
- Soporta *Look Up Tables* (LUTs) para una iluminación realista y composición más flexible.
- Presenta simuladores físicos entre los que destacan los de cabello.
- Soporta el mapeado de texturas uv.
- Soporta el *scripting* mediante su lenguaje *LScript*.
- Permite la interacción en tiempo real con las escenas 3D.

La última versión de *LightWave* es la 10, disponible desde Diciembre de 2010.

2.4.4. Maya

Inicialmente desarrollada por *Alias Systems Corporation* y adquirida por *Autodesk* en 2005, *Maya* es una suite de gráficos 3D que permite modelar, animar, renderizar y realizar efectos gráficos tridimensionales para aplicar en juegos o vídeo. Ofrece potentes herramientas para el prototipado de videojuegos, además de simuladores físicos de partículas, cabello y fluidos entre otros.

Las principales características de *Maya* son:

- Software usado por multitud de compañías de producción de cine, videojuegos y televisión.
- Ofrece soporte al trabajo colaborativo.
- Soporta técnicas de *scripting*.
- Controla los recursos del sistema como la memoria y utiliza técnicas de paralelización en los cálculos que realiza durante su ejecución para aumentar el rendimiento.
- Interoperabilidad. Es capaz de importar y exportar los modelos entre otras aplicaciones comerciales.
- Soporte para *scripting* con el lenguaje *Maya Embedded Language (MEL)*, además de contar con una interfaz para *Python* y una API para C++. *Multiplataforma*

La versión actual de *Maya* es la 2012.

2.4.5. Rhino 3D

Desarrollado por *Robert McNeel & Associates*, *Rhino* es una herramienta empleada para crear, analizar, edita, documentar, renderizar y animar curvas NURBS superficies y sólidos. *Rhino* soporta también el modelado con mallas poligonales.

Las principales características que soporta son:

- Software empleado en el diseño, prototipado, ingeniería, análisis y fabricación de elementos a cualquier escala.
- Soporte para la organización de proyectos.
- Compatibilidad con archivos generados por otras aplicaciones CAM, de renderizado, animación o ilustración.
- Soporte con gran variedad de escáneres e impresoras 3D.

- Requerimientos hardware mínimos. No requiere de ningún hardware especial
- Plugins disponibles de terceros para mejorar las capacidades de la aplicación
- Soporte para el *scripting*

La versión actual de *Rhino* es la 4.0 de Febrero de 2007, aunque hay disponible una versión 5.0 Beta.

2.4.6. Blender

Según la definición de [RS04], "*Blender* es un programa que integra una serie de herramientas para la creación de un amplio rango de contenidos 3D, con los beneficios añadidos de ser multiplataforma y tener un tamaño de unos 5MB".

Blender fue desarrollado por la compañía 'Not a Number' (NaN) y fue liberado bajo una licencia GPL tras una impresionante campaña de recaudación de fondos de la *Blender Fundation* en la que recolectó la cantidad de 100.000 € en siete semanas para poder comprar la propiedad intelectual y de código del mismo.

Esta suite puede ser usada tanto para el modelado 3D como para la generación de animaciones, generación de contenido interactivo gracias a su motor 3D en tiempo real o incluso edición de vídeo.

En la fecha en la que se ha escrito este proyecto, *Blender* ha sufrido una importante remodelación arquitectural y de presentación pasando de la versión 2.49 a la 2.50. Uno de los principales cambios ha sido la definición de la nueva API 2.50 que permite el acceso mediante Python a gran parte de la funcionalidad de *Blender*. Por otro lado, se han realizado cambios significativos en la interfaz consiguiendo acercar al usuario a una experiencia más intuitiva.

Las principales ventajas de *Blender* son:

- Independiente de plataforma y sistema operativo: Soportado para *GNU/Linux*, *Windows* y *Mac*.
- Software libre bajo una licencia GPL gratuito.
- Acceso a las herramientas mediante *scripting* con *Python 3.0* usando la nueva API 2.50.
- Tamaño del software muy pequeño inferior a los 40 MB según el sistema operativo y la plataforma.
- Soportado por una comunidad de más de 250.000 usuarios[3].

[3]Dato del 2004 extraido de [RS04]

- Acepta formatos gráficos como *JPG*, *PNG* o *TIFF*.
- Integración externa con *ray tracers* como YafRay.
- Soporta de texturas uv.
- Soporta de *baking* de texturas.
- Tiene varios tipos de luces implementadas: *Spot*, *Sun*, *Point*, *Hemi* y *Area*.
- Implementa el modo *background* y la ejecución de *scripts* por consola.

La versión actual de Blender es la 2.59 liberada el 11 de Agosto del 2011.

2.4.7. Comparativa

En esta sección se comparan las suites descritas anteriormente en base a las características descritas de cada una de ellas.

Suite	GNU/Linux	Windows	Mac OS X	Licencia	Precio ($)
3ds Max	No	Sí	No	Propietaria	3500
LightWave	No	Sí	Sí	Propietaria	900
Maya	Sí	Sí	Sí	Propietaria	3500
Rhino3D	Sí	Sí	Sí	Propietaria	1000
Blender	Sí	Sí	Sí	GPL	0

Cuadro 2.2: Comparativa de soporte entre SSOO y Licencias entre suites de gráficos 3D.

La tabla 2.2, presenta características más cercanas a términos comerciales y de implantación de las suites como son su sistema operativo, licencia y coste. De la tabla destaca que *Blender* es la única herramienta libre de las comparadas, algo muy positivo para el Proyecto de Fin de Carrera que emplea una licencia en esos términos.

Como sistema operativo principal para el desarrollo de *WILi* se determinó uno basado en *GNU/Linux*, por lo que la suite *3ds Max* y *LightWave* no podrían emplearse al no soportar este sistema operativo. En cuanto al coste , algo que repercute directamente en el coste del proyecto, las suites presentan precios muy dispares. Como herramientas más caras se aprecian *3ds Max* y *Maya* (3500$). En tercer lugar, con un coste de 1000$ aparece *Rhino3D*. La cuarta posición la ocupa *LightWave* con un coste de 900$. En la última posición en esta clasificación por costes aparece *Blender*.

En la tabla 2.3 que atiende a características técnicas deseables, se puede observar que la mayoría de las suites estudiadas presentan herramientas de modelado, soportan distintos tipos de luces (tanto direccionales como omnidireccionales), admiten técnicas de *ray tracing* y soportan texturas *uv*. Sin embargo, se puede observar que para *Rhino*, una herramienta más

Suite	Principal uso	TR[a]	Baking	Tex. uv	BG[b]	Luces	Modelado	Scripts
3ds Max	Videojuegos, animación, iluminación, modelado 3D, rendering, efectos 3D	Sí	Sí	Sí	No	Directional, Spot, Point	Sí	Sí
LightWave	Videojuegos, animación, iluminación, modelado 3D, rendering, maquetado.	Sí	Sí	Sí	No	Sun, Spot, Point, Lineal, Area	Sí	Sí
Maya	Videojuegos, animación, iluminación, modelado 3D, rendering, maquetado, edición de video.	Sí	Sí	Sí	No	Ambient, Directional, Point, Spot, Area	Sí	Sí
Rhino3D	Modelado, CAD	Sí	No	Sí	Sí	Spot, Point, Directional, Rectangular, Linear	Sí	Sí
Blender	Videojuegos, animación, iluminación, modelado 3D, rendering, efectos 3D	Sí	Sí	Sí	Sí	Spot, Sun, Point, Hemi, Area	Sí	Sí

Cuadro 2.3: Comparativa de características soportadas entre suites de gráficos 3D.

[a]Trazado de rayos
[b]Modo Background

orientada a *CAD* y al modelado, no se observaron características de soporte a técnicas de *texture baking*, por lo que no podría emplearse esta herramienta en el desarrollo de *WILi*.

También se observa en la tabla que, a excepción de *Rhino3D*, las aplicaciones estudiadas presentan usos muy parecidos, destacando el modelado, la iluminación y el *rendering*.

Las suites analizadas soportan técnicas de *Scripting*, algo muy útil para interactuar de forma automática con la aplicación. Sin embargo, no se observó que *3ds Max*, *LightWave* o *Maya* soportasen la ejecución en modo *background*, quedando *Blender* como suite que cumple las características expuestas en la tabla 2.3.

2.4.8. Conclusión

En esta sección se han estudiado algunas de las suites para gráficos 3D más relevantes del mercado. A pesar de tener funcionalidades muy parecidas y de soportar el modelado, técnicas de *ray tracing* e incluso de *scripting*, se descartaron *Rhino3D* por no observar soporte a técnicas de *texture baking* y a *3ds Max*, *LightWave* y *Maya* por no haber encontrado soporte a la ejecución en modo *background*.

Atendiendo a otra clasificación, *3ds Max* no podría utilizarse en el proyecto al no estar soportado en un entorno *GNU/Linux*. A excepción de *Blender* que presenta una licencia GPL, el resto de suites son herramientas privativas. Una de las características que debía soportar la suite elegida en el desarrollo del proyecto era que presentase una licencia libre, ya que el presente proyecto tiene una licencia GPLv3. Por tanto, las herramientas privativas no podrían ser usadas siguiendo este principio.

Además, tomando como referencia el coste de las aplicaciones, *Blender* es la única herramienta gratuita, algo que contribuye a mejorar el presupuesto de *WILi*.

Como suite para gráficos 3D multiplataforma, con el presupuesto que más se ajusta a *WILi*, basado el Software Libre, y que cumple con las características estudiadas, se puede concluir que *Blender* es la mejor opción de todas, siendo elegida para el desarrollo de este Proyecto de Fin de Carrera.

CAPÍTULO 3

OBJETIVOS

EL objetivo general del presente Proyecto de Fin de Carrera puede definirse como la creación de un sistema que facilite la detección de la iluminación principal en el mundo real para la adaptación y despliegue de modelos en el ámbito de la Realidad Aumentada.

En base a este objetivo general se detallan una serie de subobjetivos funcionales específicos:

1. **Síntesis de imagen realista**. El sistema utilizará algoritmos de síntesis de imagen realista para el cálculo de la iluminación y proyección de sombras.

2. **Despliegue interactivo en tiempo real**. Para cumplir con los requisitos de interactividad de cualquier aplicación de Realidad Aumentada, el módulo de representación deberá permitir el despliegue de los objetos virtuales en tiempo real (al menos 25 *frames* por segundo).

3. **Sistema extensible**. Independiente del método de síntesis realista empleado y del modelo que va a ser iluminado. El sistema modular permitirá incorporar nuevos métodos de síntesis de imagen realista o nuevos modelos sin cambios en el código fuente.

4. **Adaptativo**. El sistema permitirá variar la resolución de texturas y el nivel de detalle de los modelos 3D para su utilización en diferentes plataformas.

5. **Basado en componentes de bajo coste**. El sistema utilizará para la detección de la iluminación de la escena marcas visuales monocromas y una cámara de visión de bajo coste.

6. **Independiente del dominio de aplicación**. El diseño del sistema será independiente del dominio de aplicación, por lo que podrá ser utilizado en cualquier aplicación de realidad aumentada (independientemente del módulo de *tracking* y de la aplicación final desarrollada).

7. **Estándares multiplataforma**. El desarrollo se basará en el uso de estándares y Software Libre para facilitar su expansión y posterior adaptación a diferentes plataformas software.

CAPÍTULO 4

MÉTODO DE TRABAJO

EN este capítulo se describe la metodología empleada para desarrollar *WILi*. La elección de la misma ha estado condicionada por los conocimientos específicos necesarios para abordarlo (informática gráfica, visión por computador, *Blender scripting*, modelado 3D, álgebra lineal), así como los requisitos y objetivos planteados en un primer momento.

Los conocimientos específicos han hecho que se requiera de un modelo que permita la revisión de las etapas anteriores, ya que, en la medida en la que estos conocimientos se van asentando en el desarrollador, es posible que se descubran partes mal diseñadas e incluso implementadas o simplemente se descubran nuevas formas de abordar el problema más eficientes.

A continuación se describe la metodología empleada, así como todos los medios hardware y software utilizados para desarrollar *WILi*.

4.1. Metodología de trabajo

La metodología empleada para el desarrollo de *WILi* corresponde al prototipado evolutivo (ver Figura 4.1). Como su nombre indica, esta metodología sugiere la construcción de un producto a partir de sucesivas versiones que cubran los requisitos conocidos. En el modelo evolutivo se asume que no todos los requisitos son conocidos. De esta forma se construyen

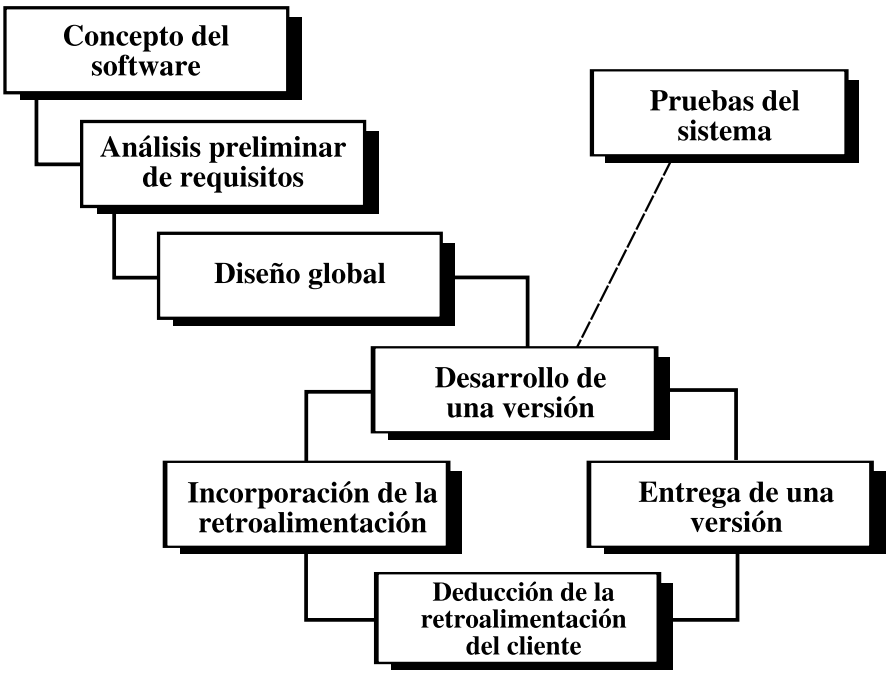

Figura 4.1: Metodología basada en prototipado evolutivo.

prototipos que van a ir mejorando conforme se añadan nuevos requisitos y se descubran las nuevas necesidades del sistema.

Una vez desarrollado un prototipo, el cliente lo prueba y ofrece una retroalimentación en la que los requerimientos del producto son actualizados. Tras esta nueva especificación, se vuelve a desarrollar otro prototipo que compartirá algunos aspectos con el anterior.

Como principal ventaja de este modelo destaca la capacidad de cubrir nuevos requisitos desconocidos. Dada la naturaleza de *WILi*, que abarca áreas muy específicas de la Ingeniería Informática, es muy probable que los requisitos cambien con el tiempo. El desarrollador adquiere un mayor conocimiento de las materias tratadas por *WILi* conforme va madurando el proyecto, siendo necesario revisar los requerimientos y la implementación del mismo.

En este caso el director del proyecto ejercerá el rol de cliente y será el que determine los nuevos requerimientos entre iteraciones. La creación de prototipos que evolucionan con el tiempo constituyen un buen indicador de la situación del proyecto en cada momento.

4.2. Herramientas

A continuación se describen los recursos, tanto hardware como software empleados en el desarrollo de *WILi*.

4.2.1. Lenguajes de programación

En el desarrollo de *WILi* se han empleado los siguientes lenguajes de programación:

- **C++:** es el lenguaje empleado para el desarrollo del subsistema de análisis de la iluminación.
- **Python:** se trata de un lenguaje muy usado para el prototipado y el *scripting*. En concreto fue empleado para desarrollar los scripts del módulo de precálculo de texturas, ya que la API de Blender 2.5 es accesible desde *Python*. La versión empleada es la 3.0.

4.2.2. Hardware

Todo el desarrollo de *WILi* se ha realizado sombre un ordenador portátil *Phoenix T7250* con un procesador *Intel(R) Core(TM)2 Duo 2.00GHz* , 2GB de memoria RAM Robson y una tarjeta gráfica *Nvidia GeForce 8600M GT* de 512MB.

Como dispositivo de captura de vídeo se ha empleado una cámara *Logitech Sphere AF*.

4.2.3. Software

A continuación se muestra el sistema operativo empleado en el desarrollo de *WILi*, las herramientas tanto de desarrollo como de documentación y las bibliotecas empleadas.

Sistema operativo

Como apuesta por el *Software Libre*, el sistema operativo empleado ha sido *Ubuntu 10.04 LTS* , conocido como la versión *Lucid Lynx*. La elección dentro del amplio campo de versiones *GNU/Linux* de *Ubuntu* se debe a la facilidad de uso e instalación de componentes que presenta, así como la gran comunidad que la sostiene.

Control de versiones

Para el control de versiones se ha empleado *Subversion*. El repositorio es público y se encuentra alojado en `https://forja.rediris.es/projects/cusl5-wili` correspondiente a la *Forja de la Comunidad del Centro de Excelencia de Software Libre de Castilla-La Mancha*.

Software de desarrollo

El software de desarrollo ha sido *Emacs*, que permite al usuario aumentar su rendimiento con la gran cantidad de atajos por teclado. Su uso en lugar de otras plataformas extendidas como *Eclipse* responde a que este automatiza muchas acciones como el compilado o la generación de makefiles, algo que quizás aumenta la productividad por abstraer dichas acciones, pero que sin duda contribuye al olvido de como se realizan estos pasos necesarios

para generar software.

Otras herramientas empleadas son:

- **Gcc:** para compilar los ficheros en C++ se ha empleado *g++* , en su versión 4.4.3.
- **Make:** usado para lanzar los archivos *makefile* de compilación, en su versión 3.81.
- **GDB:** para depurar código en entornos *GNU/Linux*, en su versión 7.1.

Documentación y gráficos

Para documentar el Proyecto de Fin de Carrera se ha empleado LaTeX, un sistema para la preparación de documentos electrónicos de alta calidad. Se ha usado la versión 2009-7.

Otras herramientas empleadas en la generación de la documentación han sido:

- **Gimp:** una herramienta de edición fotográfica empleada para realizar retoques en algunas de las imágenes de la documentación. Se ha utilizado la versión 2.6.8.
- **InkScape:** es la aplicación por excelencia para la generación de gráficos vectoriales en entornos *GNU/Linux*. Todas las imágenes vectoriales del documento han sido desarrolladas con esta herramienta. La versión empleada es la 0.47.
- **Blender:** se trata probablemente de la mejor suite de modelado 3D libre. Se ha usado tanto para crear modelos tridimensionales del proyecto como para generar imágenes del propio documento. La versión usada es la 2.56a.

Bibliotecas

- ***OpenGL*:** es una potente interfaz para producir imágenes y aplicaciones interactivas de altísima calidad usando objetos 2D y 3D. Se ha usado la versión 7.7.1.
- **SDL:** es una biblioteca para el desarrollo de aplicaciones multimedia . Se ha utilizado la versión 1.2.14.
- **DevIl:** anteriormente conocida como *OpenIl*, se trata de una biblioteca multiplataforma de carga y manipulación de imágenes. La versión usada es la 1.7.8.
- **ARToolKit:** biblioteca para el registro y tracking de marcas visuales para aplicaciones de Realidad Aumentada. Permite obtener la matriz de transformación neta de la cámara de 6 grados de libertad. La versión utilizada es la 2.72.1.
- **OpenCv:** biblioteca para el desarrollo de aplicaciones de visión por computador. Se ha utilizado la versión 2.0.
- **Bpy:** biblioteca de *Blender* que permite el acceso a gran parte de su funcionalidad.

Una vez definida la metodología y las herramientas empleadas se continuará con el capítulo que define la arquitectura del sistema.

CAPÍTULO 5

ARQUITECTURA

En este capítulo se analiza la arquitectura modular de *WILi*. Para ello se hará una descripción que comenzará con una visión más abstracta de la arquitectura para continuar aumentando el nivel de detalle progresivamente.

WILi desde su diseño y, dado que tiene que tratar diferentes problemas, fue concebido como un bloque con tres subsistemas principales diferenciados, tal y como se muestra en la figura 5.1:

- *Subsistema de precálculo de iluminación*: encargado de generar las texturas con el resultado de aplicar el modelo de iluminación realista y las sombras proyectadas adecuadamente.
- *Subsistema de análisis*: cuya función es determinar, a partir de una sombra arrojada, por un elemento conocido asociado a una marca, la localización del foco de luz.
- *Subsistema de representación*: responsable de dibujar tanto objetos 2D como 3D.

