



*ugr* | Universidad  
de **Granada**

TRABAJO FIN DE GRADO  
INGENIERÍA INFORMÁTICA

## 3DPhotoModeling

---

Modelado interactivo para reconstrucción de objetos 3D  
basado en imágenes

**Autor**

Juan Pablo Porcel Porcel

**Tutor**

Pedro Cano Olivares



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIÓN

---

Granada, 12 de diciembre de 2016



# **3DPhotoModeling: Modelado interactivo para reconstrucción de objetos 3D basado en imágenes**

Juan Pablo Porcel Porcel

**Palabras clave:** informática gráfica, modelado en tres dimensiones, reconstrucción 3D, imagen 2D, OpenGL, OpenCV, software libre

## **Resumen**

En el presente trabajo, el lector podrá encontrar cómo se ha desarrollado un sistema de modelado interactivo de objetos tridimensionales a partir de una foto. Con este sistema se pretende solucionar el problema de generar un modelo 3D que aparece en una foto de manera sencilla para poder trabajar con estos modelos en otro software.

La capacidad de extraer objetos tridimensionales a partir de una única imagen ha sido caso de estudio durante mucho tiempo, este proceso involucra numerosas tareas complejas como la estimación de la pose del objeto o el desconocimiento de los parámetros intrínsecos y extrínsecos de la cámara que capturó la imagen.

En este documento se introduce una técnica de manipulación sencilla de formas tridimensionales basadas en objetos extraídos de una sola fotografía. Esta extracción requiere entender los componentes de la figura, su proyección, su geometría y su textura. Estas tareas cognitivas son muy simples para los humanos pero son particularmente complejas para ser implementadas en algoritmos automáticos. La técnica descrita en este documento combina las habilidades cognitivas de los humanos con la capacidad de cómputo de los ordenadores para resolver este problema.

El sistema que se presenta da al usuario la posibilidad de crear objetos en tres dimensiones a partir de una imagen dibujando sobre ella. El sistema transforma la entrada del usuario en datos útiles para recrear el modelo 3D, extrayendo su textura de la imagen junto con su geometría.

Para la implementación de la aplicación se han usado las librerías de alto nivel OpenGL, OpenCV y el framework Qt para crear interfaces gráficas de usuario en C++. Se trata de una aplicación de escritorio con una arquitectura Modelo-Vista-Controlador.

Una vez creada la base de la aplicación es posible incluir nuevas funcionalidades en el futuro. Como puede ser el caso de modelar objetos con

formas complejas.

En la Figura 1 se pueden ver los resultados que pueden conseguirse con el software.



Figura 1: Reconstrucción 3D de un objeto. A la izquierda: la imagen de base que el usuario carga en la aplicación. A la derecha: la imagen resultante tras generar el modelo y realizar transformaciones sobre este (escalar y trasladar).

Los modelos generados con la aplicación 3DPhotoModeling pueden ser exportados en formato PLY para poder ser importados en otro software de visualización o modelado en 3D para trabajar sobre ellos.

# **3DPhotoModeling: Interactive modeling for the reconstruction of 3D objects with images**

Juan Pablo Porcel Porcel

**Keywords:** computer graphics, 3D modeling, object-based reconstruction, 2D image, OpenGL, OpenCV, open source

## **Abstract**

In this work, the reader will be able to find the process of develop of a interactive system for modeling 3D objects from a single photo. Whith this application is intended to solve the problem of generating a 3D model that appears in a photo in a simple way for work with these models in other software.

The ability to extract 3D objects from a single image is a long case of research. This process involves numerous complex tasks, such as estimating the object's pose or ignoring the intrinsic and extrinsic parameters of the camera that captured the object image.

This document introduces an interactive technique for manipulating simple 3D shapes based on extracting them from a single photo. This process of extraction requires understanding of the components of the shape, its projection, geometry and texture. These simple cognitive tasks for humans are particularly difficult for automatic algorithms. The technique described combines the cognitive abilities of humans with the computational accuracy of the machine to solve this problem.

The system that is presented gives the user the possibility of creating objects in 3D from a single image drawing on it. The system transforms the user's input into useful data to recreate the 3D model, extracting its texture from the image along with its geometry.

For the implementation of the application have used high-level libraries: OpenGL, OpenCV and Qt framework to create graphical user interfaces in C++. It is a desktop application with a Model-View-Controller architecture.

Once the base of the application is created, it is possible to include new functionalities in the future. As can be the case of modeling objects with complex shapes.

In the Figure 2 you can see the results that can be obtained with the software.



Figura 2: 3D reconstruction of an object. On the left: the image that the user loads into the application. Right: the resulting image after generating the model and make transformations on it (scale and traslate).

Models generated with the 3DPhotoModeling application can be exported in PLY format to be imported into other visualization or 3D modeling software to work on them.

---

Yo, **Juan Pablo Porcel Porcel**, alumno de la titulación **TITULACIÓN** de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 75928532N, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Juan Pablo Porcel Porcel

Granada a 12 de diciembre de 2016.

---

D. **Pedro Cano Olivares**, Profesor del Departamento Lenguajes y Sistemas Informáticos de la Universidad de Granada.

**Informa:**

Que el presente trabajo, titulado *3DPhotoModeling: Modelado interactivo para reconstrucción de objetos 3D basado en imágenes*, ha sido realizado bajo su supervisión por **Juan Pablo Porcel Porcel**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 12 de diciembre de 2016.

**El tutor:**

A handwritten signature in blue ink, appearing to read "Pedro Cano Olivares". Below the signature, there is a small mark or initial "W." followed by a diagonal line.

**Pedro Cano Olivares**

# Agradecimientos

A todos los que están conmigo y me apoyan tanto en los momentos buenos como en los malos.

A los que ya no están.



# Índice general

<b>1. Introducción y motivación</b>	<b>1</b>
1.1. Modelado en tres dimensiones . . . . .	1
1.1.1. Técnicas de modelado 3D . . . . .	2
1.1.2. Modelado poligonal . . . . .	2
1.1.3. Modelado de curvas . . . . .	3
1.1.4. Modelado CSG . . . . .	4
1.2. Reconstrucción de un objeto a partir de imágenes . . . . .	5
1.2.1. Técnicas multivistas . . . . .	6
1.2.2. Visión activa . . . . .	7
1.2.3. Técnicas de luz estructurada . . . . .	7
1.2.4. Telemetría láser . . . . .	7
1.2.5. Conclusiones . . . . .	8
1.3. Soluciones actuales . . . . .	8
1.3.1. 3-Sweep . . . . .	9
1.4. Motivación . . . . .	10
1.5. Objetivos . . . . .	12
1.6. Aspectos formativos previos . . . . .	13
<b>2. Especificación de requisitos</b>	<b>15</b>
2.1. Introducción . . . . .	15
2.1.1. Propósito . . . . .	15
2.1.2. Ámbito del Sistema . . . . .	15
2.1.3. Referencias . . . . .	16
2.1.4. Visión General del Documento . . . . .	16
2.2. Descripción General . . . . .	17
2.2.1. Perspectiva del Producto . . . . .	17
2.2.2. Funciones del Producto . . . . .	17
2.2.3. Características del Usuario . . . . .	18
2.2.4. Restricciones . . . . .	19
2.2.5. Suposiciones y Dependencias . . . . .	19
2.3. Requisitos específicos . . . . .	19
2.3.1. Interfaz de usuario . . . . .	19
2.3.2. Requisitos Funcionales . . . . .	20