5.1. Descripción general

Como se indicó anteriormente, *WILi* presenta una arquitectura modular dividida en tres subsistemas. Dichos subsistemas son totalmente independientes entre sí. Esto significa que

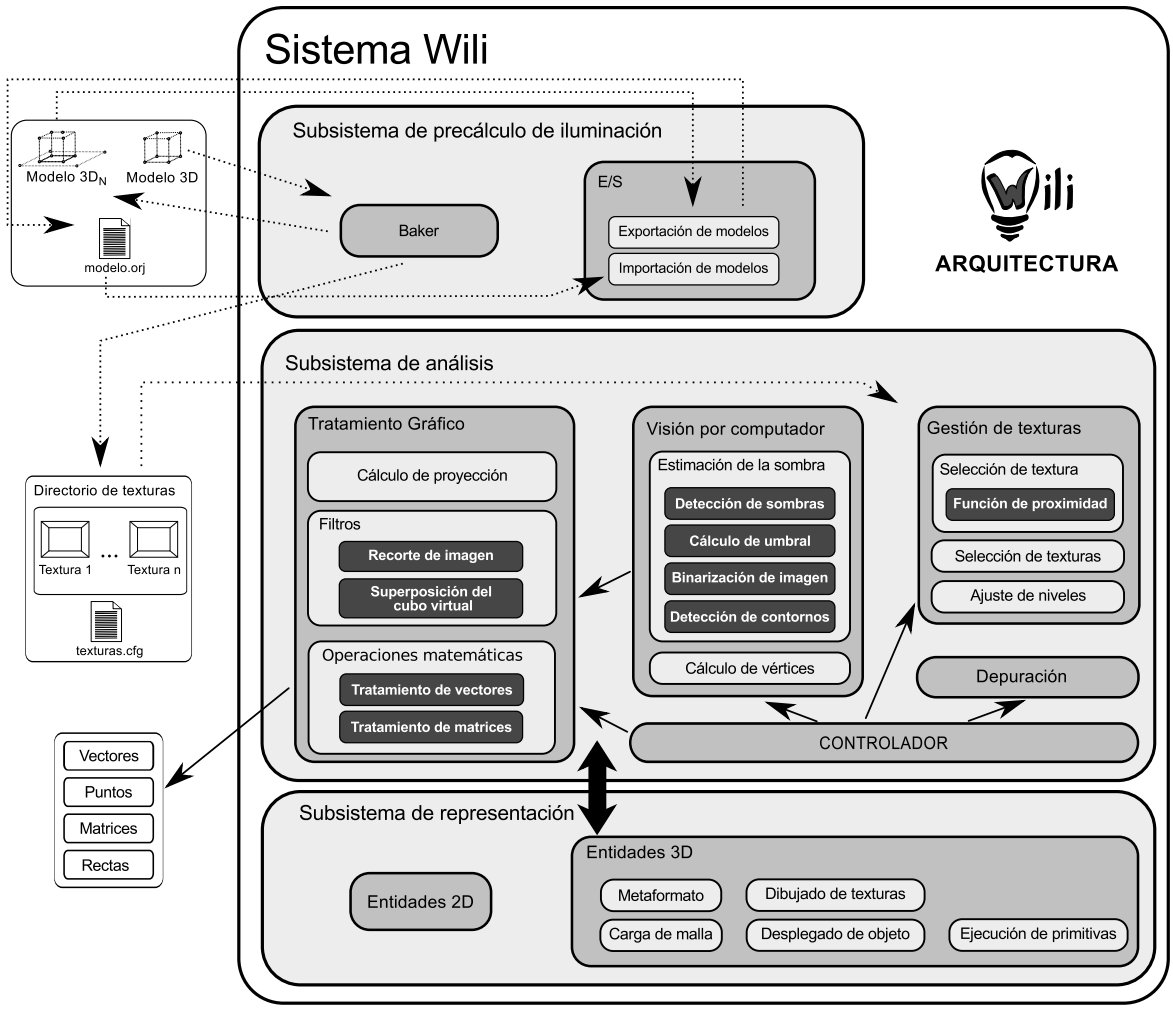

Figura 5.1: Esquema modular de *WILi*.

si se quiere implementar la generación de texturas con iluminación precalculada mediante otra biblioteca, el subsistema de análisis las reconocerá como entrada siempre que sigan el convenio establecido (que estén almacenadas en el directorio correspondiente y que el archivo de configuración se construya siguiendo la sintaxis propia especificada en la sección 5.2).

Para ofrecer una interfaz accesible desde el sistema, se empleó el patrón *Singleton*, desarrollando diferentes controladores en función de los módulos que debían gestionar y para los que tenían que ofrecer distintos servicios.

A continuación se describirá en detalle cada uno de los diferentes módulos que conforman los subsistemas anteriormente citados para obtener una visión más concreta de la arquitectura diseñada para *WILi*.

5.2. Subsistema de precálculo de iluminación

Como se indicó en la sección 2.2.1, debido a las fuertes restricciones en tiempos de respuesta y al alto coste computacional que supone realizar el cálculo de la iluminación de una escena, se optó por trabajar en el precálculo de la iluminación de una escena empleando texturas. Además, es necesario contar con un metaformato que pueda definir de forma independiente un objeto tridimensional para poder ser desplegado durante la ejecución de la aplicación de Realidad Aumentada.

Para ello este subsistema presenta dos módulos. El primero de ellos es probablemente el más importante, el módulo *Baker* (que será descrito en la sección 5.2.1). La labor de este módulo es generar texturas precalculadas de la iluminación que un objeto ante determinadas condiciones lumínicas, además de un fichero utilizado para comunicar este subsistema con el resto de *WILi*. Para que un modelo pueda ser interpretado por el resto de subsistemas, se emplea el exportador de *Orej*, un metaformato descrito en la sección 5.2.2 y que corresponde al módulo de *entrada/salida*.

5.2.1. Módulo Baker

Como se comentó en la sección 2.2.1, para conseguir una integración efectiva de los objetos virtuales en la escena real es necesario emplear métodos de síntesis realista de imagen. Estos métodos tienen como principal inconveniente el tiempo de cómputo necesario para calcular la interacción de la luz con las superficies. Además, atendiendo a las características de Azuma [A$^+$97], la representación de los objetos virtuales debe realizarse en tiempo real, por lo que no es posible con los medios hardware actuales realizar estos cálculos en tiempo de ejecución.

La solución empleada por *WILi* se integra en el módulo *Baker* que se encarga de obtener una representación de la iluminación y sombras arrojadas por el objeto y proyectarlas sobre un mapa de texturas.

Para que un objeto virtual tridimensional tenga una correcta integración dentro de la escena real, éste necesita tener las siguientes propiedades:

1. Estar iluminado en consonancia con la escena real.

2. Proyectar sombra sobre el mundo.

Para que el objeto esté iluminado en consonancia con la escena real, el módulo *Baker* trabaja en un entorno virtual controlado. Dicho entorno consiste en una escena en la que se genera una esfera encargada de englobar al modelo tridimensional, además de controlar toda la iluminación y posición de los objetos de la misma.

En las coordenadas asociadas a los vértices de esta esfera se posicionará un foco de luz, de forma que simule la iluminación del objeto cuando recibe la luz de un foco desde esas coordenadas. En una esfera modelada, el número de vértices varía en función del número de anillos y segmentos que tenga, por lo que este planteamiento permite variar la granularidad con la que se va a realizarse el precálculo de la iluminación . Cabe destacar que los vértices útiles en este cálculo serán los que correspondan a la semiesfera superior[1].

La segunda característica, arrojar sombra, es inherente a un objeto iluminado. Esta propiedad es la que permite, en una imagen plana, percibir sin ambigüedades la posición y tamaño de un objeto (ver sección 2.2.3).

Para su simulación, este módulo genera automáticamente un plano en la base del objeto que se desea analizar para poder captar la sombra proyectada por dicho modelo.

El módulo *Baker* toma como entrada un modelo tridimensional. El único requisito que debe cumplir es que tenga el origen de coordenadas local situado en su base. Antes de comenzar el proceso, el objeto virtual debe ser normalizado.

Normalización del modelo virtual

En la versión actual del prototipo desarrollado, las medidas de los objetos de la escena simulada son estáticas, es decir, una vez definidas antes de usar este módulo no varían su valor durante la ejecución. Para poder garantizar que la esfera englobe el modelo de forma correcta, así como asegurar que el plano captará las sombras que arroje, hay que establecer un tamaño normalizado para el modelo.

En el caso del módulo *Baker*, la normalización del modelo consiste en inscribirlo dentro de un cubo unitario, ajustando el tamaño del objeto proporcionalmente.

Este módulo utiliza un convenio de nombres preestablecidos para su control interno. En determinados momentos de la ejecución del módulo *Baker* pueden coexistir en la escena virtual hasta cuatro tipos distintos de objetos: una esfera, un foco, un plano y un modelo. Para realizar acciones sobre ellos y acceder a sus propiedades, les atribuye nombres predefinidos. Además, este criterio de nombres preestablecidos permite al exportador de *Orej* transformar los modelos existentes en una escena simplemente especificando directamente sus nombres.

Flujo de trabajo

A continuación se describen los pasos seguidos por el módulo *Baker* para realizar el estudio de la iluminación del objeto ante determinados escenarios:

[1] Se entiende por semiesfera superior a la semiesfera que contiene los vértices con coordenadas en el eje z positivas según la disposición del modelo normalizado

1. **Normalización del modelo**. Como se indicó anteriormente, el modelo es escalado a medidas unitarias. También es definido con un nombre preestablecido para poder identificarlo fácilmente dentro de la escena.

2. **Posicionamiento del objeto en el origen de coordenadas**. El módulo no presenta como requisito de entrada que el modelo esté centrado en el origen de coordenadas. Sin embargo, los objetos controlados en la escena siguen como referencia el origen las coordenadas (0.0, 0.0, 0.0). El modelo tiene que presentar también una posición definida y estudiada previamente. Por ello, la primera acción que realiza el módulo es centrar el modelo en el origen de coordenadas de la escena.

3. **Creación del entorno virtual**. En esta etapa se generan los objetos virtuales que van a intervenir en el estudio de la simulación de la iluminación. Estos elementos pueden clasificarse como dinámicos o estáticos:

   - *Dinámicos*: Como único elemento dinámico aparece el punto de luz. Cada punto de luz es insertado en la escena de forma unitaria en la coordenada de alguno de los vértices de la semiesfera anteriormente definida. Una vez terminada la simulación de la iluminación para este escenario, este foco es eliminado, siendo reemplazado por otro foco distinto en otro vértice de la semiesfera para obtener otro distinto.
   - *Estáticos*: Los objetos estáticos de la escena virtual corresponden a la esfera, elemento empleado para obtener las coordenadas de sus vértices y al plano, que recoge las sombras arrojadas por el modelo.
     - *Esfera*: La esfera virtual está definida por un número de anillos, un número de segmentos y un radio. Dichas dimensiones son configurables antes de la ejecución del módulo. Cabe mencionar que cuantos más vértices tenga la esfera, más precisión habrá a la hora de establecer la iluminación del objeto.
     - *Plano*: Es el encargado de recoger las sombras arrojadas por el modelo. Las dimensiones del plano también son configurables y debe tener un tamaño que garantice, en la medida de lo posible, que las sombras sean proyectadas sobre el mismo. Una mala elección de las medidas del plano puede hacer que no se recojan bien las sombras del objeto, tal y como se aprecia en la figura 5.2. Si el plano es demasiado pequeño, puede que no se recoja la totalidad de la sombra, generando así un mala integración del objeto virtual en la escena real. El precálculo de la sombra debe realizarse de tal manera que únicamente se obtengan las sombras en la textura asociada al plano.

4. **Precálculo de la iluminación**. En esta etapa para cada vértice de la semiesfera superior se inserta un punto de luz en las mismas coordenadas con unas propiedades

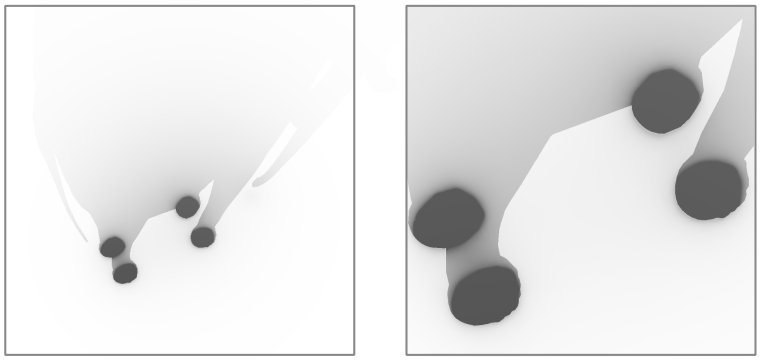

Figura 5.2: Consecuencias en la elección del tamaño del plano. A la izquierda se observa una buena elección, ya que la totalidad de la sombra puede ser recogida en el plano. A la derecha se observa como el tamaño del plano condiciona la correcta captación de la sombra.

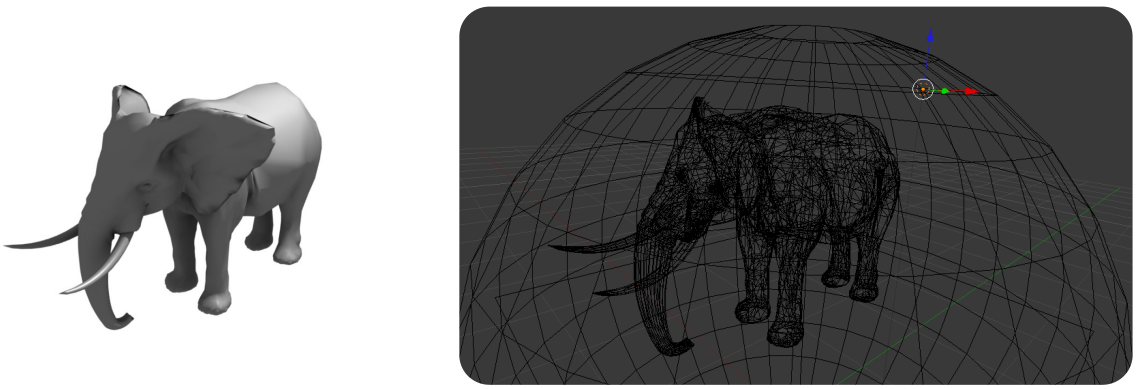

Figura 5.3: Proceso de precálculo de texturas. A la izquierda se muestra el modelo tridimensional con el que trabajará el módulo *Baker*. A la derecha se muestra una captura de este procesamiento del modelo.

lumínicas previamente establecidas. Una vez insertado el foco, se realiza el precálculo de la iluminación del modelo en ese escenario, mapeando la iluminación en texturas siguiendo la técnica *texture baking* descrita en la sección 2.2.1. Después de obtener tanto la textura del modelo que representa su iluminación como la del plano que define la sombra que proyecta el modelo, se elimina el foco de luz. Este paso se repite hasta cubrir todos los vértices de la semiesfera superior. Una captura de esta etapa puede verse en la figura 5.3.

5. **Generación del archivo de configuración**. La finalidad de este archivo es poder establecer una comunicación con el resto de módulos, indicando las texturas que ha calculado.

Las salidas de este módulo consisten en un conjunto de pares formados por la textura del modelo y su correspondiente sombra que serán almacenados en un directorio previamente

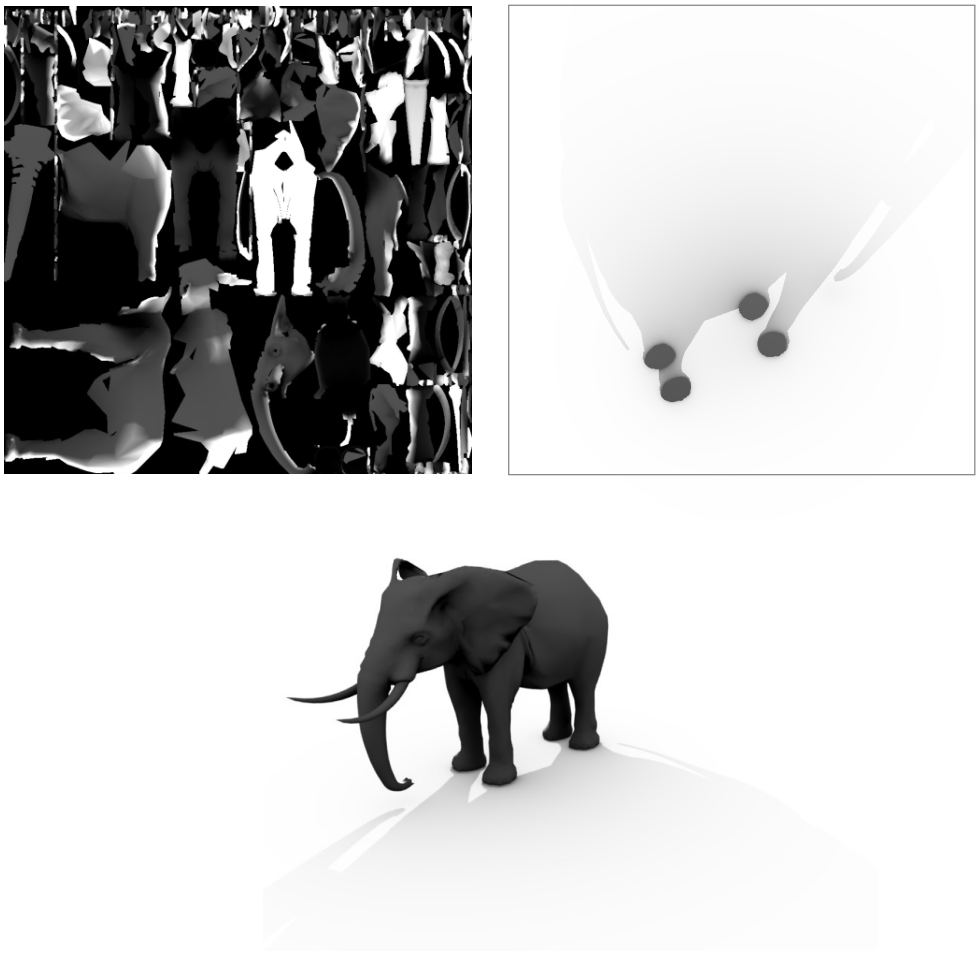

Figura 5.4: Texturas generadas por el módulo *Baker*. A la izquierda se puede observar la textura generada para el modelo virtual. A la derecha, aparece la textura correspondiente a la sombra arrojada por dicho modelo. En la zona central se muestra el modelo con la iluminación que determina las texturas anteriores.

definido, además de un archivo de configuración donde se determinarán las texturas que ha generado. En la figura 5.4 pueden contemplarse las dos texturas de salida para un foco determinado. El módulo también genera un archivo de *Blender* (.blend) para ser usado en una posible depuración.

Para la implementación de este módulo se generó la clase `Baker`. Esta clase es la encargada de recibir el modelo tridimensional y realizar los pasos especificados anteriormente para obtener las texturas y el archivo de configuración.

Para el desarrollo de esta clase se ha empleado la nueva biblioteca *bpy* que, mediante la API 2.50 de *Blender* ofrece gran cantidad de instrucciones para trabajar con *Blender* empleando *Python 3.0*.

Una de las características de esta API es que se puede acceder a las propiedades de los objetos de la escena para consultar o modificar sus valores relacionados con los materiales,

texturas, tamaños, posición, etc . También permite el acceso a propiedades de la escena como el color del horizonte o la intensidad de la iluminación global. Además es posible la ejecución de acciones sobre los objetos como seleccionarlos, entrar en modo de edición, insertarlos, borrarlos e incluso acceder y modificar sus estructuras. De igual modo, el inicio del proceso de renderizado o *baking* de texturas puede realizarse en segundo plano sin necesidad de interacción por parte del usuario.

A continuación se entrará en detalle para explicar cómo ha sido implementada la clase `Baker` siguiendo la descripción de los pasos que realiza el módulo para generar sus salidas.

En primer lugar, la normalización del modelo se hace atendiendo a su tamaño, reduciéndolo de forma proporcional hasta quedar inscrito en un cubo unitario. El método encargado de realizar esta tarea es `normalize()` de la clase `Baker`.

```
1  def normalize(obj_name):
2      obj = getObject(obj_name)
3      max_dim =max(obj.dimensions)
4      obj.dimensions = [i/max_dim for i in obj.dimensions]
```

Este método busca la coordenada más alejada del objeto del origen de coordenadas global y establece en qué medida hay que escalar el objeto para que ese vértice esté dentro del cubo unitario. Garantizando que el vértice más alejado quede inscrito se consigue que el resto lo esté también.

Otro detalle de implementación interesante es determinar las propiedades del plano para que únicamente recoja sombras. Para ello hay que asignarle al plano un material. Esta acción es realizada por el método `setNewMaterial()`. A continuación hay que alterar su propiedad `use_only_shadow` para que solo capture sombras mediante el método `onlyShadowMaterial()`.

```
1  def setNewMaterial(obj_name, material_name):
2      ops.material.new()
3
4      #Get the last created material and set its the name
5      new_material = data.materials[len(data.materials) - 1]
6      new_material.name = material_name
7      new_material.specular_intensity = 0
8
9      data.meshes[obj_name].materials.append(new_material)
10
11 def onlyShadowMaterial(material_name):
12     data.materials[material_name].use_only_shadow = True
```

Para la inserción de la esfera se empleó una capa distinta, de forma que no influyera en la iluminación de la escena.

El proceso de cálculo de la textura sigue unas etapas definidas de la siguiente forma:

1. En primer lugar, se construyen los escenarios añadiendo un foco en la coordenada determinada por un vértice de la semiesfera superior. El tipo de foco empleado puede ser configurado previamente, pudiendo elegir entre los distintos focos de *Blender*:

   - *Lamp*: Un tipo de lampara omnidireccional similar a como actúa una bombilla.
   - *Spot*: Un punto de luz direccional.
   - *Area*: Una fuente de luz para simular áreas que emiten luz como ventanas o pantallas de televisores.
   - *Hemi*: Simula una fuente de luz muy lejana y extensa, usada para simular, por ejemplo, el cielo.
   - *Sun*: Simula una fuente de luz puntual muy lejana, como el Sol.

2. A continuación se determina el nombre de las texturas. Para ello, el módulo *Baker* emplea un convenio en el nombre de la textura basado en la siguiente expresión:

   OBJECT_TYPE-f1 f2 f3

   , donde *OBJECT_TYPE* puede tomar los valores "*o*", si se trata de la textura correspondiente al objeto o "*p*" si la textura corresponde al plano. La variable *OBJECT_TYPE* va seguida de un guión, continuando con una separación entre ellas de un espacio en blanco. Después le siguen las variables *f1*, *f2* y *f3* que corresponden a la coordenada *x*, *y* y *z* del foco respectivamente con una precisión de tres decimales.

   El formato elegido se debe a una restricción que presenta *Blender* por la que el nombre de la imagen no puede superar los 20 caracteres. Cada coordenada en un eje puede ser representada en este caso con 6 caracteres (el signo menos si es negativa, la parte entera, el punto y los tres decimales), a excepción de la coordenada z, ya que los vértices corresponden a la semiesfera superior (con coordenadas en el eje z positivas). Por tanto el formato limita bastante la elección de una sintaxis más adecuada.

3. Una vez establecidos los nombres de las texturas se indica el modo de *rendering* de la textura. En el caso del objeto se ha optado por un modo de renderizado sin canal *alpha*, al contrario que en el renderizado del plano, que debe soportar la transparencia para captar únicamente la sombra.

4. Después de esto, se realiza el *baking* de la textura. El método `bake()` de la clase `Baker` es el encargado de realizar esta tarea. Este método mapea los vértices del modelo sobre la textura para obtener las coordenadas *uv* mediante el método `uvMapping()` de la clase `Baker` que emplea acciones de *Blender* como `smart_project()` o `unwrap()`. Después es necesario generar una imagen nueva para que recoja los resultados del precálculo de la iluminación. A continuación se hace el baking de la textura

mediante la acción `bake_image()` de *Blender*. Una vez calculada, se guarda en el directorio predefinido.

```
def bake(obj_name, image_name, path, x_output_size, y_output_size):

    uvMapping(obj_name)
    assignImage(obj_name, image_name, x_output_size, y_output_size)

    activeObject(obj_name)
    ops.object.bake_image()

    ops.image.pack(as_png=True)
    saveImage(image_name, path)
```

5. Por último, se elimina la fuente de luz. Los pasos se repiten hasta haber utilizado las coordenadas de todos los vértices de la semiesfera.

```
def baking():

    data.scenes[0].layers[0] = True
    data.scenes[0].layers[1] = False

    for vertice in data.meshes[SPHERE_NAME].vertices:
        if vertice.co[2] > 0.0 :
          insertSpot(vertice.co, LAMP_TYPE)

          common_name = "%.3f %.3f %.3f" % (round(vertice.co.x, 3), round(vertice.co.y, 3),
                                            round(vertice.co.z,3))

          image_object_name = OBJECT_SUBNAME + common_name
          image_plane_name = PLANE_SUBNAME + common_name # + str(cont) + ".png"

          data.scenes[0].render.color_mode = 'RGB'
          bake(OBJECT_NAME, image_object_name, output_textures_path, TEXTURE_XY, TEXTURE_XY
              )

          data.scenes[0].render.color_mode = 'RGBA'
          bake(PLANE_NAME, image_plane_name, output_textures_path, TEXTURE_XY, TEXTURE_XY)

          print("[Baker]::Bake #" + image_object_name + "# Done.")
          deleteObject(LAMP_TYPE[0] + LAMP_TYPE[1:].lower())
```

El fichero de configuración especifica las texturas disponibles siguiendo una estructura diseñada para facilitar su recuperación en base a la posición relativa en el eje z del foco asociado.

Para cada coordenada z de la semiesfera, se presenta el carácter z y dicha coordenada. A continuación se añaden las coordenadas de la semiesfera que presentan ese valor en z, anteponiéndoles el caracter "*f*" y un espacio en blanco.

Un ejemplo de este fichero de configuración se puede ver a continuación:

```
z 8.000
f 0.000 0.000 8.000

z 7.727
f 2.071 0.000 7.727
f -0.000 2.071 7.727
f -2.071 -0.000 7.727
f 0.000 -2.071 7.727
```

La elección de este sistema se debe a una optimización en el proceso de búsqueda de la mejor textura para un foco encontrado (ver sección 5.3.3).

Gracias al uso del módulo *Baker*, *WILi* puede utilizar cualquier algoritmo de síntesis de imagen realista (*RayTracing*, *PathTracing*, *Radiosidad*, *Ambient Occlusion*, ...). El precálculo de la interacción de la luz se realiza una única vez y el modelo de salida puede ser utilizado incluso en sistemas con baja capacidad de cómputo.

El diseño en capas permite, además, utilizar *render* en múltiples pasadas y combinar mediante un submódulo independiente la salida de diferentes métodos de síntesis. Así se podrían obtener sombras difusas mediante una técnica de *rendering* rápida (como *Ambient Oclussion*), definiendo las sombras mediante un método más preciso (*Raytracing*) y componiendo ambos resultados en una etapa posterior sin necesidad de realizar ningún cambio en el código del resto de módulos del sistema.

Una vez realizada la simulación de la iluminación, es necesario habilitar algún mecanismo para que pueda ser utilizado en las siguientes etapas por el resto de módulos de *WILi*. En la siguiente sección se describe el módulo de entrada/salida, encargado de transformar los modelos empleando un metaformato establecido que será empleado como mecanismo intermedio para compartir la información sobre la geometría y las texturas de la escena.

5.2.2. Módulo de entrada/salida

Las aplicaciones de Realidad Aumentada se caracterizan por desplegar objetos virtuales sobre la escena real. Para conseguir obtener objetos con el mayor realismo posible, una de las opciones es modelarlos utilizando alguna suite de gráficos 3D. El problema se presenta cuando se quiere portar dicho objeto modelado para desplegarlo en una aplicación de Realidad Aumentada. Para poder llevar cabo este despliegue, hay cantidad de factores a tener como son su color, textura, tamaño, posición, coordenadas de los vértices uv, las caras que presenta, los vértices que componen cada cara y las coordenadas de todos los vértices que conforman su esqueleto. Por tanto es necesario contar con un metaformato que exporte todas estas características y que sea capaz de recuperarlo después y desplegarlo de forma satisfactoria.

El módulo de *entrada/salida* se encarga de dar solución a este problema. Para ello se optó por emplear *Orej*. *Orej* es un metaformato definido en el Laboratorio de Investigación *Oreto* de la Universidad de Castilla-La Mancha como extensión del formato estándar *Obj* definido por Alias/Wavefront. Dicho metaformato es capaz de almacenar de forma independiente al lenguaje todas las características citadas anteriormente para realizar posteriormente el despliegue del modelo.

Mediante el exportador se genera un fichero que recoge las propiedades del modelo que recibe como entrada. Utilizando el importador se analiza dicho fichero y se cargan en memoria las estructuras de datos necesarias para su posterior despliegue.

Sin embargo, el exportador e importador de *Orej* no cubrían las necesidades de *WILi*, por lo que se optó por mejorarlos. En la tabla 5.1 puede verse una comparativa entre las características del *Orej* original y el nuevo.

Características	**Orej**	**Orej-*WILi***
Poligonos soportados		
Formatos de imagen	.ppm	.bmp, .cut, .dds, .doom, .exr, .hdr, .gif, .ico, .jp2, .jpg,.lbm, .mdl, .mng, .pal, .pbm, .pcd, .pcx, .pgm, .pic, .png, .ppm, .psd, .psp, .raw, .sgi, .tga, .tif
Modo de despliegue	WireFrame, Solid	WireFrame, Solid, White
Reserva de memoria	Estática	Dinámica
Lenguaje	C	C++

Cuadro 5.1: Mejoras y ampliaciones realizadas al importador del formato Orej

A continuación se describen estas mejoras:

- *Soporte de polígonos cuadrangulares*: el importador de *Orej* soportaba únicamente mallas triangulares, mientras que el importador mejorado reconoce tanto mallas triangulares como cuadrangulares. Esto permite que el modelo no tenga que modificar su malla a una formada únicamente por triángulos.

- *Soporte de multitud de formatos de imagen*: ésta es una de las mejoras más importantes que pueden apreciarse en el importador. El anterior formato presenta solo un tipo de textura, *.ppm*. Este formato, además de ser tremendamente ineficiente y altamente redundante (según su definición en la descripción del *man*), no soporta transparencias ni formatos de compresión. Para poder soportar la transparencia (*.png*) y otros formatos con compresión como puede ser *.jpg*, se optó por emplear la biblioteca *DevIl*.

  Empleando esta biblioteca se aumenta el abanico de posibilidades a la hora de importar una textura, soportando, además de *.ppm* los siguientes formatos: *.bmp, .cut, .dds,*

.doom, .exr, .hdr, .gif, .ico, .jp2, .jpg,.lbm, .mdl, .mng, .pal, .pbm, .pcd, .pcx, .pgm, .pic, .png, .psd, .psp, .raw, .sgi, .tga y *.tif*.

- *Modo de despliegue*: a los modos de despliegue *Solid* y *Wireframe* se ha añadido otro llamado *White*, consistente en el dibujado de un objeto tridimensional totalmente blanco. La justificación de este nuevo despliegue reside en la necesidad de dibujar un cubo virtual que solape el cubo físico a modo de filtro descrito en la sección 5.3.1.

- *Lenguaje*: el importador está escrito en C++ en lugar de usar C.

- *Reserva de memoria*: a diferencia del importador desarrollado con fines docentes del formato *Orej*, se emplearon mecanismos de reserva de memoria dinámica que ofrece el lenguaje C++.

El exportador mejorado de *Orej* está desarrollado en Python al igual que el exportador original. La principal diferencia reside en la mejora del código. Antes, para incluir las coordenadas de los vértices y las coordenadas de los vértices uv, se hacían dos iteraciones sobre las listas de vértices de los objetos. Con el nuevo exportador, cada vez que se itera sobre la lista de vértices, se incluyen ambas coordenadas, reduciendo el tiempo de exportación.

Para recuperar los elementos que conforman el modelo tridimensional mediante el importador, se desarrollaron tres clases que diesen soporte a dicha tarea. Estas clases son las siguientes:

- Clase `Vertex`: encargada de recoger las coordenadas de cada vértice. Para ello emplea un array para almacenar las coordenadas (x, y, z) del vértice.

- Clase `Face`: cuya labor es determinar que vértices conforman una cara y las coordenadas uv de cada uno de ellos.

  La clase `Face` se encarga de recoger mediante un vector el conjunto de vértices que conforman una cara. En otro vector bidimensional almacena las coordenadas uv de los vértices anteriores.

- Clase `Object`: que contiene todas las características que definen un objeto tridimensional. En el listado 5.1 puede apreciarse la clase `Object`. En ella se observan los siguientes atributos:

  - *listMesh*: recoge el identificador único de la malla.
  - *scale*: define el tamaño al que será escalado el modelo.
  - *translation*: indica las coordenadas a las que se moverá el objeto en el espacio.
  - *textures*: almacena el identificador unívoco de la textura cargada por *OpenGL*.

- *correction*: define un parámetro de corrección de la intensidad de luz para las texturas
- *objectFaces*: recoge todos los objetos `Face` que conforman el objeto.
- *objectVertices*: guarda todos los objetos `Vertex` que tiene el modelo.
- *objectVertices*: guarda todos los objetos `Vertex` que tiene el modelo.

```
1   class Object {

3     int listMesh;
4     float scale;
5     float translation[3];
6     int correction;
7     std::vector<int> textures;

9     std::vector<Face*> objectFaces;
10    std::vector<Vertex*> objectVertices;
11  };
```

Listado 5.1: Clase Object

De esta forma, cada vez que se necesite trabajar con un objeto virtual, este será del tipo `Object`, facilitando el acceso a sus atributos.

El uso del formato *Orej* permite obtener modelos representados de forma independiente al lenguaje, suite de modelado, plataforma o biblioteca gráfica empleada en el despliegue. Bastará con redefinir el exportador y el importador para poder realizar las conversiones pertinentes, pero los objetos representados por este formato no se ven afectados.

A continuación se describe el subsistema de análisis encargado de localizar el foco de luz y asignar al objeto virtual la textura con la iluminación que mejor lo integre en la escena.

5.3. Subsistema de análisis

Para integrar de forma correcta el objeto virtual, éste debe poseer unas propiedades lumínicas acordes a ella. En la anterior sección se explicó cómo realizar un estudio de la iluminación mediante un entorno simulado para generar texturas que representasen diferentes escenarios.

La problemática que trata de abordar este subsistema es el de asignar al objeto virtual la textura que lo integre con mayor realismo. Para ello es necesario realizar un análisis de la iluminación de la escena, tratando de establecer la localización del principal foco de luz y asignando la textura en la que escenario simulado y realidad sean lo más parecidos.

Para alcanzar este objetivo, el subsistema de análisis realiza un estudio a la sombra proyectada de un objeto conocido, en este caso un cubo, que posee una localización y tamaño conocidos, además de proyectar la sombra sobre una superficie controlada.

Para realizar este análisis, el subsistema se diseñó en cinco módulos:

1. *Módulo de tratamiento gráfico*. Es el encargado de realizar las operaciones de edición sobre los *frames* de entrada del subsistema (ver sección 5.3.1), además de ofrecer ciertas operaciones matemáticas auxiliares necesarias en algunas de las etapas de este subsistema.

2. *Módulo de visión por computador*. Su función es analizar los *frames* de entrada e interpretar la escena en busca de la sombra arrojada por el cubo para obtener la posición del foco (ver sección 5.3.2).

3. *Módulo de gestión de texturas*. Este módulo se encarga de elegir la textura que mejor se ajuste a la iluminación de la escena una vez que ha sido detectado el foco. Además, realiza una adaptación en la misma para conseguir un nivel de realismo acorde con el entorno (ver sección 5.3.3).

4. *Módulo de depuración*. Informa al desarrollador sobre los cálculos que se están realizando para detectar el foco (ver sección 5.3.5).

5. *Módulo controlador*. Es el encargado de coordinar todo el proceso de análisis (ver sección 5.3.4)

A continuación se describe cada uno de estos módulos.

5.3.1. Módulo de tratamiento gráfico

El módulo de tratamiento gráfico es el encargado de realizar diversas acciones relacionadas con el tratamiento de imágenes. Además, como se describió en la sección 2.1, las

aplicaciones que emplean gráficos por computador requieren del álgebra lineal y de teoría matricial para poder transformar los objetos que despliegan. Al comienzo de la sección 5.3 se estableció que uno de los propósitos de este subsistema era determinar las coordenadas del foco en la escena real. Los cálculos realizados sobre las imágenes en busca de la sombra definida por el cubo están basados en un espacio bidimensional, mientras que la realidad viene determinada por un espacio tridimensional. Transformar de coordenadas 3D a coordenadas 2D y viceversa supone otro de los problemas que pretende solucionar este módulo.

El módulo de tratamiento gráfico aglutina tres submódulos: el submódulo de cálculo de proyección, el submódulo de filtros y el submódulo de operaciones matemáticas.

A continuación se describe cada uno de ellos.

Submódulo de cálculo de proyección

En determinadas ocasiones es necesario obtener la correspondencia entre una coordenada 3D y el *píxel* que ocupa en la pantalla o determinar, a partir de un *píxel* la coordenada tridimensional que lo representa en el espacio.

Para realizar la transformación 3D-2D, basta con acceder las matrices del pipeline para calcular como se proyecta la coordenada en la pantalla (ver sección 2.2.4).

Para la transformación 2D-3D se optó por acceder a la información del *buffer* de profundidad para ese *píxel*. Mediante la variable de profundidad y el *píxel* es posible establecer la coordenada tridimensional. El *buffer* de profundidad se llena con información cuando los objetos son dibujados, lo que plantea un serio problema en el caso de que no exista en la pantalla ningún objeto virtual desplegado que cubra ese *píxel*. Como solución se decidió pintar un plano sobre el área de proyección de la sombra inhabilitando el *buffer* de color. De esta forma el objeto no se pinta en la pantalla pero si se registra en el *buffer* de profundidad.

Para obtener la coordenada de un *píxel* a partir de una coordenada 3D, se empleó la biblioteca *OpenGL* que tiene la función `gluProject()` diseñada para tal efecto.

Para conseguir la correspondencia entre un punto de la pantalla y un punto 3D se consultó el valor de ese punto en el *buffer* de profundidad. A continuación, empleando el valor de profundidad y la coordenada del punto, se obtiene la coordenada tridimensional empleando la función `gluUnproject()`.

Esta sencilla aproximación se basa en funciones estándar que se ejecutan rápidamente en la GPU y evitan añadir carga computacional a la CPU del sistema.

A continuación se hablará de los filtros empleados en las imágenes de entrada.

Submódulo de filtros

El *frame* obtenido mediante la cámara tiene un tamaño muy grande en comparación al área donde se proyecta la sombra. Dado que el tiempo de cómputo de la aplicación debe ser pequeño, es necesario recortar la imagen. De esta forma, los algoritmos empleados en el análisis de la imagen tardarán menos en procesarla.

Además, uno de los problemas que presenta el análisis de la sombra es que las zonas de penumbra del cubo sean detectadas como tal, un comportamiento indeseable que se soluciona superponiendo en la imagen un cubo virtual en la misma posición que el cubo físico.

Superposición del cubo virtual

En algunas situaciones en las que la cámara esté frente al cubo físico y el foco detrás del mismo, la zona de penumbra del cubo puede ser computada como una prolongación de la sombra. Esto supone un gran problema, ya que la estimación del foco de luz depende exclusivamente de la sombra que genere el cubo. Esta interpretación errónea conlleva a una estimación engañosa.

La solución a este problema fue superponer sobre el cubo físico un cubo virtual blanco que "limpiase" la zona de penumbra no deseada.

Con este filtro, se obtiene una imagen con una sombra mejor definida sin las zonas de penumbra del cubo.

Recorte de la imagen

Como se comentó anteriormente, la imagen hay que recortarla para obtener el área de proyección de sombra. Para ello se conocen las coordenadas tridimensionales que conforman dicho área.

Tomando estas coordenadas, es posible calcular la correspondencia 2D en pantalla (ver sección 5.3.5). Una vez hecho esto, hay que calcular un tamaño adecuado para la imagen de forma que englobe el área de proyección de la sombra.

En ocasiones, la imagen obtenida puede presentar, al mover la cámara, pequeñas áreas que no corresponden únicamente al área de proyección de la sombra. Para evitar esto, todo el *frame* a excepción del área de proyección de la sombra es pintado de un color blanco, de forma que, ante un movimiento brusco de la cámara, estas zonas aparecen coloreadas y no influyen en el procesamiento de la sombra.

Otro problema relacionado aparece cuando el área no cuadra perfectamente con el rectángulo recortado. Esta situación puede introducir ruido en el procesamiento de la sombra, causando resultados inesperados. Para evitar este problema, se añade otro filtro a la imagen que colorea triángulos blancos en las zonas en las que el anterior filtro no actuó.

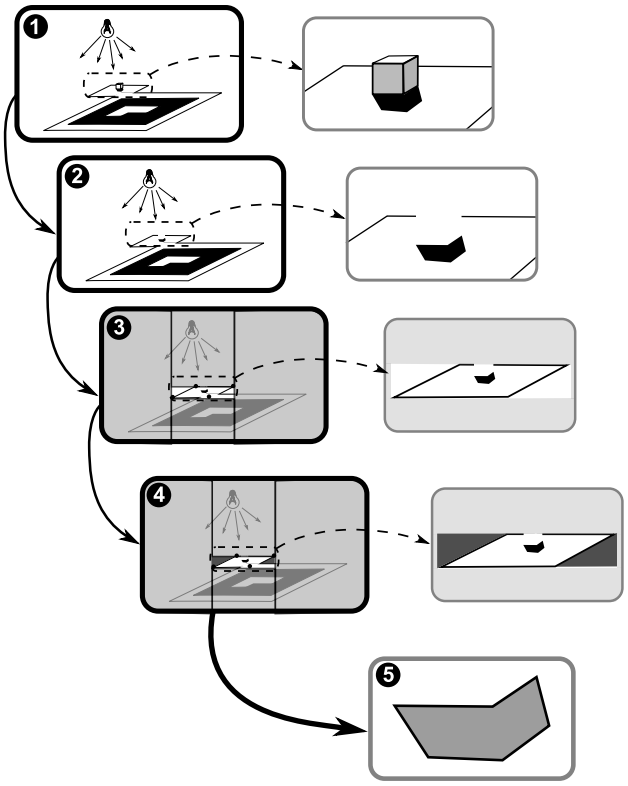

Figura 5.5: Filtros aplicados a la imagen. La figura 1 entrada sin ningún tratamiento. A la derecha de cada figura aparece la zona de proyección aumentada. La figura 2 muestra como resultado la superposición del cubo virtual. La figura 3 muestra como el resto de la imagen que no es la zona de proyección de sombra es eliminada con otro filtro y es recortada. El filtro de la imagen 4 elimina mediante triángulos las zonas sucias que el filtro de la imagen 3 no pudo limpiar. La figura 5 muestra el resultado de la aplicación de todos los filtros anteriores, obteniendo una sombra definida.

Un ejemplo de la aplicación de los filtros puede verse en la figura 5.5.

Para obtener la imagen resultante con los filtros aplicados se accede al *buffer* `GL_BACK` de *OpenGL*. Dibujando los filtros con *OpenGL* se descarga a la CPU de tener que realizar los cálculos necesarios, obteniendo mejores tiempos de ejecución.

Submódulo de operaciones matemáticas

Este módulo es el encargado de realizar las operaciones matemáticas que tengan cierta complejidad. Para ello realiza cálculos basados en álgebra lineal, empleando teoría de matrices y teoría vectorial para conseguir resolver los problemas del resto de módulos cuando sea preciso. Entre las operaciones soportadas están el cálculo de la distancia entre dos rectas, cálculo de la distancia entre un punto y una recta y cálculo de transformaciones de rotación, traslación y escalado de un vértice.

Estas funciones han sido implementadas siguiendo los principios matemáticos desarrollados en la sección 2.1.

El controlador `MathFunctions` es el encargado de ofrecer distintos servicios basados en operaciones matemáticas para que cualquier elemento del sistema pueda realizar los cálculos que estime oportuno.

5.3.2. Módulo de visión por computador

La labor de este módulo engloba una serie de operaciones basadas en algoritmos de tratamiento de imágenes para conseguir analizar la sombra que arroja el cubo añadido. El análisis debe dar como resultado la localización del foco de la escena real que será empleado para determinar y ajustar la textura que integre con mayor realismo al modelo virtual.

Para ello, siguiendo la teoría reflejada en la sección 2.2.3, el módulo debe determinar los vértices de la sombra generados como proyección de los vértices del cubo.

Sin embargo, las imágenes presentan gran cantidad de información que complica de forma sustancial el proceso de detección de la sombra. Para solucionar estos problemas, se aplican una serie de filtros, descritos en la sección 5.3.1, que reducen la información de la escena a la presente en el área de proyección de sombra.

Para poder completar este objetivo, este módulo está compuesto de otros módulos que pasan a comentarse a continuación.

Submódulo de estimación de la sombra

El primer paso para realizar el estudio de la iluminación de la escena a partir del análisis de la sombra es determinar dónde se encuentra y obtener su forma. Empleando técnicas propias de la visión por computador se puede realizar una clasificación de la imagen atendiendo a los colores que presenta y definir así que partes pueden ser consideradas sombra. Dado que gran parte de la información de la imagen ha sido eliminada gracias a los filtros, se reduce el nivel de complejidad para localizar la sombra.