2.3.3. Requisitos No Funcionales . . . . .	21
<b>3. Planificación</b>	<b>23</b>
3.1. Iteraciones . . . . .	23
3.1.1. Iteración 1 . . . . .	23
3.1.2. Iteración 2 . . . . .	24
3.1.3. Iteración 3 . . . . .	24
3.1.4. Iteración 4 . . . . .	24
3.2. Estimación del esfuerzo . . . . .	25
3.3. Recursos . . . . .	28
3.3.1. Recursos humanos . . . . .	28
3.3.2. Recursos hardware . . . . .	28
3.3.3. Recursos software . . . . .	28
<b>4. Análisis</b>	<b>31</b>
4.1. Actores . . . . .	31
4.2. Casos de uso . . . . .	31
4.2.1. Diagrama de casos de uso . . . . .	32
4.2.2. Descripción de los casos de uso . . . . .	33
4.2.3. Diagramas de actividad de los casos de uso . . . . .	43
4.3. Tecnología a utilizar . . . . .	43
<b>5. Diseño</b>	<b>47</b>
5.1. Arquitectura software . . . . .	47
5.2. Estimación de la postura 3D . . . . .	48
5.2.1. Modelo de cámara Pinhole . . . . .	49
5.2.2. Algoritmo POSIT . . . . .	51
5.3. Modelado de objetos por revolución . . . . .	52
5.4. Modelado de objetos cuboides . . . . .	55
<b>6. Implementación</b>	<b>57</b>
6.1. Licencia . . . . .	57
6.2. Lenguaje de programación . . . . .	57
6.3. Integración de OpenGL con Qt . . . . .	59
6.3.1. QtCreator . . . . .	60
6.4. Configuración de la escena 3D . . . . .	62
6.4.1. Imágenes bidimensionales . . . . .	62
6.4.2. Cámara . . . . .	64
6.4.3. Modelado en tres dimensiones . . . . .	67
6.4.4. Exportación de los modelos 3D . . . . .	74
6.4.5. Interfaz gráfica de usuario . . . . .	75

<b>7. Pruebas</b>	<b>77</b>
7.1. Modelado de objetos cuboides . . . . .	77
7.2. Modelado de objetos por revolución . . . . .	78
7.3. Exportación de modelos 3D . . . . .	81
<b>8. Conclusiones y trabajo futuro</b>	<b>85</b>
8.1. Conclusiones . . . . .	85
8.2. Trabajo futuro . . . . .	86
<b>Bibliografía</b>	<b>88</b>
<b>A. Manual de usuario</b>	<b>89</b>
A.1. Interfaz Gráfica de Usuario . . . . .	89
A.2. Importar una imagen al sistema . . . . .	91
A.3. Crear un objeto mediante revolución . . . . .	91
A.4. Crear un objeto cuboide . . . . .	94
A.5. Transformar modelo . . . . .	96
A.6. Exportar imagen en PNG . . . . .	100
A.7. Exportar modelo en PLY . . . . .	101



# Capítulo 1

## Introducción y motivación

### 1.1. Modelado en tres dimensiones

El **modelado en tres dimensiones** es una disciplina de la **Informática Gráfica**. Según Peter Shirley en [1] el modelado 3D trata del proceso de crear una representación matemática de las propiedades geométricas y de apariencia de cualquier objeto tridimensional (ya sea real o ficticio) de una manera que pueda ser almacenada en el ordenador.

Como resultado del modelado en tres dimensiones obtenemos un **modelo 3D** que representa ese objeto tridimensional, estos modelos pueden ser creados de manera automática o manual. Existen **multitud de soluciones software** para el modelado en tres dimensiones manualmente, este proceso de creación manual de un modelo 3d se puede asemejar con el proceso seguido en las artes prácticas y la escultura. Para un usuario no especializado en este proceso de modelado suele ser complejo el uso de estas herramientas.

Existen otras tecnologías que trabajan de manera **más automática** para el modelado en 3D de un objeto que van desde el escaneado 3D hasta la extracción de un objeto presente en varias imágenes. El problema de estas tecnologías es que pueden llegar a ser muy caras, como el caso del escáner 3D, o requieren demasiada información de la que puede que no dispongamos, como la necesidad de tener varias imágenes de un mismo modelo donde se represente desde distintas proyecciones. Pero, **¿qué pasa cuando no tenemos el modelo que queremos representar? ¿Y si sólo disponemos de una única imagen para tal fin?**

La capacidad de extraer objetos tridimensionales a partir de una única imagen ha sido caso de estudio durante mucho tiempo, este proceso involu-

cra numerosas tareas complejas como la estimación de la pose del objeto o el desconocimiento de los parámetros intrínsecos y extrínsecos de la cámara que capturó la imagen.

### 1.1.1. Técnicas de modelado 3D

Existen multitud de técnicas de modelado en tres dimensiones. A un alto nivel podemos hacer una primera división de estas técnicas en dos principales categorías dependiendo si el modelado se centra en definir el modelo según su volumen o únicamente por su contorno:

- **Modelado de sólidos** (Figura 1.1): Los modelos resultantes de esta técnica de modelado representan la frontera o superficie del objeto. Son más fáciles de definir y modificar. Para ello se utilizan conjuntos finitos de polígonos planos (caras).
- **Modelado de volúmenes** (Figura 1.2): Con estos modelos se representa tanto la frontera como el interior de los objetos, o propiedades localizadas en el espacio. Se utilizan en fabricación por computador y en aplicaciones médicas e industriales.

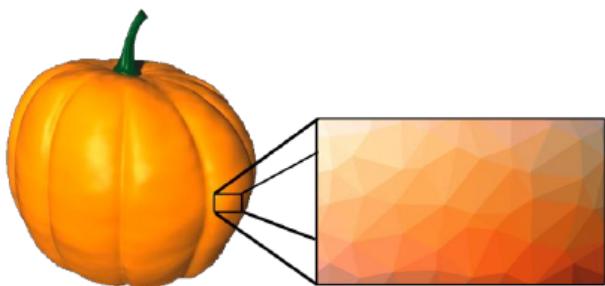


Figura 1.1: Representación del contorno de un objeto.

### 1.1.2. Modelado poligonal

Los modelos poligonales son los más utilizados debido a su velocidad de procesamiento y a la exactitud de definición que permite. Son puntos en un espacio 3D, vértices, conectados entre sí para formar una malla poligonal.

El **triángulo** es la primitiva más utilizada, debido a que las tarjetas gráficas con aceleración 3D están preparadas para trabajar óptimamente



Figura 1.2: Modelo basado en enumeración espacial. Fuente: [2]

con este tipo de polígonos. Además, el triángulo es el único polígono que garantiza que cualesquiera que sean las posiciones de sus vértices, siempre son coplanares.

Existen varias técnicas para modelar un objeto mediante polígonos:

- **Modelado de caja:** Es una técnica de modelado donde partimos de una figura prediseñada sencilla llamada primitiva (como un cubo, un cilindro, etc.) como base para añadir más vértices y crear una geometría más compleja para el modelo gane detalles.
- **Modelado por barrido (extrusión y revolución):** La técnica de modelado por barrido se basa en la idea de mover en el espacio un punto, una curva o una superficie, registrando los puntos por donde pasa. Hay dos elementos fundamentales: el generador, que es el objeto que origina el barrido, y la directriz, que es la trayectoria a través de la cual se desplazará el generador en el espacio. Dependiendo de las características de la directriz puede tratarse de un barrido translacional o rotacional. El caso del barrido rotacional también es llamado modelado por revolución. En la Figura 1.3 podemos ver un ejemplo de ambas técnicas de modelado por barrido.

### 1.1.3. Modelado de curvas

A veces, la definición de una superficie curva con un alto nivel de detalle puede requerir un altísimo número de polígonos. Por ello, en ciertas oca-

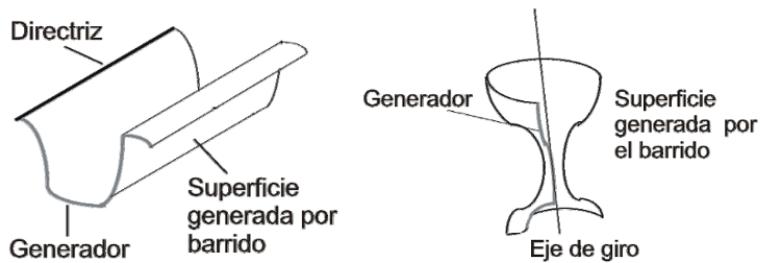


Figura 1.3: Izquierda: barrido translacional. Derecha: barrido rotacional.

siones puede ser interesante utilizar directamente superficies curvas para su representación.