Una vez determinadas las regiones donde puede haber sombra con una probabilidad alta, se estudia cuál de las áreas obtenidas tiene más posibilidades de serlo. Terminada esta elección, es necesario obtener los vértices que la forman para calcular así las coordenadas que la definen.

Para realizar esta labor, la imagen es sometida a una serie de fases que pasan a describirse a continuación (ver Figura 5.10 (Analizar Imagen)):

1. *Cálculo del umbral*. El umbral de una imagen determina a partir de qué valor se asignará el color blanco o negro a cada *píxel* de la imagen. de color la imagen se tornará de color blanco o negro. La finalidad de esta etapa es determinar qué color tiene apro-

ximadamente la sombra para umbralizar la imagen con ese valor y obtener su forma bien definida.

Para realizar el cálculo del umbral se emplea el histograma de la imagen en escala de grises. El criterio seguido por *WILi* consiste en determinar, descartando los valores blancos y negros, cuál es el valor de la base de proyección de la sombra. Este valor corresponde al valor más repetido del histograma. Una vez calculado, hay que encontrar el valor mínimo de color que aparezca en la imagen. Determinados estos dos valores, el umbral es el valor resultante de hacer la media aritmética de los mismos.

2. *Binarización de la imagen*. Una vez calculado el umbral, se binariza la imagen eliminando el resto de los elementos que no aportan información relevante para las siguientes etapas. La descripción teórica de este proceso puede encontrarse en la sección 2.3.1.

   Para binarizar la imagen se empleó la función `cvThreshold()` de *OpenCv*.

3. *Detección de contornos*. Una vez determinada la imagen binarizada, es necesario detectar su contorno. *OpenCv* está provisto de la función `cvFindContours()`, que devuelve una secuencia de puntos que la define. En la sección 2.3.1 puede encontrarse una explicación en detalle de lo que es un contorno y cómo es representado.

   Tras binarizar la imagen puede ocurrir que, además de la sombra, aparezcan pequeños elementos que formen parte de una falsa detección . Con estos elementos en medio de la imagen, *OpenCv* detectaría la existencia de más de un contorno. La solución encontrada para determinar con certeza la sombra ha sido recorrer cada uno de los contornos encontrados y estimar cuál representa la sombra en función de su área. Así se definirá el contorno de la sombra como el contorno con mayor tamaño.

4. *Reducción de vértices*. Una vez determinado el contorno, hay que disminuir el número de puntos que lo definen para reducir el tiempo de cálculo en los algoritmos de búsqueda y para concretar los vértices que definen la sombra. Para ello la función `cvApproxPoly()` de *OpenCv* es empleada para " poligonizar" el contorno. La clave del éxito de esta función reside en la elección de un buen parámetro de precisión k, como se comentó en la sección 2.3.1. Tras diversas pruebas de ensayo y error y, siguiendo la recomendación de [BK08], que indica que el parámetro debe ser una fracción pequeña del perímetro del contorno, se optó por tomar dicho parámetro como el cuatro por ciento de su perímetro, es decir, $k = 0{,}04$.

$$P_{precision} = perimetro(contorno) * k$$

De esta forma se reduce la cantidad de vértices para representar la sombra. En el listado 5.2 se puede apreciar cómo se hace una búsqueda de los contornos, comparando el tamaño de los mismos obtenido por la función `cvContourArea()`. Si la búsqueda tuvo éxito, entonces se realiza la reducción de puntos del contorno.

```
1  CvSeq *
2  ShadowAnalyzer::getPolyContours(IplImage* binImg)
3  {

5    CvSeq * first_contour = NULL;
6    CvSeq *h = NULL;
7    CvMemStorage *storage_cont = cvCreateMemStorage(0);
8    CvMemStorage *storage_poly = cvCreateMemStorage(0);
9    CvSeq *poly = NULL ;

11   /* Finding contours*/
12   cvFindContours(binImg, storage_cont, &first_contour, sizeof(CvContour),
13                  CV_RETR_TREE, CV_CHAIN_APPROX_NONE,  cvPoint(0,0));

15   int max =0;
16   int aux;
17   /* Looking for the biggest contour*/
18   for( CvSeq* c = first_contour; c != NULL; c = c->h_next ) {
19     for( h = c->v_next ; h != NULL; h = h->v_next ) {
20       aux = cvContourArea(h);
21       if (aux > max)
22       {
23         max = aux;
24         poly = h;
25       }
26     }
27   }

29   if(poly != NULL)
30   {
31     /* Reducing contours*/
32     poly = cvApproxPoly(poly, sizeof(CvContour), storage_poly,
33                         CV_POLY_APPROX_DP,cvArcLength(poly)*0.04, 0);
34   }

36   return poly;
37 }
```

Listado 5.2: Obtención del contorno de la sombra poligonizado

En este submódulo se han explicado las etapas empleadas en la detección de los puntos principales que definen el contorno de la sombra proyectada. En el siguiente submódulo se explicará cómo es capaz de determinar los vértices que intervienen en el proceso de cálculo del foco.

Submódulo de cálculo de vértices

Tras el análisis de la sombra y la obtención de los vértices más importantes de la misma, el siguiente paso es determinar los vértices principales tanto de la sombra como del cubo.

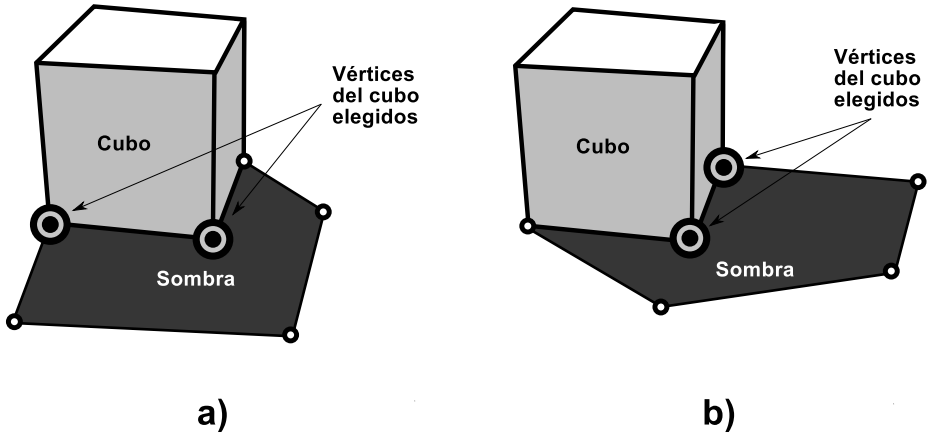

Figura 5.6: . Elección de vértices según la orientación de la sombra. Si la mayor parte de la sombra se encuentra a la izquierda del cubo, entonces se toman los vértices de la cara izquierda del mismo (a). En el caso de que la mayoría estén hacia la derecha, son los vértices de la cara derecha del cubo los elegidos(b).

En primer lugar, se buscan los vértices de la sombra que concuerden, admitiendo un pequeño margen de error, con los vértices de la base del cubo.

Los vértices que coinciden con el cubo pueden ser dos (si la sombra entra en contacto únicamente con una de las caras del cubo) o tres (si entra en contacto con dos de ellas). En la figura 2.6 puede verse una sombra que entra en contacto con dos caras y que por tanto coincide con la base del cubo en tres vértices.

Desde un primer momento, *WILi* fue diseñado para trabajar con dos vértices (elementos suficientes para determinar el foco de luz), por lo que, en caso de tener tres vértices de contacto con la sombra, hará una estimación sobre cuál es la cara cuyos vértices contribuirán a una mejor estimación del foco de luz.

Este cálculo lo realiza estimando hacia qué lado está orientada la sombra. Si la sombra presenta más vértices con una posición más a la derecha del cubo, se toman los vértices de la cara de la derecha . Sin embargo, si los vértices se reparten más en la zona izquierda, se tomarán los vértices de la cara de la izquierda. La figura 5.6 muestra un ejemplo de esta elección.

La razón por la que es necesario saber qué vértices de la sombra están en contacto con los vértices del cubo se debe a que es necesario determinar cuáles serán los vértices superiores del cubo que por intersección de los rayos del foco de luz, generan los vértices de la sombra.

Una vez conocidos los vértices del cubo que participarán en el cálculo del foco de luz, hay que encontrar sus proyecciones en la sombra. Para esto, tal y como muestra la figura

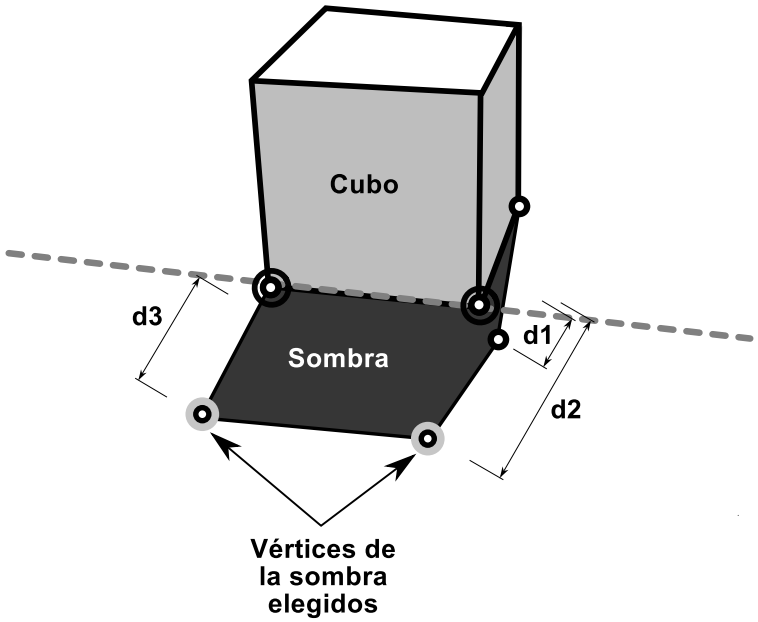

Figura 5.7: Búsqueda de vértices de la sombra. Una vez determinados los vértices de la sombra que entran en contacto se traza una linea que los una. A continuación se eligen los dos vértices más alejados para representar las proyecciones de las esquinas superiores del cubo.

5.7, se emplean los vértices de la sombra coincidentes con los vértices del cubo, se traza una recta. A partir de aquí, se buscan los puntos de la sombra más lejanos al cubo, que serán las proyecciones que se estaban buscando.

Una vez obtenidos los elementos que intervienen en la proyección de la sombra, pueden emplearse para obtener el foco que la generó. Para ello, en el siguiente apartado se describirán los pasos y cálculos necesarios para obtenerlo.

Submódulo de estimación del foco

Una vez localizados los vértices de la sombra, hay que obtener a partir de sus coordenadas 2D las coordenadas 3D antes de que se proyectaran en perspectiva. Trazando dos rectas, una por cada par de vértices sombra-cubo, se determina el foco en el punto donde la distancia entre las rectas es mínima siguiendo el algoritmo de distancia mínima entre dos rectas estudiado en la sección 2.1.1. La figura 5.8 resume cómo se realiza este cálculo.

Tras haber determinado la posición del foco de luz con el módulo de visión por computador, hay que elegir la textura que mejor se adapte al objeto virtual en el entorno. A continuación se describe el módulo de gestión de texturas encargado de esta función.

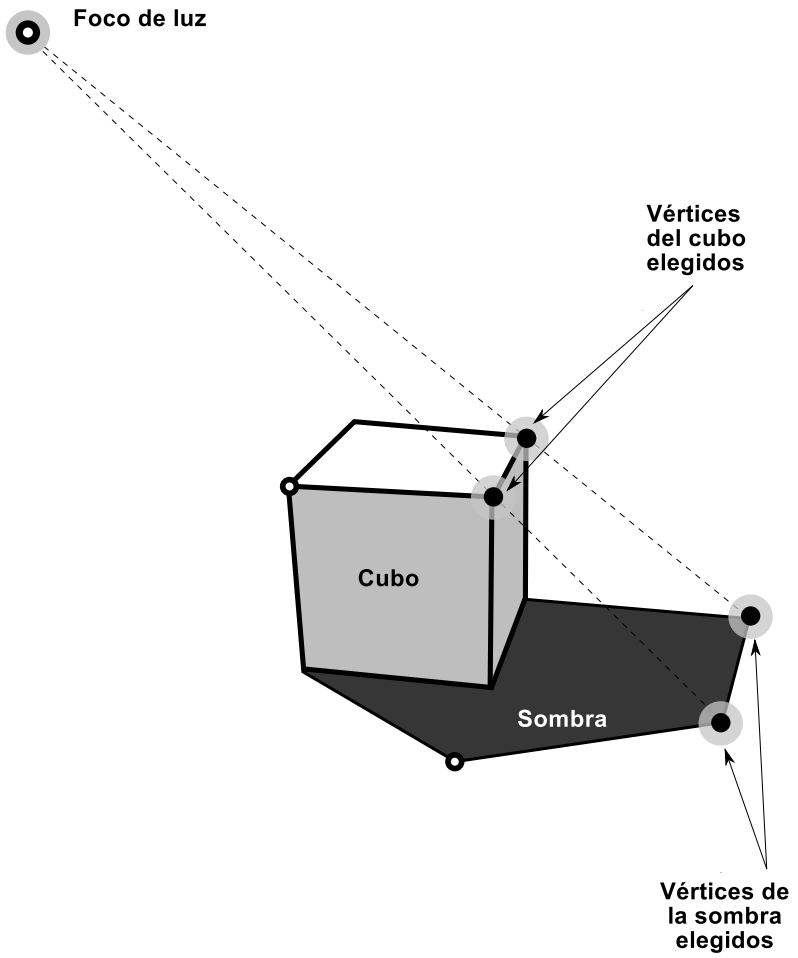

Figura 5.8: . Cálculo del foco a partir de los vértices detectados.

5.3.3. Módulo de gestión de texturas

El módulo de gestión de texturas es el encargado de buscar, asignar y modificar la textura asociada a un objeto de forma que mejor se integre en la escena real.

Este módulo conoce todas las texturas disponibles de la etapa de precálculo de texturas(sección 5.2) y asigna al objeto la que represente de mejor forma a la iluminación correspondiente al foco encontrado.

Para ello presenta un parser que interpreta el archivo de configuración donde aparecen las texturas generadas. El módulo de gestión de texturas cuenta, además, con un submódulo encargado de elegir el foco usado en la etapa de precálculo de texturas que definía con mayor precisión la iluminación generada por la fuente de luz del mundo real. Presenta también otro submódulo encargado de realizar un ajuste en las texturas en base a la intensidad de la fuente de luz del mundo real y otro submódulo encargado de cargar la textura modificada y aplicársela al objeto virtual (ver sección 5.3.3).

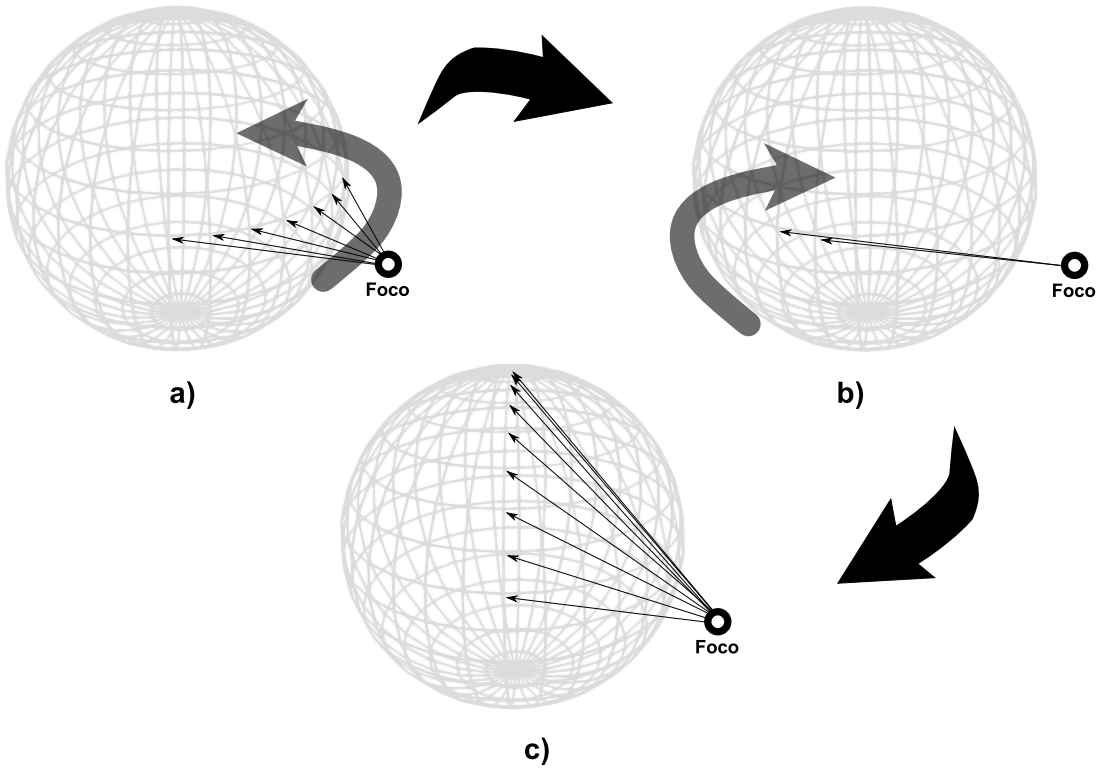

Figura 5.9: Algoritmo de búsqueda de la mejor textura. En primer lugar se determina el sector sobre el que se realizará la búsqueda, realizando una búsqueda en sentido contrario a las agujas de reloj (a). A continuación, se realiza una búsqueda en el sentido contrario (b), para terminar buscando un foco con una menor distancia en el sector calculado (c).

Función de proximidad

Este submódulo es el encargado de determinar qué texturas de las disponibles van a representar con mayor exactitud la iluminación del objeto dentro de la escena. Para ello se ha diseñado un algoritmo que busca, minimizando el coste computacional de la operación, la textura que mejor se ajuste a las características del entorno.

El módulo conoce las texturas disponibles para ese modelo, obtenidas como salida de la fase de precálculo de texturas. Como entrada toma el fichero generado por el módulo de precálculo de texturas (ver sección 5.2.1). Hay que recordar que el nombre de los archivos de las texturas codifican las coordenadas de los focos virtuales generados en el entorno controlado de la etapa de precálculo de texturas. Mediante un parser, el módulo recupera uno a uno los nombres de las texturas obtenidas ordenadas por su coordenada z. El hecho de que presente este formato responde a las necesidades de mejorar el proceso de búsqueda de la textura.

Tomando como referencia la figura 5.9, se describirá el algoritmo empleado.

En primer lugar, una vez que se dispone de la localización estimada de la fuente de luz del mundo real, se intenta buscar el sector en el que se va a encontrar el foco de menor

distancia. Para ello, tomando el anillo $z = 0$, comienza una búsqueda en el sentido contrario a las agujas del reloj, buscando los vértices que presenten una distancia menor con el foco encontrado. En el momento en el que la distancia comienza a ser creciente, se pasa a buscar en el sentido inverso hasta volver a dar con distancias crecientes. Es en este momento cuando se interrumpe la búsqueda, escogiendo para el siguiente paso el sector que contiene ese vértice. El simple hecho de descartar el resto de sectores del proceso de búsqueda hace que el tiempo estimado en completarla disminuya significativamente.

Después, determinado el sector, se recorre comprobando la distancia entre el foco del entorno virtual y el foco encontrado en la escena real. En el momento en que esta distancia es creciente, puede pararse la búsqueda , devolviendo las coordenadas del foco con la distancia mínima encontrada.

Una vez conocidas las coordenadas, se determina directamente el nombre de la textura que se cargará.

Submódulo de ajuste de niveles

La labor principal de este submódulo es ajustar la textura al nivel de intensidad de la fuente de luz y a la iluminación global de la escena. Para ello, se utiliza el valor umbral calculado en etapas anteriores y se compara con el valor negro absoluto y se determina un factor de corrección que será aplicado a la textura cuando se despliegue.

Submódulo de selección de textura

Este submódulo es el encargado de acceder a la estructura del objeto y asignarle a su textura la imagen editada, de forma que cuando se despliegue, presente un aspecto realista.

Cabe destacar de este submódulo que no se crea en ningún momento una textura adicional a la que tenía asociada el propio objeto. Simplemente se cambia la imagen de su textura sin crear una nueva.

5.3.4. Módulo controlador

El módulo controlador es el encargado de coordinar las distintas fases por las que pasa la imagen capturada por la cámara hasta que, tras determinar la posición del foco de luz, consigue que el objeto tenga cargada la textura correspondiente.

Este módulo va realizando una serie de pasos en los que emplea los distintos módulos y submódulos descritos en este subsistema. Cada vez que termina la ejecución de algún módulo, obtiene una imagen o valor de salida que será empleado como entrada para la siguiente tarea.

La figura 5.10 presenta el diagrama de flujo del subsistema de análisis con las acciones coordinadas por el controlador.

5.3.5. Módulo de depuración

La depuración de una aplicación que utiliza visión por computador y realidad aumentada es una tarea más difícil y tediosa. La dificultad viene dada porque hay una serie de factores que van incorporando errores a los resultados obtenidos. Es importante tener en cuenta estos errores, ya que en etapas posteriores es complicado mitigarlos y pueden afectar en la precisión del resultado final obtenido.

A continuación se describen algunos de estos errores.

Errores inherentes en la imagen digital

La obtención de una imagen digital no concuerda totalmente con la realidad. Esto se debe a una serie de factores que pasan a describirse a continuación:

- **Problemas de óptica**.

  La calidad de la imagen obtenida depende directamente, entre otros factores, de la calidad óptica de la cámara. Dado que es el instrumento que obtiene imágenes digitales y sirve de entrada a todo el proceso de análisis de *WILi*, la cámara es uno de los dispositivos hardware más sensibles a los que debe prestarse una especial atención.

  A continuación se describen algunos de los factores relativos a la cámara que pueden degradar la calidad de la imagen:

  - **Enfoque de la cámara**

    Solo se pueden enfocar elementos que estén a una determinada distancia, por lo que puede darse situaciones en las que la imagen tenga objetos borrosos.

  - **Tiempo de exposición**

    El tiempo de exposición influye en la iluminación de la imagen. A mayor exposición, mayor es la cantidad de luz recogida por la cámara. Con valores muy grandes de exposición o muy pequeños, la imagen puede verse saturada.

    Además, puede ser que en el momento de realizar la captura, recoja objetos que estuviesen moviéndose, presentando estos deformidades y estando mal enfocados.

  - **Sensibilidad**

    Si la cámara está configurada para tener una alta sensibilidad, es posible que aparezca ruido en la imagen.

    Cabe destacar que muchas cámaras no permiten realizar ajustes en los parámetros anteriormente citados, sino que se limitan a realizar ajustes automáticos.

    Por tanto, la calidad de la cámara influirá de forma directa en la calidad de sus componentes y en la calidad en la configuración automática de sus parámetros.

Como consecuencia de esto, la calidad de la cámara determinará directamente la calidad de la imagen obtenida.

- **Discretización de la realidad dentro del espectro RGB**.

  Al digitalizar la imagen con una cámara, se pasa de un espacio continuo a un espacio discreto (digital). En el caso de usar el espectro RGB, la realidad pasa a ser representada únicamente con 256^3 colores. Esto causa que aparezcan pérdidas en la información del color.

  Otro factor importante es que las cámaras son capaces de captar radiación que se escapa al espectro reconocible por el ser humano, pudiendo mostrar elementos que a la vista no aparecen. Por ejemplo, las cámaras son capaces de detectar la radiación infrarroja, como la de un mando a distancia, cuando en la realidad esta radiación no es visible.

- **Aleatoriedad del entorno de ejecución**.

  La iluminación de una escena presenta unas variaciones imperceptibles para el ojo humano pero que si son sensibles a una cámara. Aún controlando la posición de todos los objetos que intervengan en una escena así como la intensidad de la iluminación, la trayectoria de la misma sigue patrones aleatorios que hacen muy difícil su tratamiento. Por ejemplo, una sombra ante un foco tiene una posición y medidas determinadas. Sin embargo, pequeñas variaciones(ruido) a nivel de *píxel* pueden dar lugar a cambios en los algoritmos de detección de contornos, afectando así a los cálculos realizados.

Imprecisión en correspondencia 3D/2D

Como se explicó en la sección , usando la función `gluProject()` de *OpenGL* se obtiene la correspondencia entre un punto con coordenadas tridimensionales y el *píxel* que lo representa en la pantalla.

El problema de imprecisión observado se debe a que, mediante unas coordenadas en punto flotante, se obtiene otro punto bidimensional en coma flotante. Sin embargo, los píxeles vienen representados por coordenadas enteras, siendo necesario truncar o redondear el punto en coma flotante para obtener su *píxel* equivalente. Este redondeo puede dar lugar a errores que, dependiendo del tamaño del objeto a estudiar, pueden ser relevantes.

Como se ve en la figura 5.11, un punto determinado del espacio puede tener una correspondencia en un punto bidimensional en coma flotante que no tiene por qué coincidir con una coordenada entera equivalente a un *píxel*. En el peor de los casos, como muestra la figura, el punto bidimensional puede estar situado en medio de cuatro píxeles. Ante esta situación, hay que, bien truncar las coordenadas para elegir el *píxel* superior izquierdo, o bien redondearlas de alguna forma que se ajuste a alguno de estos cuatro píxeles. En cualquier caso, la decisión que se tome influirá en la precisión con la que se calcule el foco.

WILi establece como *píxel* por defecto el obtenido como resultado de truncar las coordenadas de dicho punto bidimensional.

Este error de cálculo no puede ser solucionado salvo estudiando el comportamiento histórico de la región estudiada y realizando algún tipo de heurística que mitigue el ruido, ya que no depende de un cálculo sino de una equivalencia entre un valores en coma flotante y su discretización en valores enteros.

Herramientas de depuración

El módulo de depuración es el encargado de ofrecer mecanismos para que el desarrollador pueda verificar las distintas acciones que se están llevando a cabo. Para depurar el subsistema de análisis, *WILi* puede desplegar un sistema de pantallas que representa cada una de las etapas más importantes visualizando la salida que producen.

Actualmente, *WILi* presenta las siguientes pantallas de depuración (ver Figura 5.12)

- *Imagen de entrada*. Esta pantalla visualiza la imagen de entrada recortada y con los filtros aplicados.
- *Imagen binarizada*. Muestra el resultado de binarizar la imagen. Esta ventana ayuda a que el desarrollador pueda determinar, comparando con la imagen de entrada, si el resultado ha sido correcto.
- *Contorno detectado*. En esta ventana se visualiza el contorno detectado en color verde sobre la imagen de entrada. Además, pinta de azul los vértices que definen la sombra una vez poligonizado el contorno.
- *Elección de vértices*. Para poder verificar la elección de los vértices de la imagen y el cálculo del foco, se establece una imagen en la que se pintan con colores distintos los vértices calculados, además de otra información relevante.

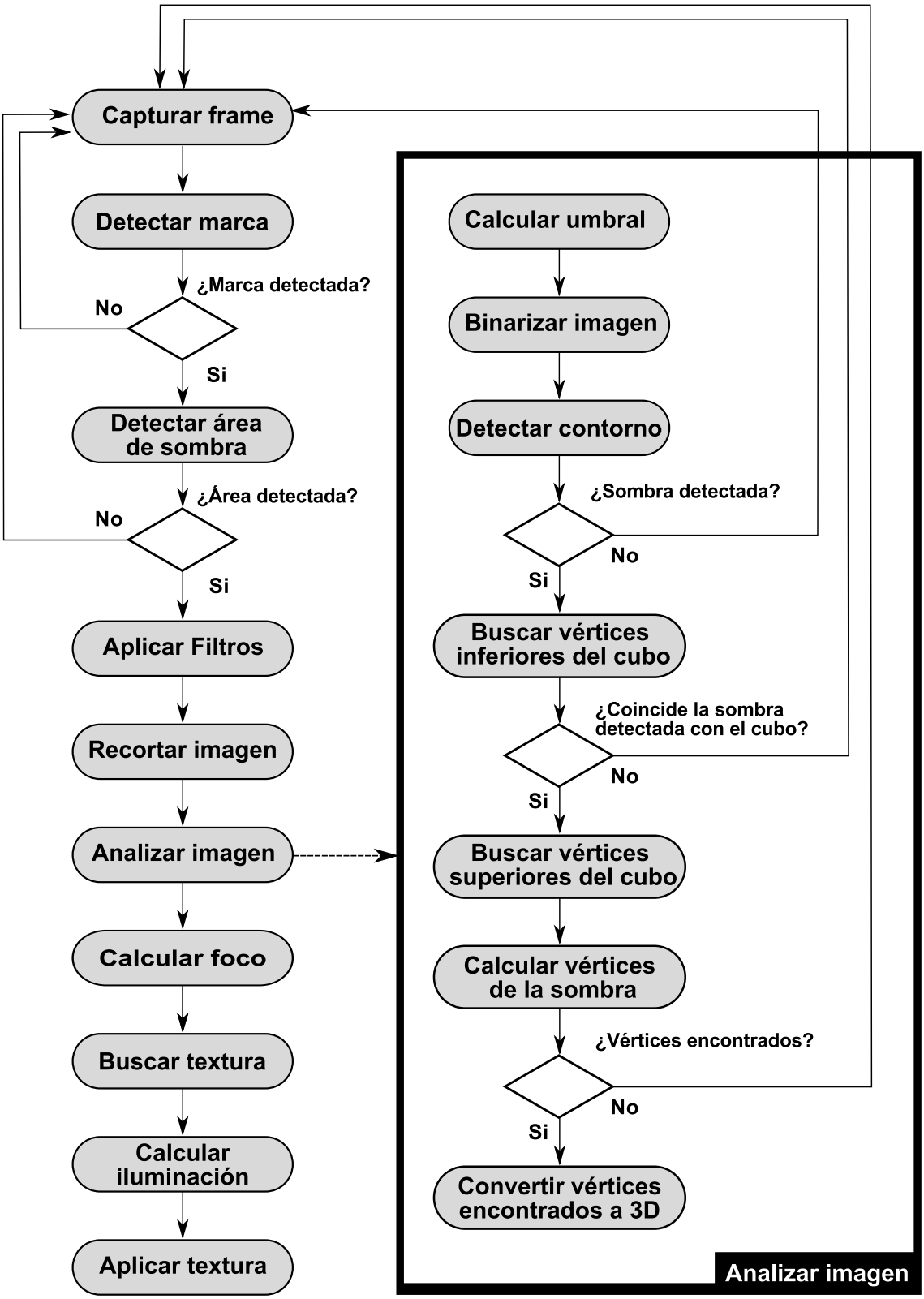

Figura 5.10: Diagrama de flujo del subsistema de análisis.

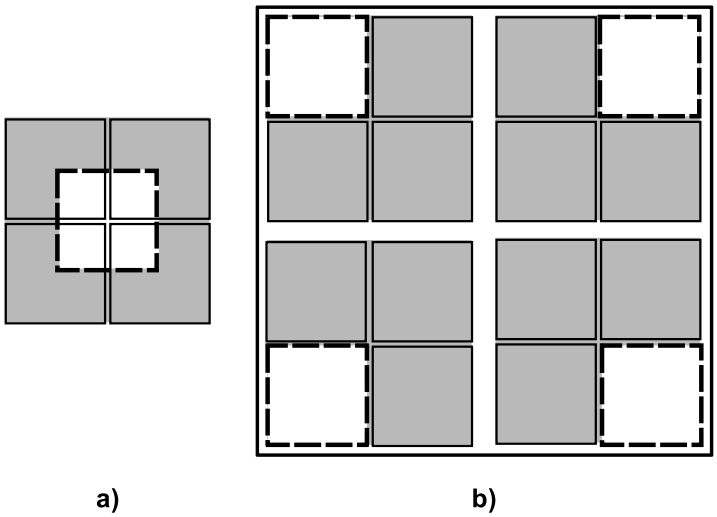

Figura 5.11: Imprecisión en la correspondencia 3D/2D. En la figura de la izquierda se observa, en blanco, un punto bidimensional que no presenta una correspondencia directa con los píxeles mostrados (mostrados en gris). A la derecha se muestra los posibles valores que puede tomar dicho punto.

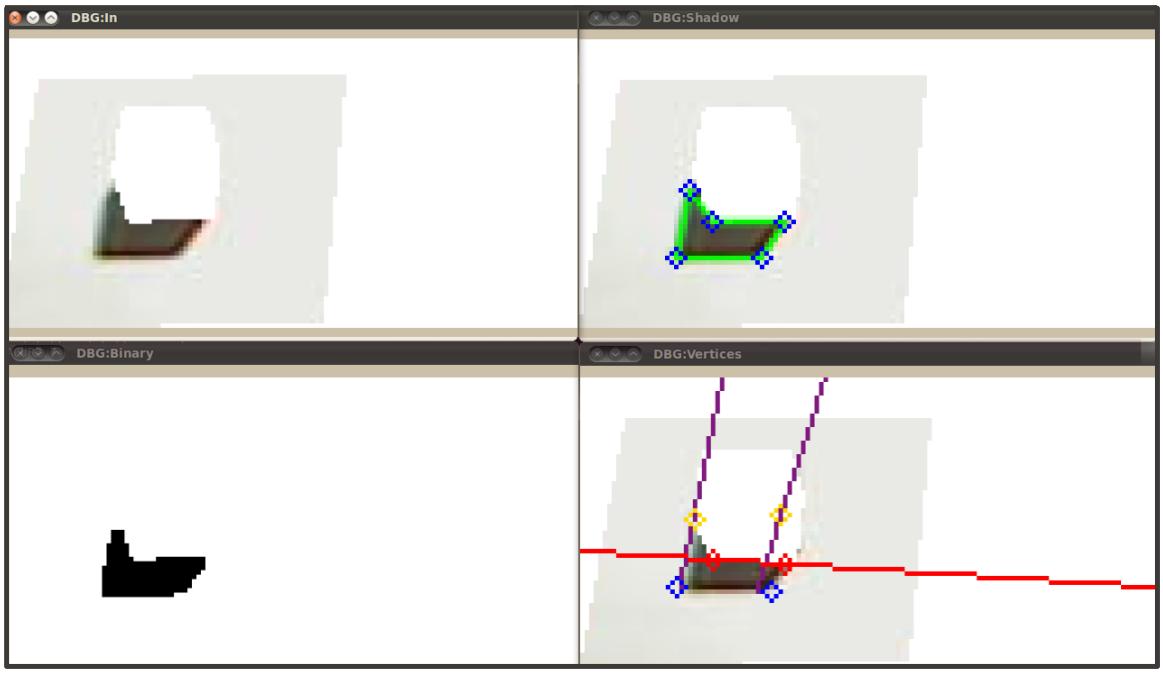

Figura 5.12: Captura de las ventanas de depuración.

5.4. Subsistema de representación

El subsistema de representación es el encargado de llevar a cabo las acciones relacionadas con el dibujado de objetos 3D y 2D. Este subsistema es capaz de desplegar objetos, cargar mallas de vértices, cargar y modificar texturas y dibujar imágenes planas.

Este subsistema pretende abstraer acciones muy específicas que dependen de la biblioteca de gráficos 3D empleada.

A su vez, el subsistema se divide en dos módulos, el *módulo de entidades 2D* y el *módulo de entidades 3D* ambos basados en la biblioteca *OpenGL*. A continuación se describe cada uno de ellos.

5.4.1. Módulo de entidades 2D

Este módulo se encarga de dibujar tanto primitivas bidimensionales (como un cuadrado o una línea) como de cargar y dibujar imágenes planas.

Para dibujar en pantalla imágenes planas, la técnica empleada consiste en la carga de un cuadrado al que se le asocia como textura la imagen que se desea cargar. La principal peculiaridad es que hay que cambiar el modo de dibujado en perspectiva por una vista ortográfica. Una vez dibujada la textura, se restaura la vista en perspectiva. Para habilitar y deshabilitar el modo de dibujado en 2D se implementaron las funciones `enable2D` y `disable2D` de la clase `GlDrawer`.

```
1  void
2  GlDrawer::enable2D()
3  {
4
5    GLint viewport[4];
6    glGetIntegerv(GL_VIEWPORT, viewport);
7    /* Loading screen values*/
8    int minX = viewport[0];
9    int minY = viewport[1];
10   int maxX = viewport[2];
11   int maxY = viewport[3];
12
13   glDisable(GL_DEPTH_TEST);
14
15   glMatrixMode(GL_PROJECTION);
16   glPushMatrix();
17   glLoadIdentity();
18
19   /* Changing  to orthogonal view*/
20   glOrtho(minX, maxX, minY, maxY, -1, 1);
21
22   glMatrixMode(GL_MODELVIEW);
23   glPushMatrix();
24   glLoadIdentity();
25 }
```

```
27  void
28  GlDrawer::disable2D()
29  {
30    /* Restoring modelview*/
31    glMatrixMode(GL_MODELVIEW);
32    glPopMatrix();

34    /* Restoring projection*/
35    glMatrixMode(GL_PROJECTION);
36    glPopMatrix();

38    /* Ready to draw in 3D mode*/
39    glEnable(GL_DEPTH_TEST);
40    glMatrixMode(GL_MODELVIEW);

42  }
```

Para determinar el tamaño de la pantalla, se puede recurrir a la transformación de pantalla (*Viewport*). Leyendo sus atributos se puede determinar la posición x e y de la ventana, que en la mayoría de los casos es 0. El tercer y cuarto parámetro corresponde a la anchura y altura de la pantalla respectivamente.

Otra función a tener en cuenta es `glOrtho()`, que multiplica la matriz de transformación actual por una matriz ortográfica, abandonando la vista en perspectiva. La finalidad de esto es que los objetos se plieguen con el tamaño establecido independientemente de la profundidad.

Para deshabilitar la función de dibujado en 2D se obtiene la matriz anterior de las pilas de matrices `GL_MODEL_VIEW` y `GL_PROJECTION` para volver al modo anterior (que fue convenientemente guardado con una llamada a `glPushMatrix()`).

5.4.2. Módulo de entidades 3D

El módulo de entidades 3D es el responsable de cargar mallas de objetos tridimensionales, desplegar objetos y asignarles texturas.

Para poder realizar estas tareas se empleó la clase `Object`, favoreciendo la abstracción de las propiedades particulares de los objetos que pueden ser representados.

En la carga de las texturas se empleó la biblioteca *DevIl*, desarrollada a específicamente para que soporte los mismos tipos de datos que *OpenGL*. Esta decisión en el diseño de la biblioteca contribuyó a evitar que se complicase más el proceso de carga de textura en el objeto virtual.

Para cargar una textura sobre un modelo, se siguen una serie de pasos que se explican a continuación:

- *Creación del objeto textura*. Para crear un objeto textura se emplea `glGenTextures()` encargado de su generación. Con esa función se obtiene un identificador único del objeto textura, de forma el objeto queda cargado y puede ser usado posteriormente con este identificador. Para poder usar una textura, hay que marcarla como objeto activo en *OpenGL* utilizando `glBindTexture()`. La finalidad de marcar un objeto textura como activo es indicar qué textura es la que se desea usar y sobre la que se van a realizar las siguientes operaciones.

- *Asignación de la textura al objeto activo*. Una vez creado, activado y configurado el objeto textura, hay que asignarle la textura propiamente dicha, es decir, la imagen que queremos desplegar en el objeto. Obtenida la imagen, se asigna al objeto textura con `glTextImage1D`/`glTextImage2D`/`glTextImage3D`, dependiendo del tipo de textura de una dimensión, bidimensional o tridimensional.

A continuación se describen algunas características a tener en cuenta en la representación de objetos 2D/3D.

Convenios de colores

En esta sección se describirán los convenios más comunes en la informática gráfica que han tenido que tenerse en cuenta en el desarrollo de *WILi*. Según las bibliotecas con las que se trabaje, los colores adquieren diferentes formas de representación. Partiendo de una representación que emplee 24 bits para representar los colores, destacan:

- *RGB*. Es el formato empleado por *OpenGL* que utiliza 8 bits para representar cada canal de color rojo (R), verde (G) y azul (B). Cada canal puede tomar valores que van desde el 0 hasta el 255 empleando esos 8 bits.

- *BGR*. Es el formato empleado por defecto en *OpenCv*. Difiere del anterior formato únicamente en el orden en el que los canales son almacenados. En este caso en primer lugar se representa el nivel de azul, seguido del verde y en último lugar, el rojo.

Orígenes de coordenadas

A la hora de implementar aplicaciones basadas en gráficos por computador, el desarrollador tiene que tener muy claro el sistema de referencia que emplean las bibliotecas con las que trabaja, ya que pueden diferir en el mismo. Al definir un vértice se indica su coordenada en base a un origen del Sistema de Referencia Universal. Este origen es definido por convenio y puede diferir entre bibliotecas (no hay un estándar aceptado por todas las implementaciones).

Se han observado dos tipos distintos de coordenadas en el espacio 2D:

- *Bottom-Left*. Como su nombre indica, el origen de coordenadas parte de la esquina inferior izquierda y corresponde al origen de coordenadas empleado por *OpenGL*.
- *Top-Left*. En este caso, el origen está localizado en la esquina superior izquierda. Este origen de coordenadas es típico de la biblioteca *OpenCv* y de *Glut*, la biblioteca externa de *OpenGL* para el manejo de ventanas.

Trabajar con una imagen empleando las dos bibliotecas tiene como inconveniente tener que realizar transformaciones basadas en volteados horizontales o *flips* que permitan obtener una imagen correcta.

El trabajo en el espacio 3D requiere igualmente tener en cuenta los diferentes convenios tomados por las bibliotecas empleadas. El sistema de bases ortonormales de *OpenGL*, por ejemplo, es diferente del empleado internamente por *ARToolKit*

CAPÍTULO 6

EVOLUCIÓN, RESULTADOS Y COSTES

En este capítulo se describen las fases empleadas en el desarrollo de *WILi*, mostrando los hitos que las definen, así como información específica relativa a la complejidad y complejidad asociada.

Al final del capítulo se resume de forma global el proyecto, proporcionando una estimación general del coste asociado al desarrollo del mismo.

6.1. Evolución del proyecto

En esta sección se describirán las fases empleadas en el desarrollo del proyecto siguiendo la metodología especificada en el capítulo 4. En una primera etapa, se trabajó en un estudio sobre el estado del arte en busca de proyectos similares y se determinaron algunas de las bibliotecas que se emplearían. Después de esto, se establecieron los primeros requisitos y se tomaron las primeras decisiones sobre el diseño. Una vez concluida esta fase preliminar, se comenzó el desarrollo por iteraciones del proyecto que se recogen a continuación.

6.1.1. Infraestructura para la metodología

Como sistema de control de versiones se utilizó *Subversion*, teniendo un repositorio público alojado en `https://forja.rediris.es/projects/cusl5-wili` correspon-

diente a la Forja de la Comunidad del Centro de Excelencia de Software Libre de Castilla-La Mancha (*CESLCAM*).

6.1.2. Concepto del software

Con un mercado en auge, la Realidad Aumentada requiere de técnicas innovadoras que permitan al usuario sentir nuevas experiencias, complementando su realidad con objetos virtuales. Sin embargo, la integración de los mismos no se produce de forma realista.

Este proyecto surge ante la necesidad de incorporar nuevas formas de integración realista y avanzar en dicho campo ofreciendo una alternativa basada en el precálculo de la iluminación empleando métodos de síntesis realista.

6.1.3. Análisis preliminar de requisitos

Tras establecer la temática del proyecto y determinar qué se pretendía con el mismo, se realizó una revisión del estado del arte en busca de artículos científicos relacionados con el realismo de aplicaciones de Realidad Aumentada. Se descubrió que se trataba de un campo de reciente investigación en la que apenas había alternativas y en la que se podrían emplear mecanismos de precálculo de la iluminación empleando métodos de síntesis de imagen realista.

Además, como método de análisis de la iluminación, se planteó el uso de un entorno controlado basado en el estudio de la sombra arrojada por un objeto en la escena para determinar la posición del foco principal del entorno y poder así ajustar las propiedades lumínicas del objeto.

Tras estas comprobaciones y período de investigación, se determinó la necesidad de tener dos sistemas independientes que separasen la etapa de precálculo de la iluminación con la de análisis y aplicación de texturas. Esta decisión no sólo se basó en el principio de Ingeniería del Software de desacoplamiento de componentes, sino que además se tuvieron en cuenta factores como el lenguaje de programación, el uso de otras bibliotecas y la definición de objetivos adicionales. De este modo se definieron tres subsistemas que separan en otra capa independiente los modelos de representación.