Frente a la representación basada en conjuntos de polígonos, las curvas pueden ser descritas de un modo preciso mediante una ecuación. Estas ecuaciones pueden ser evaluadas y convertidas a conjuntos de polígonos (triángulos) en el momento de su representación para pintarlas.

En la Figura 1.4 podemos ver una representación de una superficie mediante una técnica de modelado de curvas (**NURBS**) y la misma superficie representada mediante polígonos.

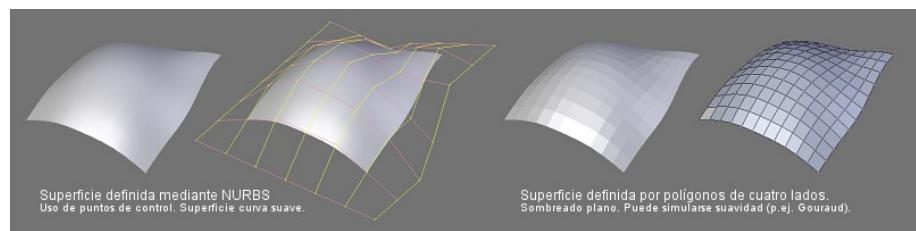


Figura 1.4: Representación de una superficie curva.

#### 1.1.4. Modelado CSG

La **Geometría Sólida Constructiva (CSG)** es un método para generar modelos 3D complejos mediante la aplicación de operaciones booleanas a los objetos primitivos. Las operaciones booleanas con las que se trabaja en CSG son:

- Unión
- Intersección

- Diferencia
- Complementario
- Diferencia simétrica

La aplicación de estas operaciones sobre los objetos funciona análogamente a la aplicación de las mismas en teoría de conjuntos. Así, se irán consiguiendo distintas formas, que irán aumentando en complejidad hasta dar lugar al resultado buscado (ver Figura 1.5).

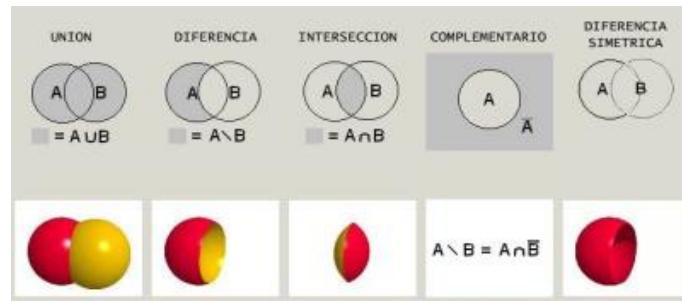


Figura 1.5: Operaciones booleanas en CSG.

## 1.2. Reconstrucción de un objeto a partir de imágenes

En los últimos años, el interés por las técnicas de **reconstrucción de objetos** reales en 3D ha aumentado no sólo en áreas de **visión artificial** sino también en otras áreas como la medicina, robótica, arqueología y multitud de campos que requieren modelado en tres dimensiones de ambientes reales.

El objetivo principal de estas técnicas de reconstrucción es obtener un modelo en tres dimensiones a partir de una o varias imágenes. Existen varias propuestas en la literatura del proceso de reconstrucción de objetos 3D cuyo principal objetivo es obtener un algoritmo que sea capaz de realizar la transformación de un conjunto de puntos representativos del objeto a una representación en tres dimensiones del mismo.

Podemos clasificar estas técnicas principalmente en cuatro grupos:

1. **Técnicas multivistas:** mediante estas técnicas conseguimos extraer

la información tridimensional mediante correspondencias de la información bidimensional procedente de dos o más imágenes [3].

2. **Visión activa:** permite extraer la información 3D a partir del flujo de imagen que proporciona una cámara móvil [4].
3. **Técnicas de luz estructurada:** permiten extraer la información tridimensional mediante la distorsión producida por la proyección de patrones generados mediante luz coherente o luz láser sobre los objetos reales [5].
4. **Telemetría láser:** permiten determinar el mapa de profundidad de la escena con base al tiempo transcurrido entre la emisión y detección de un punto láser [6].

### 1.2.1. Técnicas multivistas

El término **multivista** hace referencia a la existencia de más de una vista de la escena. Mediante esta técnica se pueden extraer las características tridimensionales de la escena en estudio a través de varias imágenes de la misma escena tomadas desde distintos puntos de vista. Según el número de imágenes que se use se aplican una serie de restricciones basadas en la geometría. La geometría de dos vistas es conocida como geometría epipolar o bifocal.

Según encontramos en [7] podemos clasificar las técnicas multivistas en dos tipos: dispersas o densas. La reconstrucción **dispersa** trata de obtener las coordenadas tridimensionales de ciertas partes de la escena conocidas como puntos de interés, como bordes, esquinas u otro tipo de puntos característicos. Este tipo de reconstrucción se utiliza en aplicaciones que necesitan un conocimiento del entorno en tiempo real y sin un detalle óptimo. La reconstrucción **densa** pretende obtener todos los puntos proyectados de cada objeto de la escena. Se utiliza en aplicaciones cuyo objetivo sea modelar digitalmente de manera realista una escena del mundo.

Las técnicas multivistas se centran en resolver principalmente dos problemas [8]:

- El problema de buscar la correspondencia entre un punto del plano de imagen izquierdo con el mismo punto en el plano de imagen derecho para poder recrear la escena.
- El problema de reconstrucción que trata de obtener, dados dos puntos correspondientes en ambos planos de imagen, las coordenadas tridimensionales del punto.

### 1.2.2. Visión activa

La técnica de **visión activa** permite la detección de objetos en movimiento y su seguimiento a través de la escena. Las cámaras disponen de sensores para moverse adecuadamente y seguir al objeto en cuestión, de manera que exista una correspondencia entre el mundo real y el virtual. Este tipo de técnicas tienen gran aplicación en la robótica.

### 1.2.3. Técnicas de luz estructurada

Este sistema se caracteriza por ser una técnica **directa y activa**. Directa porque puede obtener conclusiones estudiando los datos obtenidos directamente de las imágenes como encontramos en [9]. Activa porque es necesaria una fuente generadora de luz estructurada, por lo que necesita cambiar el entorno que se estudia.

Los sistemas de luz estructurada tratan de estudiar la deformación que sufre un patrón de luz sobre un objeto. El principal problema de estas técnicas es que se necesitan herramientas especiales para realizar el proceso, no son útiles cualquier sistema de iluminación normal, ya que está compuestos por ondas de diferente frecuencia. Además del patrón de luz, es necesario disponer de una cámara capaz de capturar todas las imágenes donde se muestre la deformación de la luz sobre el objeto.

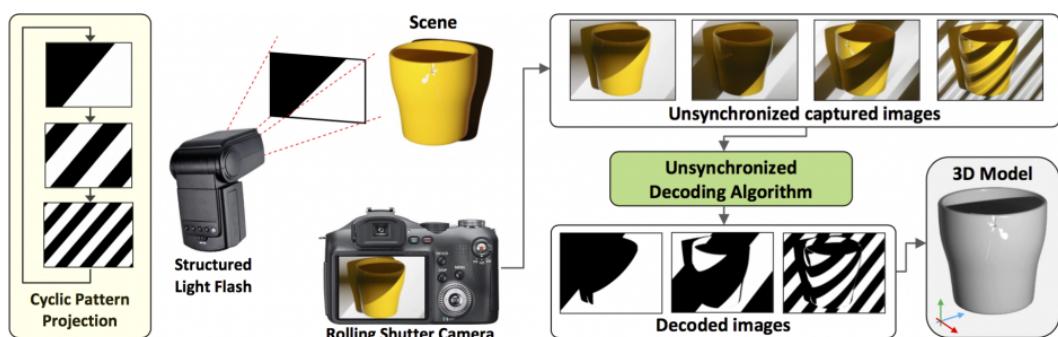


Figura 1.6: Funcionamiento del escáner 3-D de luz estructurada no sincronizado. Imagen: Taubin Lab. Fuente: Universidad Brown.

### 1.2.4. Telemetría láser

La técnica de **telemetría láser** se basa en medir el tiempo de recorrido de un rayo láser hasta la superficie de medida. Se puede medir mediante

dos formas: midiendo el tiempo entre la emisión del impulso luminoso y la observación del retorno o midiendo el desfase entre el rato emitido y la luz retornada al regular el impulso luminoso siguiendo una frecuencia determinada.

#### **1.2.5. Conclusiones**

En el estudio de las distintas técnicas de reconstrucción 3D a partir de imágenes se ha podido comprobar que el trabajo en la reconstrucción a partir de una única imagen no se ha explotado. Existen multitud de técnicas para tal efecto pero todas se basan en al menos la disposición de dos o más imágenes de la misma escena desde distintas proyecciones.

Resulta inviable el desarrollo de una técnica para la reconstrucción 3D de un objeto que aparece en una imagen de manera automática. La imagen digital, como la imagen analógica no es métrica. Es decir, **no podemos derivar medidas** de los objetos que aparecen en ella. Simplemente podemos visualizarlas e interpretarlas, pero no explotarlas métricamente.

Se puede realizar un estudio en profundidad de la **imagen bidimensional** para extraer información de interés, como bordes y esquinas, e intentar interpretar esta información para mapear los puntos 2D a coordenadas 3D en un sistema de coordenadas de mundo. Pero este trabajo es demasiado complejo y no aporta información suficiente para la reconstrucción de la escena.

Por este motivo, en el presente documento se analiza, diseña e implementa **una posible solución al problema de la reconstrucción tridimensional de una escena a partir de una sola imagen**. Esta técnica requiere de la ayuda del usuario para lograr extraer la información de interés de la imagen y así poder realizar la reconstrucción del objeto.

### **1.3. Soluciones actuales**

Para el modelado de objetos en 3D de manera manual existe en el mercado gran cantidad de software. Como 3D Studio Max, Maya, Blender y SketchUp entre otros. Con este software podemos conseguir una precisión en el modelo muy alta, hasta el punto de realismo que nosotros busquemos, pero a cambio puede llegar a ser muy complejo y llevar mucho tiempo de aprendizaje y trabajo.

En el caso de generación de modelos 3D a partir de imágenes también podemos encontrar programas como 123D Catch, Agisoft PhotoScan y PhotoModeler entre otros. El trabajo que realizan suele ser más automático. Un buen resultado depende mucho de la calidad de las imágenes o fotografías que se proporcionan al programa y de la pose que guarda el objeto en las distintas imágenes. Éstas suelen tener un formato específico y se suele necesitar una gran cantidad de imágenes para obtener un modelo fiel a la realidad.

En cuanto a software que genere un modelo 3D a partir de una única imagen no podemos encontrar mucho. El trabajo ya no es automático, suele ser necesaria la ayuda de un humano. Ejemplos de este tipo de software son Smoothie-3D, 3-Sweep y SketchUp entre otros. Especial interés en 3-Sweep, cuyo proyecto sirve de inspiración para este.

### 1.3.1. 3-Sweep

**3-Sweep** da nombre a una técnica desarrollada por investigadores de la Universidad de Tel Aviv y el Centro Interdisciplinario de Herzliya. El software fue presentado en la conferencia anual de SIGGRAPH (Special Interest Group on Computer GRAPHics and Interactive Techniques) de Asia en 2013 [10].

Se trata de un software de extracción de modelos 3D a partir de una imagen. El sistema es tan **simple y fácil** de usar que puede ser utilizado y comprendido por cualquier persona. 3-Sweep aprovecha las fortalezas de los usuarios que lo usan y la velocidad de cálculo del ordenador. Nuestras capacidades perceptivas reconocen la posición y las formas de los objetos, mientras que el ordenador realiza todo el trabajo pesado de texturización, cálculo y detección de bordes. El nombre de 3-Sweep, en español 3-Trazas, hace referencia a la técnica desarrollada. Con sólo tres trazas con el ratón sobre la imagen el software es capaz de crear formas y objetos con mayor o menor complejidad.

La idea principal de esta técnica es que mediante la fijación de la proyección de una parte del objeto, su posición y orientación pueden ser determinadas por sólo uno o dos valores de profundidad.

El usuario define una forma tridimensional usando **tres trazos** para las tres dimensiones, que se utilizan para definir un sistema de coordenadas ortogonal 3D. Primero, se define el origen del sistema de coordenadas en un



Figura 1.7: Ejemplo de uso del software 3-Sweep

punto de referencia  $R_i$  en la proyección de esta forma. Para un cuboide, se selecciona el punto que conecta el primer con el segundo trazo del usuario, y para un cilindro se selecciona el punto que une el segundo con el tercero. Debido a la ortogonalidad interna de la forma, el perfil suele ser perpendicular al eje principal. Por lo tanto, se puede usar el punto final del trazo del usuario (después de ajustarlo a la imagen) para definir tres puntos que juntos con  $R_i$  crean este sistema ortogonal. Este sistema de coordenadas es definido en coordenadas de cámara. Una vez definido el sistema ortogonal se puede estimar la posición de la cámara mediante el algoritmo POSIT [11].

Desde la misma aplicación se puede crear el modelo y realizar transformaciones sobre el mismo como se puede ver en la Figura 1.8 (e).

3-Sweep levantó mucha expectación tras su presentación en una de las conferencias de informática gráfica más importantes del mundo pero desde entonces no han surgido más noticias sobre él.

## 1.4. Motivación

Este proyecto surge con el objetivo de desarrollar una herramienta software de **modelado 3D** que dada únicamente una imagen como entrada y con la ayuda del usuario, éste sea capaz de crear una representación ma-

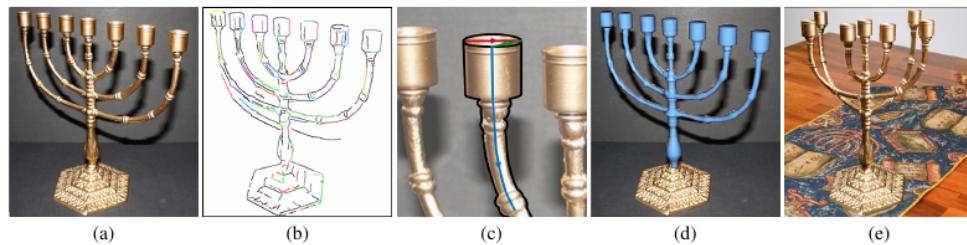


Figura 1.8: (a) Imagen de entrada (b) extracción de ejes (c) modelado de un componente del objeto (d) el modelo completo (e) transformaciones sobre el modelo. Imagen extraída de [10]