Después de esta decisión de diseño y, teniendo en mente los principales objetivos, se pasó a estudiar y determinar las bibliotecas que serían empleadas, teniendo en cuenta que debían ser compatibles con la licencia elegida en el desarrollo del proyecto (GPL v3).

También se tomaron decisiones sobre el método de registro que se emplearía en la aplicación. Debido al amplio uso en la comunidad de desarrolladores de aplicaciones de Realidad Aumentada del *tracking* por marcas, y siendo *ARToolKit* una biblioteca estándar con un extenso número de ejemplos desarrollados, se estableció ésta como método básico de registro y t*racking* absoluto.

Tras realizar este estudio del arte y tomar las primeras decisiones, se comenzó el desarrollo del proyecto.

Los requisitos establecidos en esta fase fueron:

1. El proyecto debe estar basado en Software Libre.
2. El desarrollo se debe hacer de forma modular y extensible.
3. Los tres subsistemas que forman el proyecto deben estar desacoplados, de forma que su mejora y extensión no debe requerir ningún cambio en el resto.
4. Debe ser posible la carga de elementos virtuales en capturas de vídeo obtenidas con una cámara.
5. El análisis de la iluminación de la escena ha de realizarse en tiempo real.

6.1.4. Diseño general

WILi fue planteado como un conjunto formado por tres subsistemas independientes. Estos subsistemas, a su vez estaban formados por módulos y submódulos desacoplados agrupados según el tipo de acciones que debían realizar o las bibliotecas que empleaban para ello.

Como principal patrón de Ingeniería del Software, se utilizó el *Singleton* para implementar los distintos controladores que presenta *WILi*.

Iteraciones

En esta sección se numeran las distintas iteraciones realizadas para desarrollar el Proyecto de Fin de Carrera, definiendo los requisitos de cada una, los problemas encontrados en el desarrollo de la misma, el nivel de complejidad y el esfuerzo empleado. Como se definió en la sección 4, estas iteraciones se corresponden con ciclos de la metodología de desarrollo del prototipado evolutivo.

Iteración 0

En esta iteración inicial se instalaron las herramientas y bibliotecas necesarias para el desarrollo del proyecto. Se generó también en esta iteración la estructura de directorios inicial que albergarían el proyecto.

Iteración 1

En esta etapa comenzó el desarrollo del subsistema de precálculo de iluminación.

- **Desarrollo** - Se implementó la clase `Baker` encargada de generar la escena virtual donde se simularía la iluminación sobre un objeto empleando la API de *Blender 2.55*

beta y de obtener las texturas que representasen mediante técnicas de *baking* la iluminación del objeto. Para determinar la posición de los focos virtuales, se empleó una esfera que inscribiese al objeto. Tomando los vértices de dicha esfera se podían posicionar focos de luz representados por coordenadas esféricas.

- **Pruebas** - Se compararon algunas de las texturas de salida muestras correctas previamente definidas. Se realizó una comprobación manual de la correctitud de la sombra proyectada en las mismas.
- **Entrega de versión** - Esta versión era capaz de insertar luces en la escena con unas coordenadas determinadas, además de ajustar algunas propiedades de la misma.
- **Análisis de nuevos requisitos** - Se detectaron algunos comportamientos anómalos en el prototipo, como que la fuente de luz empleada no determinaba de forma correcta las sombras ni la iluminación del objeto.

Iteración 2

En esta etapa se corrigieron los errores de la anterior iteración, además de soportar la proyección correcta de sombra.

- **Desarrollo** - Tras revisar el código exhaustivamente, se determinó que no había ningún error que pudiese generar los fallos del prototipo. Se procedió a cambiar la versión de *Blender 2.56 beta* que presentaba gran cantidad de correcciones, entre ellas en la API de *Python* empleada en *scripting*. Una vez incorporada la nueva versión, los fallos desaparecieron.

  En cuanto a la iluminación, se estableció por defecto una fuente de luz omnidireccional de tipo *Point* que representaba mejor la iluminación de una escena. Además, se implementó la proyección de sombras mediante un plano que presentase un material con la propiedad `only_shadow`. De esta forma, únicamente la sombra aparecería en la textura del plano asociado. En previsión de un posible requisito, se activó el canal *alpha* de esta textura para integrar la sombra en el entorno de una forma más natural.
- **Pruebas** - Se realizaron pruebas para comprobar que la sombra recogida por el plano tuviese una correspondencia con el plano virtual.
- **Entrega de versión** - El prototipo en esta iteración ya era capaz de reproducir las propiedades lumínicas de un objeto en distintos escenarios. La clase `Baker` del subsistema de precálculo de iluminación quedó prácticamente definida.
- **Análisis de nuevos requisitos** - Se planteó la necesidad de crear un exportador y un importador que siguiese el metaformato *Orej* para independizar el diseño de los módulos que conforman el sistema.

Iteración 3

En esta iteración se desarrolló el exportador e importador de *Orej*.

- **Desarrollo** - Se desarrolló el exportador teniendo en cuenta la nueva arquitectura de *Blender 2.50* y siguiendo el metaformato *Orej*. Para el importador se implementó un parser (`OrejParser`) que convirtiese el archivo *.orj* a objetos en memoria. Para dar soporte tanto al parser como al importador, se generaron una serie de clases que ayudasen a recoger toda la información necesaria para el despliegue del objeto. Se implementó la clase `Face` para dar soporte a las caras de los objetos. También se implementó la clase `Vertex`, que permitió el acceso y modificación de las coordenadas de un vértice y la clase `Object`, que fue desarrollado para almacenar la estructura del objeto, texturas y diversas propiedades del mismo. Además se empleó *ARToolKit*, *OpenGL* y *Glut* para generar un prototipo que permitiese probar la funcionalidad del exportador e importador. En este momento, el importador era capaz de dibujar el objeto virtual usando diferentes modos de despliegue, incluyendo texturas.

- **Pruebas** - Se comprobó que el despliegue de las texturas fuese el correcto. También se depuró el desarrollo del exportador y el importador mediante un *log* por consola. Se compararon los archivos *.orj* generados con otros archivos de los mismos modelos ya disponibles.

- **Entrega de versión** El prototipo en esta iteración era capaz de exportar un modelo 3D utilizando el exportador de *Orej*. El importador podía leer el fichero con ese formato, interpretarlo y desplegar el modelo en una aplicación de Realidad Aumentada usando *ARToolKit* para el registro de marcas y captura de vídeo, *OpenGL* para el despliegue de los objetos y *Glut* para el manejo de ventanas y eventos de teclado.

- **Análisis de nuevos requisitos** - Se determinó como siguiente requisito detectar la iluminación a partir de la sombra arrojada por un objeto conocido de la escena.

Iteración 4

En esta iteración se implementa la detección de la sombra que arroja un objeto conocido de la escena.

- **Desarrollo** - En primer lugar se estudiaron las alternativas sobre la forma geométrica que debía presentar dicho objeto para poder determinar el foco principal de la escena a partir de su sombra. Como conclusión de este estudio, se determinó que un cubo sería el objeto con el que sería más fácil analizar la escena.

  Para el análisis de los *frames* del vídeo capturado por la cámara se decidió utilizar *OpenCv*, la librería estándar en el desarrollo de aplicaciones de visión por computador

por el amplio soporte de dispositivos hardware que incluye en su distribución oficial. En esta etapa fue necesario adquirir conocimientos muy específicos basados en el tratamiento de imágenes para el análisis de las mismas, además de adquirir una importante base teórica relativa a la visión por computador.

Se decidió crear siguiendo el patrón de diseño *Singleton* un controlador que coordinase las acciones de la etapa de análisis (`WiliController`). Este controlador se comunicaría con los distintos módulos para ir formando soluciones parciales en el reconocimiento de la imagen.

Se implementó `ShadowAnalyzer`, encargado de realizar las distintas acciones para procesar la imagen, soportando el cambio a escala de grises, binarizado y búsqueda de contornos.

- **Pruebas** - Se generaron pruebas con imágenes previamente definidas como entrada para observar si el comportamiento era el adecuado. Además se probó usando un vídeo previamente grabado la detección de contornos.

- **Entrega de versión** - En esta iteración se obtuvo un prototipo capaz de detectar y dibujar sobre la imagen capturada por la cámara determinados contornos de la imagen interpretados como sombra.

- **Análisis de nuevos requisitos** - Tras desarrollar esta iteración, se descubrió que multitud de contornos eran detectados en la imagen como sombra, por lo que era necesario disponer de algún filtro que acotase el área de acción de búsqueda de la misma. Además, puesto que el análisis debía ser en tiempo real, sería conveniente recortar la imagen para disminuir su tamaño y, por tanto, el tiempo empleado en los cálculos por los algoritmos de *OpenCv*. También se observó que las caras del cubo localizadas en la zona iluminada indirectamente eran interpretadas como sombra, por lo que habría que construir otro filtro que eliminase esas caras de la imagen a analizar.

Iteración 5

En esta etapa se desarrollaron varios filtros para eliminar partes de la imagen consideradas como ruido y se procedió a obtener una subimagen del *frame*, recortándolo para obtener únicamente el área de proyección de la sombra.

- **Desarrollo** - En esta iteración se implementó un filtro eliminase la zona externa al área de proyección de la sombra.

  Se procedió a implementar un cubo virtual que, al ser desplegado, eliminase de la imagen las caras en umbría del cubo situado en la marca, haciéndolo coincidir con el mismo. De esta manera se obtendría una sombra limpia sin elementos externos que influyesen negativamente en su detección.

Dado que se estaba empleando *OpenCv* y el desplegado se hacía mediante *OpenGL*, hubo que determinar la forma de integrar lo que *OpenGl* dibujaba en una estructura `IplImage`. Para ello se accedió al *buffer* `GL_BACK` (doble *buffer*) de *OpenGL* especificando los píxeles que se querían leer del mismo y se volcó dicha información a un objeto `IplImage` de *OpenCv*.

También se decidió separar la parte de captura del prototipo y la del registro, ambas realizadas con *ARToolKit*. La finalidad era utilizar *ARToolKit* solo como método de registro y obtener las capturas de la cámara mediante *OpenCv*. Para realizar esto fue necesario revisar los archivos de *ARToolKit*, estudiando qué realizaba internamente cada función de la misma. También fue necesario redefinir en *OpenGl* el *frustrum* de la aplicación, ya que *ARToolKit* utiliza un convenio de proyección distinto.

- **Pruebas** - Se probaron distintas configuraciones con el foco de la escena real para ver si se detectaba la sombra correctamente empleando las ventanas de depuración. También se definieron un conjunto de trayectorias de prueba en el foco de iluminación principal para observar si la sombra era correctamente detectada.

- **Entrega de versión** - Al terminar esta iteración se obtuvo un prototipo que capturaba vídeo usando *OpenCv* y que empleaba únicamente *ARToolKit* para el registro de la marca. Además, una vez obtenida la imagen de entrada, aplicaba una serie de filtros para obtener únicamente el área de proyección de sombra, habiendo eliminado mediante el despliegue de un cubo virtual las caras en penumbra del cubo físico. Se añadió también soporte para la depuración de estos pasos mediante ventanas manejadas por *OpenCv*.

- **Análisis de nuevos requisitos** - Como siguientes requisitos se determinó la necesidad de localizar los vértices que intervienen en el cálculo del foco y estimación del mismo.

Iteración 6

En esta etapa obtuvieron los vértices que intervienen en el cálculo del foco así como la estimación de las coordenadas 3D del mismo.

- **Desarrollo** - En primer lugar, fue necesario reducir el número de puntos que formaban el contorno de la sombra. Se implementaron diversos algoritmos que conseguían reducir el número de vértices `CvPoint` presentes en un contorno `CvSeq` . Sin embargo, conforme evolucionaba el proyecto y se adquirían conocimientos de la biblioteca *OpenCv*, se descartaron dichos algoritmos por conseguir el mismo resultado modificando la entrada de otros ya empleados. Una vez determinada la sombra con un número mínimo de vértices, se procedió a implementar la detección del resto que estaban

implicados en el proceso de cálculo del foco. Para ello, primero había que determinar los puntos que entraban en contacto con la sombra y, a partir de ellos, determinar los vértices cuyas proyecciones coincidían con los de la sombra. Una vez encontrados estos, se podrían estimar también los vértices de la sombra generados por dicha proyección. A continuación, para calcular el foco, se minimizaría la distancia entre las rectas formadas por los vértices de la sombra y del cubo.

- **Pruebas** - Se definieron distintas posiciones con el foco siguiendo trayectorias predefinidas para comprobar que la estimación se hacía correctamente.

- **Entrega de versión** - Tras esta iteración se obtuvo un prototipo capaz de identificar el foco de la escena real mediante el análisis de la sombra arrojada por el cubo situado en la marca.

- **Análisis de nuevos requisitos** - Una vez establecido el foco, es necesario calcular la textura, de entre las existentes en la base de texturas precalculadas, que integra de forma más realista al objeto en la escena.

Iteración 7

En esta iteración se implementó la asignación de la textura al objeto a partir de la localización del foco.

- **Desarrollo** - Se creó el controlador `TextureController` que asigna al objeto la textura que más se ajuste a la iluminación de la escena real. Se implementó un algoritmo de búsqueda que calculaba, a partir de las coordenadas del foco y las de los focos empleados en la simulación de texturas, aquella que mejor representaba al objeto en la realidad. Para comunicar el subsistema de análisis con el de precálculo de la iluminación, se creó un formato específico que indicase las coordenadas de los focos empleados en el precálculo, lo que permitía calcular las texturas a partir de estas coordenadas. También se desarrolló un módulo que adaptaba la intensidad de las texturas para ajustarla a la intensidad de la iluminación de la escena.

  Finalmente se refactorizaron algunas clases para mejorar la eficiencia y la elegancia de la solución.

- **Pruebas** - Se realizaron las mismas pruebas que en la iteración anterior comprobando que las texturas cargadas correspondían con una iluminación realista del objeto.

- **Entrega de versión** - El prototipo de esta iteración era capaz de evaluar la escena y cargar objetos virtuales con una iluminación realista, satisfaciendo los requisitos iniciales del proyecto.

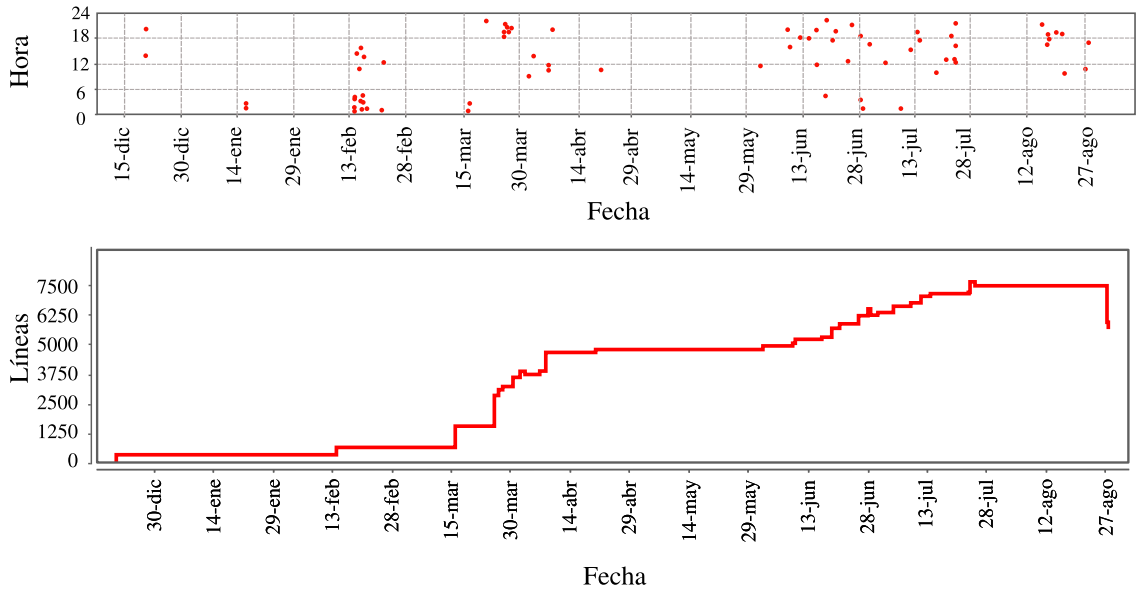

Figura 6.1: Gráfica de *commits* realizados y evolución de *WILi*. En la figura superior, se muestran los *commits* realizados organizados por fecha y hora. En la imagen inferior aparece la evolución de las líneas de código del proyecto.

6.1.5. Estadísticas del repositorio

En esta sección se describen algunas estadísticas relevantes del repositorio de *WILi*. Para ello se empleó *StatSVN*

En la Figura 6.1 se puede observar una gráfica de puntos que expresa el número de *commits* realizados, así como una estimación de la hora en la que se hicieron. En esta gráfica se puede observar que los principales periodos de desarrollo corresponden a Febrero, Marzo, Junio, Julio y Agosto.

En la misma Figura aparece también la evolución de líneas de código durante el desarrollo del proyecto. En ella se puede observar como en determinadas revisiones del código se eliminaron algunos algoritmos, por lo que la gráfica presenta en varios momentos decrecimientos.

En la tabla 6.1 puede observarse el número de líneas que tiene *WILi*, desglosadas por el lenguaje empleado y tipo de archivo analizado empleando la herramienta *cloc*.

6.2. Resultados

En esta sección se analizan algunos resultados sobre el tiempo de cómputo de *WILi* en distintas fases. Se ha intentado clasificar los resultados en función de aquellos que presentan dependencia con la geometría del modelo, representados en la tabla 6.2 y aquellos más relacionados con el análisis de la imagen y asignación de texturas mostrados en la tabla 6.3.

Lenguaje	Archivos	Espacios en blanco	Comentarios	Lineas de código
C++	31	1537	1590	3887
C/C++ Header	32	494	554	850
C	3	76	49	297
Python	5	121	78	230
make	1	10	1	18
Bourne Shell	2	3	0	4
Total	74	2261	2273	**5286**

Cuadro 6.1: Estadísticas del repositorio.

Para realizar las mediciones se tuvieron en cuenta tres modelos distintos:

- **Modelo 1 - Simple**. Modelo de 2462 caras.
- **Modelo 2 - Medio**. Modelo de 10150 caras.
- **Modelo 3 - Complejo**. Modelo de 69451 caras.

Los tiempos empleados por *WILi* para realizar un precálculo de la iluminación con una esfera de 12 segmentos y 10 anillos pueden verse en la tabla 6.2. De ella puede extraerse que a medida que se complica la geometría del modelo, el tiempo de cómputo de las etapas de precálculo, exportación e importación es mayor.

Funcionalidad	Modelo 1	Modelo 2	Modelo 3
Precálculo de la iluminación	196.28 s	209.4287 s	828.26 s
Exportación	0.11 s	0.49 s	4.40 s
Parseado del fichero Orej del plano	0.167 ms	0.141 ms	0.141 ms
Importación del plano	0.303 ms	0.26 ms	0.258 ms
Parseado del fichero Orej del objeto	33.364 ms	133.286 ms	875.971 ms
Importación del objeto	40.103 ms	161.975 ms	1081.19 ms

Cuadro 6.2: Tiempo de cómputo de acciones dependientes de la geometría.

En la siguiente tabla se muestran los tiempos de cómputo de diversas fases empleadas por *WILi*. Puede observarse que la gran mayoría de la funcionalidad está por debajo de 1 *ms*. El tiempo total requerido por las acciones independientes de la geometría es de 164.682 *ms*, lo que permite el despliegue del objeto en tiempo real (al menos 25 fps) incluso en computadores de bajas prestaciones. Los resultados pueden verse en la tabla 6.3.

El efecto visual que aporta *WILi* tras precalcular, analizar y asignar una textura a el objeto virtual puede verse en la Figura 6.2.

Funcionalidad	Tiempo (ms)
Filtros y recorte	16.92
Binarizado	3,642
Obtención del contorno de la sombra y reducción de vértices	0,978
Cálculo de vértices inferiores del cubo	0,306
Cálculo de vértices de la sombra	0,408
Cálculo del foco	12,852
Búsqueda de la mejor textura	0.162
Cálculo del factor de retoque	0,972
Procesamiento de *WILi*	36,24

Cuadro 6.3: Tiempo de cómputo de acciones independientes de la geometría.

En esta sección se han mostrado algunos datos de interés sobre el coste computacional de las funcionalidades más importantes de *WILi*. En la siguiente sección se hará una estimación de los costes humanos y materiales que han sido necesarios para desarrollar este Proyecto de Fin de Carrera.

6.3. Recursos y costes

En esta sección se determinan los recursos hardware , software y en recursos humanos empleados para el desarrollo de *WILi*, así como el coste que han supuesto económicamente.

6.3.1. Coste económico

El periodo de desarrollo se prolonga desde Noviembre del 2010 hasta Agosto del 2011. En estos 10 meses que ha durado el desarrollo se ha hecho una estimación de 820 horas invertidas [1] en el mismo de las cuales 320 corresponden al periodo que va desde Noviembre del 2010 a Mayo del 2011, en el que se compatibilizaba el proyecto con el curso académico y 480 horas correspondientes al periodo de Junio a Agosto del 2011, en el que se dedicaba entre 5 y 6 días por semana con una media diaria de 10 horas.

Para realizar la estimación de los recursos humanos, se tuvo en cuenta un precio orientativo de 24 €/hora (calculado a partir de ofertas en `http://www.infolancer.net`).

Los recursos empleados para el desarrollo de *WILi*, así como los costes que generan aparecen desglosados en la tabla 6.4.