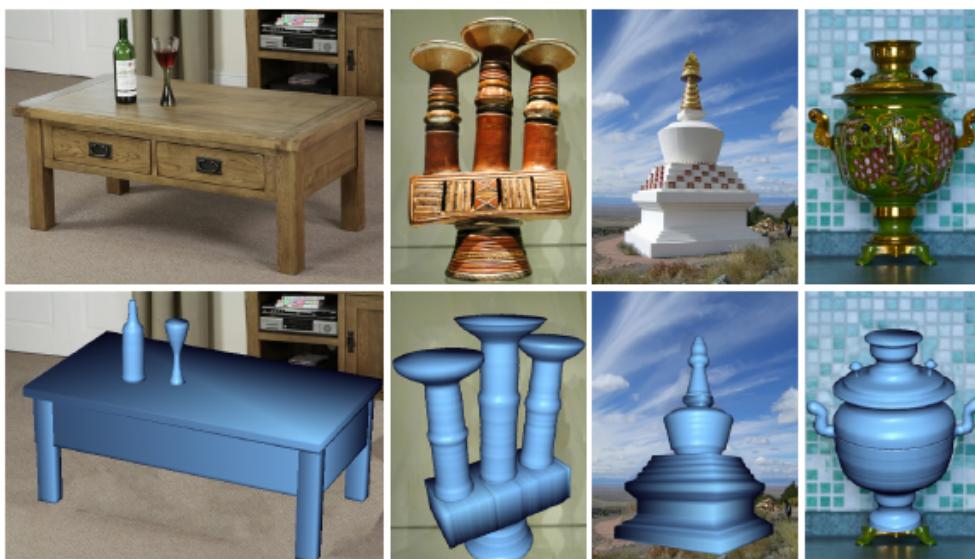


Figura 1.9: Objetos modelados con 3-Sweep. Arriba la imagen de base. Abajo el modelo extraído de la imagen. [10]

temática, un modelo 3D, de un objeto que se encuentre en dicha imagen. No será necesario un conocimiento profundo de modelado 3D para usar el software, así como no se necesita talento artístico para crear un modelo. Lo que ofrece esta herramienta es la posibilidad de crear objetos con una estructura simple basada en su superficie.

Tras la generación completa del modelo, que incluye el proceso de creación de la geometría del modelo junto con su apariencia, el usuario puede **transformar** el modelo para crear una nueva imagen. Estas transformaciones incluyen duplicar, escalar, trasladar y/o rotar el propio objeto para crear un modelo 3D completamente nuevo.

Los modelos creados desde la herramienta pueden ser exportados en formato **PLY** (Polygon File Format) para utilizarlos en otras herramientas de modelado o visualización en tres dimensiones. La principal ventaja de usar el formato PLY es que, aparte de que es un formato libre, permite una fácil comprensión y utilización. Este formato básicamente almacena información de los vértices y las caras. Permite una descripción mediante información en formato ASCII.

Así, el mismo software permite modelar un objeto a partir de una imagen de entrada de una manera sencilla para el usuario y poder retocar dicha imagen desde la misma herramienta. Las imágenes creadas con el software pueden ser exportadas en formato jpg o png.

## 1.5. Objetivos

Se desea desarrollar un **software de modelado 3D a partir de una sola imagen** utilizando **software libre**. El software estará destinado a usuarios que no tienen experiencia en el modelado 3D por lo que se espera que sea sencillo de usar. De esta manera se puede simplificar el proceso de generar una imagen retocada haciendo uso de los mismos objetos que aparecen en ella.

Se llevará a cabo un estudio, documentación y comparativa de los distintos algoritmos de detección de pose en una imagen que existen en la literatura. De esta manera se podrá llegar a una conclusión final de qué algoritmo implementar en el ámbito de este proyecto.

Se desea utilizar tecnologías libres para que cualquier desarrollador pueda aportar su conocimiento al proyecto. Para ello el software debe de estar acompañado de un proceso de documentación acorde a los proyectos de software libre.

A grandes rasgos los objetivos a cumplir son:

- **OBJ-1.** Estudiar y comprender el proceso de detección de pose en una imagen, generación de modelos 3D, etc.
- **OBJ-2.** Analizar, diseñar y desarrollar una interfaz gráfica sencilla y fácil de usar para un usuario medio.
- **OBJ-3.** Utilizar tecnologías libres para que cualquier desarrollador pueda aportar su conocimiento al proyecto.

- **OBJ-4.** Posibilidad de guardar los modelos extraídos de la imagen para poder usarlos en otro software. Además de exportar las imágenes retocadas.
- **OBJ-5.** Documentar todo el proceso de creación del software.

En los siguientes capítulos se tratará en detalle las técnicas utilizadas para la estimación y configuración de la escena 3D y el modelado de sólidos 3D (capítulo 4), la implementación con las distintas bibliotecas utilizadas (capítulo 6), un ejemplo de uso del sistema desarrollado (capítulo 7) y por último las conclusiones y el trabajo futuro (capítulo 8).

## 1.6. Aspectos formativos previos

Par el desarrollo de este proyecto son necesarios los conocimiento adquiridos en las siguientes asignaturas del grado:

- Dirección y Gestión de Proyectos: para el análisis y especificación de requisitos y para la planificación del proyecto.
- Informática Gráfica y Sistemas Gráficos: para comprender las tecnologías usadas en creación de escenas en tres dimensiones.
- Programación Orientada a Objetos: para el diseño y desarrollo del proyecto con un lenguaje como C++ orientado a objetos.



## **Capítulo 2**

# **Especificación de requisitos**

A continuación se detalla la especificación de requisitos software para la aplicación 3DPhotoModeling para el modelado en tres dimensiones y retocado de imágenes. Esta especificación se ha estructurado basándose en las directrices dadas por el estándar IEEE Guide to Software Requirements Specifications ANSI/IEEE 830, 1998.

### **2.1. Introducción**

#### **2.1.1. Propósito**

El presente documento tiene como propósito definir las especificaciones funcionales y no funcionales para el desarrollo de un sistema de modelado en tres dimensiones de objetos a partir de una sola imagen y retocado de dichas imágenes.

Este documento va dirigido a todas las personas que harán uso del sistema una vez haya sido terminado.

#### **2.1.2. Ámbito del Sistema**

El producto a desarrollar tomará el nombre de 3DPhotoModeling. Este producto puede ser clasificado como una aplicación de escritorio para el modelado en tres dimensiones de objetos sencillos a partir de una imagen para su posterior retocado aplicando distintas transformaciones a los modelos generados.

El producto debe ser capaz de calcular la geometría y apariencia de un objeto que aparece en una imagen gracias a la ayuda del usuario. Tras esto, el sistema debe realizar una estimación de la pose que guarda el objeto en la imagen para calcular la posición y orientación de la cámara que capturó dicha imagen.

El usuario podrá retocar la imagen de entrada gracias a las transformaciones que puede aplicar a los modelos generados: duplicar, trasladar, rotar o escalar además de importar nuevos objetos a la escena. En todo momento el usuario tendrá la opción de exportar tanto los modelos 3D generados como la imagen con las nuevas transformaciones. Así el producto debe de incorporar la característica WYSIWYG (What You See Is What You Get) para todos los formatos en que puede exportar. Esto es que lo que el usuario ve en la pantalla es lo que va a obtener en la exportación para un formato dado.

Los distintos formatos para exportar el trabajo realizado son:

- PLY. Para la exportación de los modelos 3D. Esto supone la posibilidad de guardar esos modelos para la posterior importación en la misma herramienta u otros programas de modelado o visualización en tres dimensiones. Con este formato podemos guardar fácilmente la información relativa a la geometría como la apariencia del modelo.
- PNG y JPG. Las imágenes retocadas con la herramienta podrán ser exportadas en formato png y/o jpg gracias a las utilidades que proporciona la librería OpenCV.

### 2.1.3. Referencias

Para la redacción de este texto se ha tenido en cuenta la siguiente documentación:

- [1] IEEE Std 830- IEE Guide to Software Requirements Specifications.

### 2.1.4. Visión General del Documento

Este documento de especificación de requisitos consta de tres secciones. Esta primera sección proporciona una visión general de la ERS. En la sección segunda se da una descripción general del sistema, con el fin de conocer las principales funciones que debe realizar, los supuestos, restricciones y dependencias que afectan al desarrollo, sin entrar en excesivo detalle. Por último, en la sección tercera, se definen de una manera detallada los requisitos que

debe satisfacer el sistema.

## 2.2. Descripción General

A continuación se describen los factores que afectan al producto y a sus requisitos.

### 2.2.1. Perspectiva del Producto

El presente producto se desarrolla como una aplicación de escritorio multiplataforma.

### 2.2.2. Funciones del Producto

Las funciones que debe realizar el producto se pueden clasificar en varios bloques:

#### 1. Almacenamiento en memoria de los modelos creados

- Almacenar en memoria la figura en dos dimensiones que el usuario está creando.
- Almacenar en memoria los distintos modelos generados.
- Posibilidad de eliminar un modelo.
- Almacenar la imagen de base y las modificaciones sobre ella.
- Exportar e importar un modelo en tres dimensiones en formato PLY.
- Exportar la imagen retocada.

#### 2. Herramientas para el modelado de objetos

- Herramienta para la creación un objeto por revolución.
- Herramienta para la creación de un cuboide genérico.
- Herramienta para la creación de una esfera.
- Asistente para la unión de los distintos objetos como un objeto más grande.

#### 3. Estimación de la orientación y posición de la cámara

- Estimar la orientación y posición de la cámara.

- Realizar un proceso de post-snapping para ajustar el modelo a la imagen.
  - Cambiar el campo de visión de la cámara en tiempo real.
4. Extracción de la apariencia del modelo
    - Extraer la textura del modelo de la imagen.
    - Ajustar los parámetros de luz que afectan al material del objeto.
  5. Transformaciones sobre los modelos creados
    - Escalar un objeto.
    - Rotar un objeto.
    - Trasladar un objeto.
    - Duplicar un objeto.
  6. Ayuda y asistencia para el usuario
    - Deshacer acciones realizadas.
    - Rehacer acciones deshechas.
    - Selección única y múltiple de modelos.

### **2.2.3. Características del Usuario**

A continuación se entra en detalle a describir qué tipo de usuarios van a usar el software y como afectan estos a las funciones que debe realizar el sistema.

En primer lugar y más importante, tenemos al usuario medio, sin un conocimiento profundo en modelado en tres dimensiones, que espera un uso sencillo de la aplicación sin necesidad de conocimientos técnicos ni talento artístico. Este tipo de usuario hace uso de la aplicación para generar una imagen retocada haciendo uso de los mismos objetos que aparecen en ella. El usuario espera poder exportar las imágenes creadas en un formato conocido y legible por otros sistemas software de visualización de imágenes.

Otro nivel de usuario son los que disponen de un conocimiento más extenso de modelado en tres dimensiones y desean una aplicación para generar un modelo que se encuentra en una imagen. El usuario desea exportar el modelo en un formato legible para otros sistemas software donde hará uso del modelo.

### 2.2.4. Restricciones

El sistema será desarrollado en C++ y se usarán distintas librerías de alto nivel:

- **OpenGL:** Es una librería que define una API multilenguaje y multiplataforma para la creación de gráficos 2D y 3D. En el contexto de esta aplicación se usa para el dibujado y transformación de los modelos 3D.
- **OpenCV:** Se trata de una biblioteca libre de visión artificial que dispone de funcionalidades para la estimación de la orientación y posición de la cámara necesarias para el desarrollo de este software.
- **Qt:** Es un framework multiplataforma de código abierto para desarrollar aplicaciones que hagan uso de una interfaz gráfica de usuario.

### 2.2.5. Suposiciones y Dependencias

El sistema 3DPhotoModeling funciona autónomamente, sin necesidad de comunicarse con otros sistemas externos, por lo que no hay dependencias respecto de otros sistemas.

## 2.3. Requisitos específicos

En este apartado se presentan los requisitos específicos que deberán ser satisfechos por el sistema. Estos requisitos se han especificado teniendo en cuenta, entre otros, el criterio de “testabilidad”: dado un requisito, debería ser fácilmente demostrable si es satisfecho o no por el sistema.

### 2.3.1. Interfaz de usuario

- RIU.1** La interfaz gráfica de usuario deberá ser intuitiva de manera que, sin un manual de uso, el usuario identifique rápidamente los componentes y herramientas del sistema.
- RIU.2** La interfaz de usuario debe ser orientada a ventanas y el manejo del programa se realizará a través de teclado y ratón.
- RIU.3** La interfaz gráfica debe de disponer de un acceso a todas las herramientas del programa desde el entorno de ventanas.
- RIU.4** La interfaz gráfica debe de mostrar en todo momento la imagen de base donde se muestren las operaciones realizadas sobre ella.

**RIU.5** El sistema debe de disponer de atajos de teclado para las acciones más realizadas por el usuario.

### 2.3.2. Requisitos Funcionales

- RF.1** El sistema debe ser capaz de almacenar en memoria todas y cada una de las figuras en dos dimensiones creadas por el usuario para su posterior manejo. Esta condición es necesaria para que puedan realizarse los siguientes requisitos. Esta función recibe como parámetro las trazas que el usuario ha realizado sobre la imagen con la ayuda de las distintas herramientas. El proceso consiste en almacenar en memoria estas figuras mediante una representación interna que identifique todas las propiedades de cada una de las figuras dibujadas sobre la pantalla.
- RF.2** El sistema debe ser capaz de almacenar en memoria los modelos en tres dimensiones generados a partir de las figuras dibujadas por el usuario. El sistema almacena una representación matemática de la geometría del modelo así como su apariencia. Es requisito indispensable para el posterior uso del modelo por parte del usuario.
- RF.3** El sistema debe ser capaz de eliminar un modelo de memoria si el usuario así lo desea. Esta acción se puede deshacer y recuperar el modelo generado.
- RF.4** El sistema debe almacenar la imagen de base para poder realizar acciones sobre ella.
- RF.5** El sistema debe ser capaz de exportar los modelos generados por el usuario en un formato legible por otros sistemas de modelado y visualización. La exportación de los modelos en tres dimensiones se hace en formato PLY.
- RF.6** El sistema debe ser capaz de exportar las imágenes retocadas por el usuario en un formato legible por otros sistemas software. Los formatos de exportación serán JPG y PNG.
- RF.7** El sistema debe ser capaz de importar un modelo en tres dimensiones en formato PLY para que el usuario pueda trabajar con él en la propia aplicación.
- RF.8** El sistema deberá de disponer de una herramienta de creación de modelos por revolución para generar objetos cilíndricos. El sistema dispondrá de una herramienta de fácil uso para el usuario para desarrollar esta función con una exactitud suficiente para poder representar cualquier objeto de revolución que aparezca en una imagen.

- RF.9** De igual manera que el requisito RF.8, el sistema deberá de disponer de una herramienta para crear cuboides con suficiente exactitud.
- RF.10** El sistema dispondrá de una herramienta para la creación de esferas.
- RF.11** El sistema deberá disponer de un asistente para combinar los modelos creados con las herramientas anteriormente descritas en un único modelo más complejo.
- RF.12** El sistema deberá ser capaz de calcular la posición y orientación de la cámara que capturó la imagen con una precisión muy alta. De esta forma el usuario podrá trasladar los objetos en la imagen así como añadir nuevos modelos a la escena que guarden la misma proyección.
- RF.13** El sistema deberá de calcular la proyección exacta de cada modelo en la imagen. Para este fin se realiza un proceso de post-snapping tras calcular la estimación de la pose para ajustar el modelo lo mejor posible a la proyección que guarda en la imagen.
- RF.14** El sistema debe tener una opción de cambiar el ángulo de apertura del campo de visión de la cámara en tiempo real para que el usuario pueda adaptar de mejor manera la proyección.
- RF.15** El sistema debe ser capaz de extraer la textura de un modelo de manera muy precisa para poder posteriormente pegar esta textura al modelo generado.
- RF.16** El sistema debe tener unas operaciones básicas de transformación de modelos: rotación, traslación y escalado para que el usuario pueda modificar los objetos generados.
- RF.17** El sistema debe tener la opción de duplicar un objeto ya generado.
- RF.18** El sistema debe tener la opción de deshacer y rehacer una acción que el usuario realice.