[1] Esta estimación no tiene en cuenta el número de horas invertidas en la generación de la documentación del Proyecto de Fin de Carrera , que ha supuesto 220 horas aproximadamente

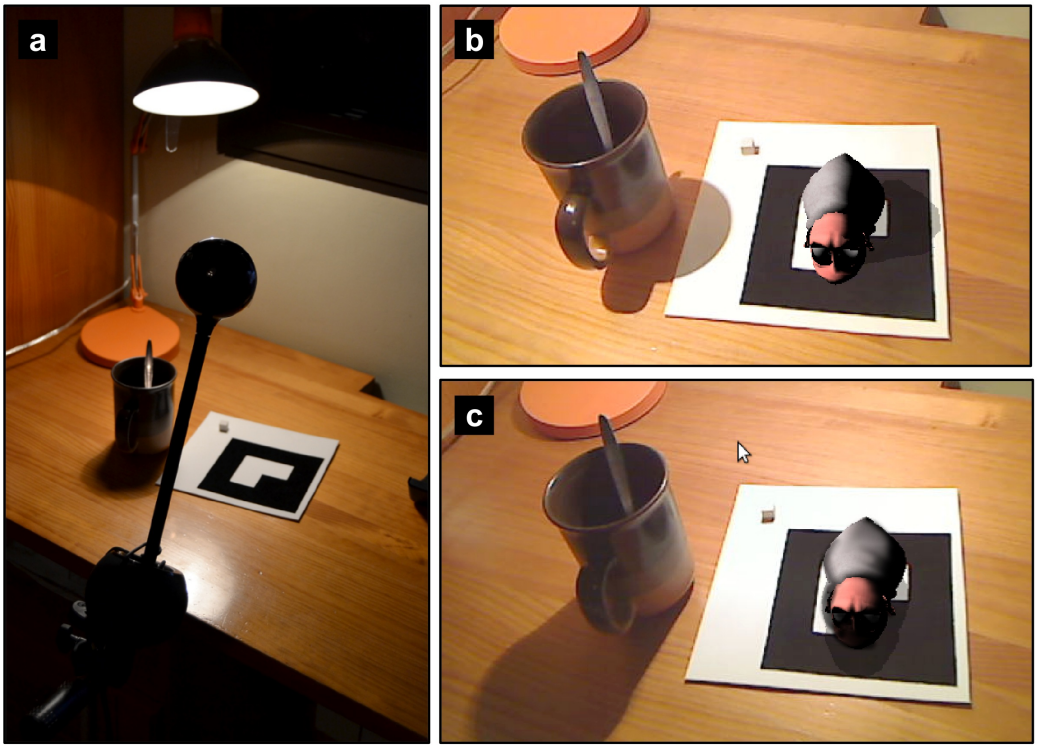

Figura 6.2: . Ejemplo de salida de *WILi* con una escena real: (a) Configuración del ejemplo, con la marca, un objeto real y una fuente de luz real direccional. Modificando la posición de la fuente de luz real en (b) y (c) se obtienen diferentes resultados en el objeto virtual. Nótese cómo la sombra proyectada del objeto virtual se ajusta perfectamente a la proyección de la sombra real de la taza en ambos casos.

Recurso	**Precio Unidad**	**Unidades**	**Total**
Equipo portátil	1100 €	1	1100 €
Webcam Logitech Sphere AF	117 €	1	117 €
Recursos Humanos	24 €/hora	820	19680 €
TOTAL			20897 €

Cuadro 6.4: Desglose económico del coste de *WILi*.

CAPÍTULO 7

CONCLUSIONES Y PROPUESTAS

En este capítulo se hace una valoración de los objetivos cumplidos. Además se definen futuras líneas de trabajo en las que se puede continuar trabajando para mejorar la funcionalidad de *WILi*. Por último, se hace una reflexión en base a los objetivos desarrollados y el aporte personal que han supuesto desarrollar el presente Proyecto de Fin de Carrera.

7.1. Objetivos alcanzados

Con el desarrollo de *WILi* se ha creado un prototipo funcional basado en dos subsistemas independientes que pretende abordar el problema de la iluminación adaptativa de objetos virtuales en entornos de Realidad Aumentada mediante el uso de métodos de síntesis de imagen realista.

Para ello, por un lado realiza un estudio de la iluminación en un entorno virtual, generando texturas que representan la iluminación que recibe un objeto cuando la luz de un foco con una configuración determinada incide sobre él.

El otro subsistema se encarga de estudiar la sombra arrojada por un elemento conocido y controlado de la escena para determinar la posición del foco y asignarle al objeto virtual la textura que más se ajuste para representar dicha iluminación. La versión actual del prototipo funcional se basa en el estudio de la sombra proyectada por un cubo cuyas dimensiones y posición relativa al centro de la marca son conocidas.

En todo momento se ha pretendido desarrollar un software que aisle al usuario de la complejidad de estos procedimientos basados en visión por computador y fundamentos matemáticos de la informática gráfica. De esta forma el usuario solo tiene que añadir un par de líneas a su código para desplegar el objeto que mejor se adapte a las condiciones de iluminación de la escena.

Los subsistemas han sido diseñados en módulos desacoplados, de forma que la modificación de uno no altera los resultados de otro. Por ejemplo, podría sustituirse *Blender* como suite de modelado empleado para la generación de las texturas precalculadas por otro software sin que el subsistema de análisis se vea afectado.

El subsistema de análisis presenta diversos módulos que pueden ser complementados en cualquier momento. El *Controlador* es el responsable de coordinar la secuencia de acciones que permiten detectar el foco y asignar al objeto virtual la textura correspondiente. Cada módulo funcional puede ser redefinido para mejorar las acciones que realiza.

El cálculo del umbral requiere especial atención para determinar qué parte de la imagen capturada corresponde a la sombra proyectada. En aplicaciones de visión por computador la principal dificultad suele recaer en el cálculo de este parámetro umbral. Se ha implementado un algoritmo que determina automáticamente un valor umbral adecuado para detectar convenientemente la zona de sombra.

A pesar de los errores inherentes al trabajo con imágenes de baja resolución y a los errores acumulados en la consecuente aplicación de filtros que producen una pérdida en la precisión del cálculo, se han conseguido buenos resultados, adaptando de forma correcta el objeto virtual al medio.

Los módulos de representación 2D y 3D permiten el dibujado de primitivas tanto bidimensionales como tridimensionales, además de desplegar objetos virtuales y cargar texturas.

El subsistema de análisis cuenta, además con un módulo que ayuda a la depuración visual, en el que algunas de las etapas del módulo de visión por computador son monitorizadas en ventanas individuales.

El módulo de gestión de texturas es el encargado de determinar la mejor textura que defina las propiedades lumínicas del objeto en la escena. Para ello emplea un submódulo de selección de texturas con un algoritmo que realiza una búsqueda entre las texturas disponibles. Además cuenta con otro submódulo encargado de corregir la intensidad de la iluminación en función de la intensidad del foco de la escena real.

Por último, todos los algoritmos y módulos desarrollados responden a la necesidad del software de realizar tanto el análisis como el despliegue del objeto en tiempo real, controlando los recursos que emplean y tratando de optimizar el uso de la CPU, delegando en la GPU algunas tareas de filtrado cuya ejecución es mucho más eficiente en este hardware dedicado.

7.2. Propuestas de trabajo futuro

Como se ha indicado en la sección anterior, los objetivos funcionales definidos inicialmente para el desarrollo de *WILi* se han completado satisfactoriamente. Sin embargo, durante la evolución del proyecto se han detectado diversas líneas de trabajo que pueden complementar al proyecto.

A continuación se citan estas propuestas de trabajo futuro.

- *Determinación de propiedades adicionales de las fuentes de luz y postprocesado del modelo*. Una de las principales líneas de trabajo futuro se orientan en la detección de parámetros adicionales de la fuente de luz (como intensidad, color, nivel de borrosidad de la sombra detectada, ...). Tras la detección de estos parámetros adicionales podría realizarse un postprocesado del modelo elegido para adaptar su textura al entorno de ejecución. De esta forma, el objeto virtual queda mejor integrado en escenas con una iluminación fuera de lo convencional. Para ello sería necesario crear un módulo que detectase los niveles de color del área de la sombra. Sabiendo que este color debe ser blanco, podría analizarse la imagen separando por canales y determinando si hay algún color que predomina en la imagen. En función de esto, puede ampliarse la clase `GLDrawer` para soportar la modificación de color, ampliar la clase `Object` para que sea capaz de almacenar estas modificaciones. Esta tarea tiene una complejidad asociada de nivel alto, con una carga de trabajo aproximada de 4 semanas.

- *Construcción de un entorno de pruebas determinista*. Quizás una de las líneas de trabajo abiertas para pasar de prototipo funcional desarrollado por una persona a un sistema mantenido por la comunidad sea la generación de un entorno de pruebas automatizado. Como se comentó en la sección 5.3.5, en el proceso realizado por *WILi* intervienen multitud de parámetros muy difíciles de controlar o simular. Las entradas de estos algoritmos son imágenes generadas bajo unas condiciones muy difíciles de simular con componentes aleatorios. La alternativa que más se ajusta a esta necesidad de probar el software podría ser el modelar una escena determinada de la realidad usando *Blender*. Empleando dicha escena y situando focos de luz con una intensidad y en posiciones que el software determine, se podrían renderizar dichas escenas obteniendo imágenes donde se conoce la posición del foco. Hecho esto, se podrían asignar como entrada del *subsistema de análisis* de *WILi* estas imágenes y, puesto que se conocen las posiciones de los focos para cada imagen, comparar la calidad de los resultados generados por *WILi*. En cualquier caso son pruebas que funcionarían para una escena determinada, siendo necesario crear una batería de pruebas que evalúen la correcta salida de cada componente del sistema.

  Esta tarea es de nivel de complejidad medio con una carga de trabajo estimada de 3-4 semanas.

- *Cálculo de la posición de la sombra con oclusiones*. Este es uno de los grandes retos de la solución aportada. *WILi* es capaz de localizar la posición del foco de luz analizando la sombra que arroja. El problema se presenta cuando se producen oclusiones por el propio cubo o cuando el foco de la luz principal está directamente sobre el objeto. En estas situaciones, la versión actual del prototipo no es capaz de determinar la posición del foco y utiliza la última configuración válida. Como alternativa se podrían analizar las caras del cubo buscando diferencias en la iluminación para hacer una estimación del foco de luz. También podría realizarse una estimación en base a un histórico en el que se viese la tendencia en el movimiento de la sombra. De esta forma, podría determinarse la posición del foco de luz siguiendo el patrón de movimiento de la sombra. Para ello podría emplearse un nuevo componente que fuese capaz de determinar, a partir de datos almacenados del foco, la posición que ocupa en un determinado momento, basándose en métodos de interpolación. El nivel de dificultad es alto con un tiempo de desarrollo aproximado de 3-4 semanas.

- *Mejora del algoritmo de búsqueda de la mejor textura*. La búsqueda de la mejor textura es calculada siguiendo un algoritmo descrito en 5.3.3. Los resultados obtenidos con él son bastante buenos, ya que elimina desde los primeros cálculos gran cantidad de candidatos, para trabajar finalmente en una búsqueda con muy pocos elementos. Este algoritmo usado podría mejorarse siguiendo técnicas de búsqueda basadas en estructuras de datos del tipo *B-tree* que contribuirían a mejorar los tiempos de búsqueda. La dificultad para realizar esta mejora complejidad media con un tiempo estimado de 1-2 semanas.

- *Soporte multifoco*. *WILi* desde sus inicios fue definido como un prototipo que trabajaría con una única sombra para realizar los cálculos de un único foco de luz. Una mejora muy interesante sería la investigación de métodos de detección multifoco, en el que intervienen luces con distintas intensidades y colores y, por lo tanto, con distintas sombras. Posiblemente el desarrollo de este módulo requiera de hacer un análisis por capas, donde cada capa está caracterizada por un nivel de intensidad en la sombra. Esto determinaría de forma iterativa los distintos focos de luz que forman la escena para concluir en una etapa de composición de la escena final. El nivel de complejidad es muy alto debido a la dificultad en la detección adecuada de regiones con con niveles de color (típicamente gris) muy cercanos. Esta tarea presenta un período de desarrollo estimado de 3-6 meses.

- *Mejora del proceso de cálculo del umbral*. Esta propuesta consistiría en buscar e implementar algún algoritmo alternativo para detectar con mayor precisión la sombra arrojada por el cubo. Pueden usarse tanto técnicas que busquen el umbral necesario para acotarla, como algoritmos de detección empleando la reducción de colores

que consigan obtener directamente el contorno de la sombra. Esta función sustituiría a `getShadow()` de la clase `ShadowAnalizer`. Esta labor presenta una complejidad alta, ya que habría que determinar un nuevo algoritmo que condiciona la funcionalidad del programa. El tiempo estimado para esta mejora es de 2-4 semanas.

- *Mejora del exportador e importador de objetos 3D*. Tanto el exportador como el importador fueron mejorados con respecto a la versión inicial del formato *Orej*, tal y como se comentó en la sección 5.2.2. A pesar de ello, sería interesante continuar la mejora de la implementación para que soporte:

  - Animación de los objetos virtuales. Para ello podría utilizarse como referencia el exportador e importador original. En el caso del exportador habría que extenderlo implementando en *Python* usando la biblioteca *bpy*. En cuanto al importador, habría que añadir dicha funcionalidad en C++ a la clase `OrejParser` y `OrejImporter`.
  - Nuevas formas de representación. Se pueden añadir nuevas formas de representación modificando la clase `GLDrawer`, añadiendo a la función `deployObject()` los nuevos modos de despliegue (con soporte para *shaders* o eliminación de la geometría no visible en modo `wireframe`).
  - Tratamiento de errores. Añadiendo el tratamiento de errores mediante interrupciones en C++.

  La complejidad de estos cambios es de nivel bajo con una carga de trabajo de 1 semana.

- *Difusión del proyecto*. La realización del proyecto supuso el estudio de diversas bibliotecas de informática gráfica y visión por computador. Para realizar diversos módulos que solucionasen tareas muy específicas fue necesario desarrollar varias versiones de un mismo componente hasta conseguir el resultado esperado. Esto es debido a que no se encontraban referencias adecuadas para resolver los problemas específicos que surgían en el desarrollo del proyecto.

  Ante esto es necesario la creación de un sitio web donde se defina, no solo la finalidad del proyecto y la difusión de su contenido, sino además, la solución a los problemas encontrados para proporcionar soporte a futuros desarrolladores. La dificultad de esta tarea es baja con un tiempo aproximado de desarrollo de 1-2 semanas.

7.3. Conclusión personal

El desarrollo de un Proyecto de Fin de Carrera multidisciplinar contribuye enormemente al desarrollo del ingeniero. Durante los años invertidos en la carrera se han estudiado diferentes metodologías, lenguajes, técnicas y toda una base teórica que conforman los conocimientos

necesarios para comprender la Ingeniería Informática. El Proyecto de Fin de Carrera brinda la oportunidad de unir todos estos conocimientos para conseguir un resultado eficiente y robusto.

Sin embargo, la Ingeniería Informática está caracterizada por tratar multitud de ramas o especialidades, algunas de ella no contempladas en el plan de estudios. Es por ello que este Proyecto de Fin de Carrera ha sido desarrollado pensando más en abordar esas especialidades que cuentan con poca carga docente actualmente, así como en aprender nuevos lenguajes para complementar la formación recibida.

También supone un reto personal que ayuda a madurar el perfil profesional del ingeniero, no solo por complementar sus conocimientos y "saber un poco de todo", sino por ser capaz de abordar temas desconocidos para el estudiante de ingeniería. En el mercado laboral, un ingeniero tiene que ser capaz de adaptarse a las condiciones, exigencias y requisitos de un proyecto para poder abordarlo con éxito. Tratar un proyecto multidisciplinar con herramientas y lenguajes desconocidos por el estudiante le pone a prueba como futuro ingeniero.

Con este proyecto se han adquirido, entre otros, conocimientos de visión por computador, modelado 3D, informática gráfica, *scripting* y realidad aumentada. Se ha realizado una ampliación de la base matemática y se ha obtenido una base teórica importante de todas las especialidades anteriormente citadas. Como lenguaje aprendido destaca C++, un lenguaje robusto y potente.

Además, las reuniones con el director de proyecto, la aplicación de técnicas propias de Ingeniería del Software, la toma de decisiones y la resolución de problemas que comprometen al desarrollo del proyecto son esenciales para completar la formación.

El desarrollo de *WILi* no pretende conseguir un producto comercial ni una solución definitiva al problema de la iluminación adaptativa de objetos virtuales. *WILi* ha sido concebido como una alternativa, como una manera de ver y tratar este problema, pero no tiene por que ser la única y mejor forma de resolverlo.

Todo el proyecto ha sido desarrollado bajo una licencia GPLv3 (ver anexo D) para distribuir el conocimiento entre el resto de gente y contribuir a acelerar los avances que están por venir. De esta manera, el resto de profesionales que estén investigando cómo resolver el problema de la iluminación adaptativa, pueden ver un prototipo desarrollado y con un estudio ya realizado que presenta sus ventajas y desventajas contabilizadas y definidas. Esto les da la libertad de coger, bajo los términos de la licencia, partes de *WILi* que consideren relevantes, redefinirlas para hacerlas más eficientes o resolver problemas no contemplados e incluso descartarlas por tener alternativas mejores.

En definitiva, pretende contribuir al desarrollo de la mejor solución en el ámbito de la iluminación adaptativa, compartiendo con cualquier persona los conocimientos adquiridos durante la realización del presente Proyecto de Fin de Carrera.

ANEXOS

ANEXO A

CONSTRUCCIÓN DE MARCAS

El objetivo de este anexo es explicar cómo construir las marcas de *WILi-ARToolKit*.

Se hará una descripción de las marcas, así como los materiales necesarios para crearlas.

A.1. Las marcas

Las marcas de *ARToolKit* sirven para establecer el origen de coordenadas del mundo real. Se caracterizan por presentar un cuadrado negro con un dibujo blanco en su interior como puede verse en la figura A.1. El tamaño de la marca contribuye a la facilidad y distancia con la que es detectada. Cuanto mayor sea la marca, mayor será la distancia desde la que *ARToolKit* la detectará.

Para garantizar la detección, se ha optado por trabajar con el tamaño estándar de marcas definido por la biblioteca de 12 cm.

A.2. Los materiales

Normalmente las marcas de *ARToolKit* son impresas en papel o cartulina blanca y pegadas sobre una superficie lisa como puede ser el cartón. En el caso de utilizar una marca con *WILi* hay que tener en cuenta una serie de recomendaciones adicionales. A continuación se listan los diversos aspectos que deben considerarse a la hora de trabajar con las marcas en *WILi*.

- **El soporte**. Como se indicó anteriormente, la base comúnmente empleada suele ser el cartón. Sin embargo este ligero soporte tiende a curvarse por la humedad ambiental y el uso. Esto puede generar una marca que no es plana, alterando las coordenadas de los objetos virtuales. *WILi* emplea objetos virtuales a modo de filtros en la pantalla, por lo que este tipo de distorsiones puede alterar el correcto funcionamiento del mismo. Se recomienda el uso de planchas de madera o *dm*[1] para utilizar como base.

- **La base**. Normalmente la marca es impresa sobre papel o cartulina. Sin embargo estas bases provocan que la luz brille en la misma. Para evitar falsas detecciones es recomendable utilizar como base *Goma EVA*[2] de color blanco, que presenta una textura con una rugosidad prácticamente inapreciable pero que contribuye a dispersar la luz.

- **La figura**. La tinta negra suele brillar cuando recibe la luz directa de un foco. Esto puede causar que el brillo impida a *ARToolKit* detectar la marca. Para realizar la figura negra puede emplearse, bien terciopelo autoadhesivo o goma EVA de color negro.

- **El cubo**. Para la construcción del cubo se empleó la impresora 3D del laboratorio Oreto. Dicha impresora construye modelos superponiendo capas de adhesivo y material base similar a la escayola . También podrían emplearse recortes de perfiles cuadrados de madera u otro material para construir el cubo, teniendo en cuenta que es necesario contar con una precisión alta a la hora de cortarlos. Podría emplearse incluso un dado si éste presenta las esquinas bien definidas.

A.3. Disposición del cubo en la marca

El cubo es sin duda el elemento más importante de la marca. Una mala colocación del cubo puede derivar en malos resultados a la hora de establecer el foco. Aunque en la figura presenta una posición determinada, puede situarse en cualquier posición, y luego modificar el archivo `WILiConstants.h` como se explica en el anexo B.

El cubo del prototipo presenta unas medidas de 1cm de lado. Incluso con este pequeño tamaño se han obtenido buenos resultados. Sin duda, a mayor tamaño del cubo, mejor detección de la sombra proyectada.

Hay que prestar especial interés en la colocación del cubo, evitando que quede rotado en la posición dada, evitando así que *WILi* estime de forma incorrecta la posición del cubo.

La marca es una pieza fundamental en el correcto funcionamiento de *WILi*. La correcta construcción de la misma garantiza mejores resultados en la etapa de detección de la sombra y posterior asignación de la iluminación virtual.