### 2.3.3. Requisitos No Funcionales

- RNF.1** El sistema debe ser fácil de usar para un usuario sin conocimientos de modelado en tres dimensiones.
- RNF.2** Debe permitir que se puedan añadir funciones: para futuras mejoras.
- RNF.3** El sistema no debe de hacer un gasto excesivo de la capacidad de cómputo y almacenamiento del sistema operativo donde se ejecute.
- RNF.4** El sistema debe disponer de un manual de usuario donde se detalle cómo utilizar el software con un lenguaje apropiado al nivel del usuario.



# Capítulo 3

## Planificación

Para el desarrollo de este proyecto se seguirá una metodología de desarrollo ágil basada en varias iteraciones. Cada iteración supone: planificación, análisis, diseño, implementación, pruebas y documentación.

El objetivo de cada iteración es completar una nueva funcionalidad del software sin errores. Así el valor del software será incrementado tras finalizar una iteración.

### 3.1. Iteraciones

Se han diseñado las siguientes iteraciones:

#### 3.1.1. Iteración 1

**Descripción** Desarrollar un entorno gráfico donde poder visualizar una imagen y dibujar sobre ella figuras en dos dimensiones.

**Objetivos**

- Comprender el uso de la librería de visión artificial OpenCV y sus funciones principales.
- Diseñar la jerarquía de clases para crear un proyecto con Qt Creator.
- Crear una interfaz gráfica de usuario sencilla para realizar operaciones básicas de dibujado.
- Estudiar y comparar los distintos algoritmos de detección de pose dada una imagen que se podemos encontrar en la literatura.
- Documentar todo el proceso realizado en esta iteración.

### 3.1.2. Iteración 2

**Descripción** Crear un entorno 3D para pintar los modelos en tres dimensiones generados a través de las figuras en 2D.

- Objetivos**
- Usar OpenGL para crear visualizar una escena en tres dimensiones con modelos 3D.
  - Implementar un algoritmo de detección de la posición y orientación de la cámara que capturó una imagen.
  - Crear una interfaz gráfica de usuario sencilla para realizar operaciones básicas de dibujado.
  - Crear distintas herramientas para que el usuario sea capaz de generar modelos básicos como cilindros, cubos y esferas.
  - Documentar todo el proceso realizado en esta iteración.

### 3.1.3. Iteración 3

**Descripción** Texturizado de los modelos y ajuste de la proyección.

- Objetivos**
- Implementar un algoritmo para extraer el material de un objeto que aparece en la imagen.
  - Desarrollar un algoritmo de post-snapping para ajustar de forma más precisa la proyección del objeto.
  - Completar la interfaz gráfica de usuario para introducir las nuevas funcionalidades.
  - El usuario será capaz de realizar transformaciones básicas sobre los modelos: rotar, escalar y trasladar.
  - Documentar todo el proceso realizado en esta iteración.

### 3.1.4. Iteración 4

**Descripción** Exportación de la escena.

- Objetivos**
- Dotar al sistema con una funcionalidad para exportar los modelos 3D creados en formato PLY.
  - El sistema será capaz de exportar las imágenes retocadas en formato PNG y JPG.
  - Finalizar la interfaz gráfica de usuario.
  - Documentar todo el proceso realizado en esta iteración.

Tras finalizar cada iteración se dispondrá de un prototipo usable con las funcionalidades implementadas en dicha iteración.

### 3.2. Estimación del esfuerzo

Haciendo uso de la especificación de requisitos se han extraído una serie de tareas que se detallan en las tablas 3.1 y 3.2. Cada tarea tiene asignada una prioridad y una estimación, así como la iteración donde se enmarca. La prioridad está medida en el rango 1 a 3, siendo 1 el valor de más prioridad. La estimación del esfuerzo está expresada en horas de trabajo. Para cada tarea se tiene en cuenta el tiempo de análisis, diseño, implementación, pruebas y documentación que se llevará a cabo para finalizarla.

El total de horas de trabajo estimadas es de **411 horas**. Suponiendo una media de trabajo diario de 4 horas, sin incluir fines de semana y festivos, el proyecto durará alrededor de **tres meses**. Esta duración es aproximada e ideal, no se tienen en cuenta los **riesgos** que pueden surgir durante el desarrollo del proyecto que pueden llegar a ocasionar que el tiempo de desarrollo aumente considerablemente.

La planificación temporal se muestra de manera gráfica con un diagrama de Gantt en las Figuras 3.1 y 3.2.

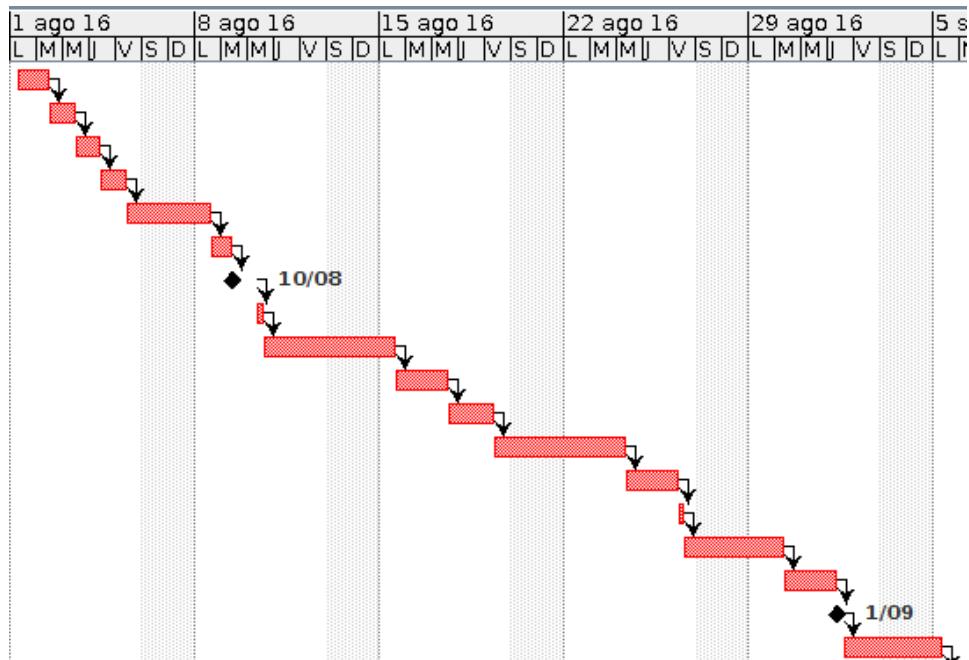


Figura 3.1: Diagrama de Gantt iteraciones 1 y 2

Nombre	Prioridad	Estimación	Iteración
Crear la interfaz de usuario.	1	12	1
Mostrar la imagen en la GUI.	1	8	1
Diseñar e implementar una estructura para guardar las figuras 2D.	1	7	1
Usar OpenCV para almacenar la imagen.	1	8	1
Diseño final prototipo iteración 1.	1	12	1
Pruebas prototipo iteración 1.	1	4	2
Documentación iteración 1.	1	6	1
Diseñar e implementar una estructura para guardar las figuras 3D.	1	6	2
Diseñar e implementar una herramienta para la creación de modelos por revolución.	1	24	2
Diseñar e implementar una herramienta para la creación de modelos cuboides.	1	16	2
Diseñar e implementar una herramienta para la creación de modelos esferas.	1	10	2
Estimación posición y orientación de la cámara.	1	24	2
Implementar algoritmo post-snapping.	1	16	2
Uso adecuado de los recursos del sistema operativo.	3	4	2
Diseño final prototipo iteración 2.	1	12	2
Pruebas prototipo iteración 2.	1	4	2
Documentación iteración 2.	1	6	2

Cuadro 3.1: Listado de tareas detalladas (I)