[1] El dm consiste en un material consistente en un aglomerado de fibras de madera de densidad media y resina

[2] El nombre químico de este compuesto es el Etileno Vinil Acetato, de ahí las siglas que conforman su nombre

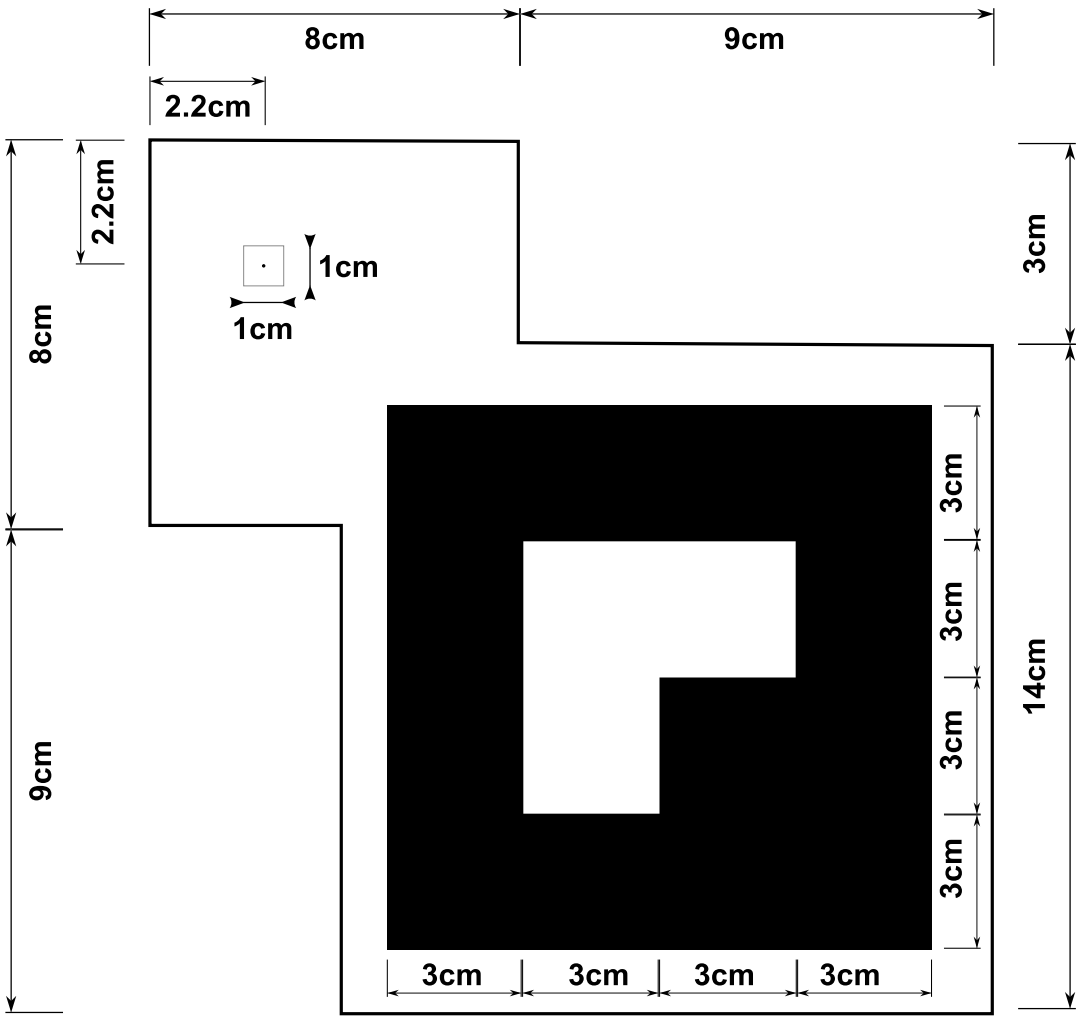

Figura A.1: Medidas utilizadas en la construcción de la marca del prototipo empleado para el desarrollo de *WILi*.

ANEXO B

CONSTANTES

En este anexo se definen las constantes empleadas en Wili. A continuación se muestran dichas constantes organizadas según el subsistema que las utiliza.

B.1. Subsistema de precálculo de iluminación

B.1.1. Constantes de los focos de luz

En el cuadro B.1 se muestran las constantes que definen las propiedades de los focos insertados en la escena.

Constantes	Descripción	Recomendaciones
`LAMP_TYPE`	Especifica el tipo de foco	Se recomienda no alterar este valor. Corresponde a los tipos de focos soportados por Blender
`ENERGY`	Indica la intensidad de iluminación del foco	-
`DISTANCE`	Indica la distancia máxima a la que llega la luz del foco	-

Cuadro B.1: Constantes de los focos de luz

B.1.2. Constantes de rutas

En las constantes del cuadro B.2 se definen las rutas donde se van a guardar los resultados obtenidos. *Es preferible no modificarlas*.

Constantes	Descripción	Recomendaciones
`PATH`	Especifica la ruta padre	Se recomienda no alterar este valor
`output_blender_path`	Indica la ruta en la que se guardará el fichero de Blender de depuración	Se recomienda no alterar este valor
`output_textures_path`	Indica la ruta en la que se guardarán las texturas	Se recomienda no alterar este valor
`CONFIG_PATH`	Indica la ruta completa del fichero de configuración	Se recomienda no alterar este valor

Cuadro B.2: Constantes de rutas

B.1.3. Constantes de nombres de objetos

En el cuadro B.3 se indican las constantes que definen de forma interna el nombre de los objetos con los que trabaja. *No hay que modificarlas bajo ningún concepto*.

Constantes	Descripción	Recomendaciones
`OBJECT_NAME`	Especifica el nombre interno que identifica al modelo	Este valor no debe ser alterado en ningún caso
`OBJECT_SUBNAME`	Indica la letra con la que es identificado el modelo	Este valor no debe ser alterado en ningún caso
`PLANE_NAME`	Especifica el nombre interno que identifica al plano que recoge las sombras modelo	Este valor no debe ser alterado en ningún caso
`PLANE_SUBNAME`	Indica la letra con la que es identificado el plano donde se recogen las sombras del modelo	Este valor no debe ser alterado en ningún caso

Cuadro B.3: Constantes de nombres de objeto

B.1.4. Constantes de la esfera

Las constantes del cuadro B.4 definen la esfera. A excepción de `SPHERE_SEGMENTS` y `SPHERE_RING_COUNT`, el resto de constantes no hay que modificarlas bajo ningún concepto.

Constantes	Descripción	Recomendaciones
SPHERE_NAME	Especifica el nombre interno por el que se identifica la esfera	Este valor no debe ser alterado en ningún caso
SPHERE_SEGMENTS	Indica el número de segmentos de la esfera	Este valor puede modificarse para aumentar o reducir la precisión
SPHERE_RING_COUNT	Indica el número de anillos de la esfera	Este valor puede modificarse para aumentar o reducir la precisión
SPHERE_SIZE	Indica el tamaño de la esfera	Se recomienda no alterar este valor

Cuadro B.4: Constantes de la esfera

B.1.5. Otras constantes

Se define en el cuadro B.5 algunas constantes que no presentan una clasificación especial. Es importante remarcar que *no hay que modificarlas bajo ningún concepto*.

Constantes	Descripción	Recomendaciones
PLANE_SIZE	Especifica el tamaño de los lados del plano en centímetros	Se recomienda no alterar este valor
TEXTURE_XY	Indica en píxeles la longitud de los lados de la textura	Se recomienda no alterar este valor

Cuadro B.5: Otras constantes del subsistema de precálculo de iluminación

B.1.6. Constantes del fichero Baker.py

Ejemplo de las constantes incluidas en el fichero `Baker.py` .

```
LAMP_TYPE = 'POINT'
ENERGY = 6
DISTANCE = 20

PATH = '../' # Don't forget '/' in the end
output_blender_path = PATH + 'baker-results/' # Don't forget '/' in the end
output_textures_path = PATH + 'baker-textures/' # Don't forget '/' in the end
CONFIG_PATH = output_textures_path + "tex.cfg"

OBJECT_NAME = 'main_object'
OBJECT_SUBNAME = 'o'
PLANE_NAME = 'main_plane'
PLANE_SUBNAME ='p'

SPHERE_NAME = 'main_sphere'
SPHERE_SEGMENTS = 12
SPHERE_RING_COUNT = 4
SPHERE_SIZE = 9 #3

PLANE_SIZE = 10
TEXTURE_XY = 1024
```

B.2. Subsistema de análisis

Se describirán las constantes empleadas de dicho subsistema. En la figura B.1 se puede apreciar de forma gráfica lo que representan algunas de estas constantes.

B.2.1. Constantes del cubo

El cubo presenta constantes que definen su tamaño, posición y escalado tal y como se aprecia en el cuadro B.6.

B.2.2. Constantes del área de proyección de la sombra

La sombra del cubo se proyecta sobre un área cuadrada conocida. Es necesario por tanto especificar las coordenadas tridimensionales de los vértices que la componen usando las constantes del cuadro B.7.

B.2.3. Otras constantes

En el cuadro B.8 se muestran otras constantes que no presentan ninguna clasificación especial.

Constantes	**Descripción**	**Recomendaciones**
CUBE_WIDTH	Especifica en milímetros el tamaño del cubo	Se recomienda no alterar este valor
CUBE_SCALE	Especifica el escalado que debe realizarse al cubo	Se recomienda no alterar este valor
CUBE_X_POSITION	Indica la posición en el eje X con respecto al centro de la marca donde está situado el centro del cubo	En el caso de cambiar la posición del cubo, establecer en este parámetro la nueva coordenada X.
CUBE_Y_POSITION	Indica la posición en el eje Y con respecto al centro de la marca donde está situado el centro del cubo	En el caso de cambiar la posición del cubo, establecer en este parámetro la nueva coordenada Y.
CUBE_Z_POSITION	Indica la posición en el eje Z con respecto al centro de la marca donde está situado el centro del cubo	Dado que el cubo descansa sobre la marca y el centro del mismo se situó en la base, la posición en el eje Z debe ser igual a 0. Se recomienda no alterar este valor

Cuadro B.6: Constantes del cubo

Constantes	Descripción	Recomendaciones
LEFT_TOP_X	Coordenada X del vértice superior izquierdo del área de proyección de sombra	-
LEFT_TOP_Y	Coordenada Y del vértice superior izquierdo del área de proyección de sombra	-
LEFT_TOP_Z	Coordenada Z del vértice superior izquierdo del área de proyección de sombra	-
RIGHT_TOP_X	Coordenada X del vértice superior derecho del área de proyección de sombra	-
RIGHT_TOP_Y	Coordenada Y del vértice superior derecho del área de proyección de sombra	-
RIGHT_TOP_Z	Coordenada Z del vértice superior derecho del área de proyección de sombra	-
RIGHT_BOTTOM_X	Coordenada X del vértice inferior derecho del área de proyección de sombra	-
RIGHT_BOTTOM_Y	Coordenada Y del vértice inferior derecho del área de proyección de sombra	-
RIGHT_BOTTOM_Z	Coordenada Z del vértice inferior derecho del área de proyección de sombra	-
LEFT_BOTTOM_X	Coordenada X del vértice inferior izquierdo del área de proyección de sombra	-
LEFT_BOTTOM_Y	Coordenada Y del vértice inferior izquierdo del área de proyección de sombra	-
LEFT_BOTTOM_Z	Coordenada Z del vértice inferior izquierdo del área de proyección de sombra	-

Cuadro B.7: Constantes del área de proyección de la sombra

Constantes	Descripción	Recomendaciones
`INF`	Especifica el valor de infinito	Se recomienda no alterar este valor
`WILI_TEX_PATH`	Especifica la ruta donde se encuentran las texturas en el sistema	Modificar este valor sin olvidar la barra al final de la ruta. La ruta debe ir entrecomillada

Cuadro B.8: Otras constantes del subsistema de análisis

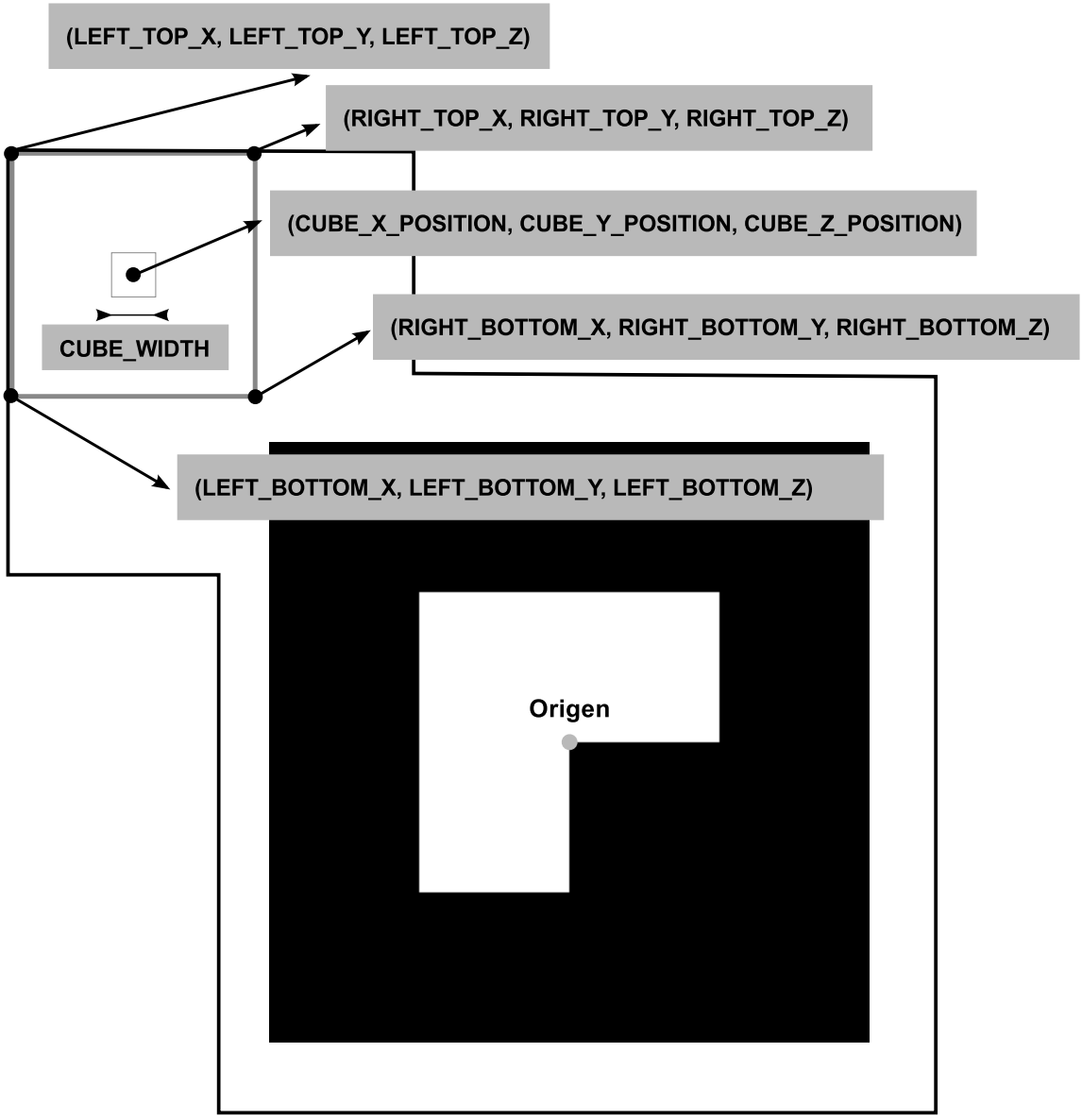

Figura B.1: Constantes que representan elementos de la marca.

B.2.4. Fichero WiliConstants.h

Ejemplo de las constantes incluidas en el fichero `WiliConstans.h`.

```
1  /* Wili Cube Values*/
2
3  #define CUBE_WIDTH               10     /* Width in mm*/
4  #define CUBE_SCALE                1
5  #define CUBE_X_POSITION         -80
6  #define CUBE_Y_POSITION          80
7  #define CUBE_Z_POSITION           0
8
9  /*Shadow Area*/
10
11 #define LEFT_TOP_X              -98
12 #define LEFT_TOP_Y              100
13 #define LEFT_TOP_Z                0
14
15 #define RIGHT_TOP_X             -60
16 #define RIGHT_TOP_Y             100
17 #define RIGHT_TOP_Z             0
18
19 #define RIGHT_BOTTOM_X          -60
20 #define RIGHT_BOTTOM_Y           60
21 #define RIGHT_BOTTOM_Z            0
22
23 #define LEFT_BOTTOM_X           -98
24 #define LEFT_BOTTOM_Y            60
25 #define LEFT_BOTTOM_Z             0
26
27 #define INF                  999999
28
29 #define WILI_TEX_PATH        "../baker-textures/"
```

ANEXO C

CÓDIGO FUENTE

Debido a la extensión del código fuente (unas 5000 líneas), se incluirá el código fuente en su versión electrónica en el CD adjunto a este documento.

Los directorios en los que se estructura dicho código se muestra a continuación:

- **/Baking:** contiene aquellas clases correspondientes al precálculo de la iluminación.
- **/Analysis:** contiene aquellas clases que intervienen en el análisis de la imagen.
  - **/MathObjects:** contiene los objetos usados en operaciones matemáticas.
  - **/Controllers:** contiene la implementación de los controladores.
  - **/Objects:** contiene los objetos empleados por *WILi* en el análisis de la imagen.
- **/Orej:** contiene diferentes clases para el soporte de *Orej*
  - **/Importer:** contiene la implementación del importador de *Orej*.
  - **/Exporter:** contiene la implementación del exportador de *Orej*.

ANEXO D

GNU GENERAL PUBLIC LICENSE
VERSION 3, 29 JUNE 2007

PREAMBLE

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

   b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

   c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

   d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

   a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

   b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

   c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

   d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

   e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

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, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights

granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

    The Free Software Foundation may publish revised and/or new versions of the GNU General Public 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.

    Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

    If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

    Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <textyear>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program>  Copyright (C) <year>  <name of author>

This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `http://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `http://www.gnu.org/philosophy/why-not-lgpl.html`.

Bibliografía

[A+97] R.T. Azuma et al. A survey of augmented reality. *Presence-Teleoperators and Virtual Environments*, 6(4):355–385, 1997.

[ABB+01] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, y B. MacIntyre. Recent advances in augmented reality. *Computer Graphics and Applications, IEEE*, 21(6):34–47, 2001.

[AMHH08] T. Akenine-Möller, E. Haines, y N. Hoffman. *Real-time rendering*. AK Peters, 2008.

[BB11] M. Bartaripou y H.R. Babae. A Survey on Web-based AR Applications. *International Journal of Computer Science Issues*, 8(1), Junio 2011.

[BHF+99] A. Butz, T. Höllerer, S. Feiner, B. MacIntyre, y C. Beshers. Enveloping users and computers in a collaborative 3D augmented reality. En *iwar*, página 35. Published by the IEEE Computer Society, 1999.

[BK02] M. Billinghurst y H. Kato. Collaborative augmented reality. *Communications of the ACM*, 45(7):64–70, 2002.

[BK08] G. Bradski y A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, 2008.

[BO01] J. Branch y G. Olague. La visión por computador: Una aproximación al estado del arte. *Revista Dyna (133)*, 2001.

[CP11] M. Calle y F. Pulido Galán. Realidad aumentada con servicios OGC implementada con librerías de fuentes abiertas. En *V Jornadas de SIG Libre*. Universitat de Girona. Servei de Sistemes d'Informació Geogràfica i Teledetecció, Marzo 2011.

[dI07] Escuela Superior de Informática. *Normativa del Proyecto de Fin de Carrera*. Universidad de Castilla-La Mancha, 2007. url: `http:`

//www.esi.uclm.es:8081/www/documentos/Normativas/NormativaPFC2007.pdf.

[ERF07] P. Eisert, J. Rurainsky, y P. Fechteler. Virtual mirror: Real-time tracking of shoes in augmented reality environments. En *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, volume 2, páginas II–557. IEEE, 2007.

[EZ11] M. El-Zayat. Augmented Reality platform for enhancing integration of virtual objects. En *Proceedings of CESCG 2011: The 15th Central European Seminar on Computer Graphics*, 2011.

[Fer03] J.A. Ferwerda. Three varieties of realism in computer graphics. En *Proceedings SPIE Human Vision and Electronic Imaging*, volume 3. Citeseer, 2003.

[FJS01] W. Friedrich, D. Jahn, y L. Schmidt. ARVIKA-augmented reality for development, production and service. En *Proceedings of the International Status Conference HCI*, volume 3, página 34, 2001.

[Gei05] S. Geisen. Augmented Reality in Surgery. En *Seminar : Medical Images*, 2005.

[HHD00] T. Horprasert, D. Harwood, y L.S. Davis. A robust background subtraction and shadow detection. En *Proc. ACCV*, páginas 983–988. Citeseer, 2000.

[Hol11] W. Holden. Mobile Augmented Reality : Opportunities, Forecasts & Strategic Analysis 2011-2015. Technical report, Juniper Research, feb 2011.

[Joh10] L. Johnston. Web Reviews: Augmented Reality. *Sci-Tech News*, 64(1):10, 2010.

[Kar11] T. Karppi. Working Papers 2010. *Unfolding Media Studies*, página 93, 2011.

[LRJ+02] M.A. Livingston, L.J. Rosenblum, S.J. Julier, D. Brown, Y. Baillot, J.E.S. II, J.L. Gabbard, y D. Hix. An augmented reality system for military operations in urban terrain. En *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*. NTSA, 2002.

[Mad09] J. Madden. Augmented Reality : Adding Information to Our View of the World. Technical report, ABI Research, 2009.

[ML10] Madsen y Lal. *Probeless Illumination Estimation for Outdoor Augmented Reality*. InTech, Enero 2010.

[NS10] Z. Noh y M.S. Sunar. Soft Shadow Rendering based on Real Light Source Estimation in Augmented Reality. *Advances in Multimedia-An International Journal (AMIJ)*, 1(2):26, 2010.

[PT03] W. Piekarski y B.H. Thomas. ARQuake-Modifications and hardware for outdoor augmented reality gaming. En *4th Australian Linux Conference*. Citeseer, 2003.

[RBBS06] B. Reitinger, A. Bornik, R. Beichel, y D. Schmalstieg. Liver surgery planning using virtual reality. *Computer Graphics and Applications, IEEE*, 26(6):36–47, 2006.

[Rid96] D.F. Riddle. *Geometría analítica*. International Thomson Editores, 1996.

[RS04] T. Roosendaal y S. Selleri. *The Official Blender 2.3 guide: free 3D creation suite for modeling, animation, and rendering*. No Starch Press, 2004.

[SD03] B. Schwald y B. De Laval. An augmented reality system for training and assistance to maintenance in the industrial context. *Journal of Winter School of Computer Graphics (WSCG)*, 11(1):425–432, 2003.

[SE03] P.J. Schneider y D.H. Eberly. *Geometric tools for computer graphics*. Morgan Kaufmann Pub, 2003.

[Shr10] D. Shreiner. *OpenGL programming guide*. Addison-Wesley Reading, MA, 2010.

[TCD+00] B. Thomas, B. Close, J. Donoghue, J. Squires, P. De Bondi, M. Morris, y W. Piekarski. ARQuake: An outdoor/indoor augmented reality first person application. En *International Semantic Web Conference (ISWC)*, página 139. Published by the IEEE Computer Society, 2000.

[TPPP08] T. Theoharis, G. Papaioannou, N. Platis, y N.M. Patrikalakis. *Graphics & visualization: principles & algorithms*. AK Peters Ltd, 2008.

[TS10] C.T. Tan y D. Soh. Augmented Reality Games: A Review. *Proceedings of GAMEON-ARABIA, EUROSIS*, 2010.