Nombre	Prioridad	Estimación	Iteración
Crear un menú donde mostrar todas las herramientas.	1	10	3
Crear objetos más complejos.	1	30	3
Extraer textura del modelo.	1	24	3
Transformaciones sobre el modelo.	3	28	3
Diseño final prototipo iteración 3.	1	12	3
Pruebas prototipo iteración 3.	1	4	3
Documentación iteración 3.	1	6	3
La interfaz gráfica de usuario deberá ser intuitiva.	2	8	4
Crear atajos de teclado para las acciones frecuentes.	2	6	4
Exportar modelos en formato PLY.	2	8	4
Exportar imagen retocada en formato PNG y JPG.	1	8	4
Importar un modelo en formato PLY.	2	8	4
Ajustar parámetros cámara desde GUI.	3	2	4
Duplicar modelos ya generados.	1	6	4
Opción de deshacer y rehacer.	1	12	4
Facilidad de uso del sistema.	2	10	4
Debe permitir que se puedan añadir funciones: para futuras mejoras.	3	8	4
Manual de usuario.	2	20	4
Diseño final prototipo iteración 4.	1	12	4
Pruebas prototipo iteración 4.	1	4	4
Documentación iteración 4.	1	6	4

Cuadro 3.2: Listado de tareas detalladas (II)

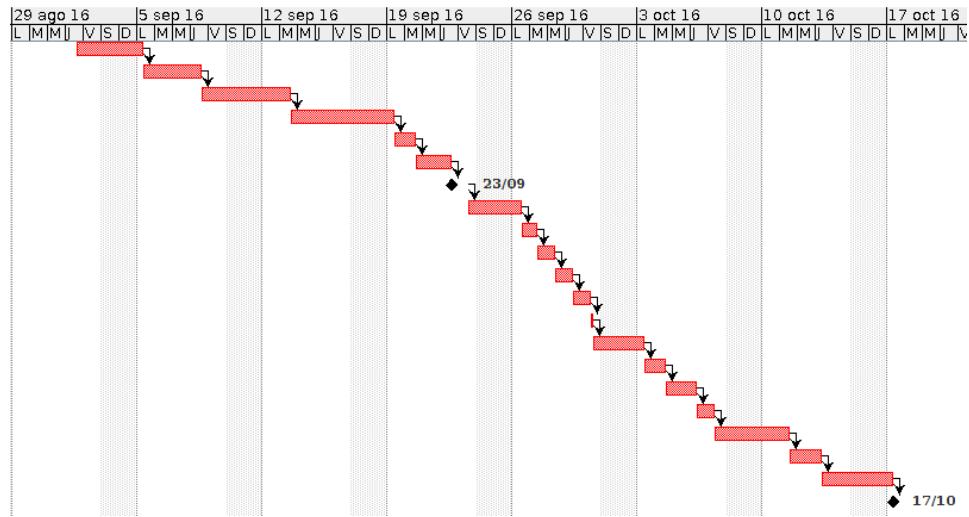


Figura 3.2: Diagrama de Gantt iteraciones 3 y 4

### 3.3. Recursos

#### 3.3.1. Recursos humanos

Gracias a que es un proyecto de software libre, todo aquel que desee participar en el proyecto puede hacerlo realizando aportaciones en los repositorios del proyecto [12].

Los recursos humanos contados para el desarrollo del proyecto tratan de una sola persona, el autor de este trabajo:

- Juan Pablo Porcel Porcel alumno de la Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones (ETSIIT) de la Universidad de Granada.

#### 3.3.2. Recursos hardware

- Ordenador portátil personal.

#### 3.3.3. Recursos software

- Sistema operativo Ubuntu 16.04 Xenial. Instalado en el ordenador personal, el software será ejecutado bajo este sistema operativo.
- Editor de textos Kate.

- Entorno de desarrollo Qt Creator. IDE que se utilizará para desarrollar el código y compilar el proyecto. La elección de este IDE frente a otros es la facilidad de trabajar y compilar C++ con las distintas librerías utilizadas.
- ProjectLibre. Software utilizado para la planificación temporal del proyecto.
- StarUML. Para el diseño de diagramas UML que se usan en el análisis y documentación del software.
- Justinmind. Para el prototipado de interfaces gráficas de usuario.
- TeXworks. Compilador Latex para la creación de la documentación.
- Librerías de alto nivel OpenGL, OpenCV y Qt.



# Capítulo 4

## Análisis

A continuación se documenta la fase de análisis del desarrollo del software. Aquí se describen las características necesarias para llevar a cabo los objetivos planteados.

### 4.1. Actores

Los actores son las entidades que interactúan con el sistema, pueden ser personas, programas u otros sistemas remotos. En el ámbito de este proyecto sólo encontramos un tipo de actor:

- **Usuario:** Se trata de la persona que usa el software final para generar una imagen retocada haciendo uso de los mismos objetos que aparecen en ella. El público objetivo pueden ser desde usuarios sin conocimientos en modelado en tres dimensiones y retocado de imágenes hasta profesionales en estos ámbitos. El objetivo final del usuario será exportar un modelo 3D creado con el programa y/o exportar una imagen retocada.

La Tabla 4.1 incluye una descripción del actor **Usuario**.

### 4.2. Casos de uso

La técnica de modelado de casos de uso nos ayuda a delimitar el sistema a estudiar, determinar el contexto de uso y describir el punto de vista de los usuarios del sistema. A continuación se detallan algunos de los casos de uso más importantes del software.

ACT-1	
<b>Actor</b>	Usuario
<b>Descripción</b>	Actor que usa el sistema directamente.
<b>Características</b>	Puede ser cualquier persona con conocimientos o no en el ámbito del software.
<b>Relaciones</b>	No tiene relaciones con otros actores.
<b>Referencias</b>	CU-1

Cuadro 4.1: Descripción del actor usuario

#### 4.2.1. Diagrama de casos de uso

En el diagrama de casos de uso que se muestra en la Figura 4.1 podemos ver gráficamente todos los elementos que forman parte del modelo de casos de uso junto con la frontera del sistema.

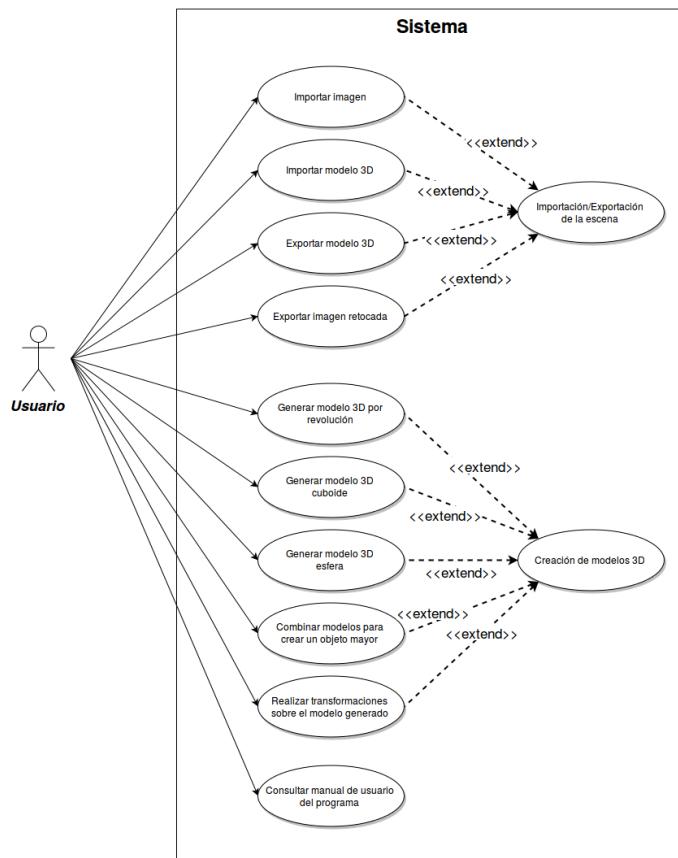


Figura 4.1: Diagrama de casos de uso

#### 4.2.2. Descripción de los casos de uso

<b>CU-1</b>	
<b>Caso de Uso</b>	Importar imagen
<b>Actores</b>	Usuario
<b>Tipo</b>	Primario, esencial
<b>Referencias</b>	
<b>Precondición</b>	La imagen que se quiere importar debe de existir y tiene que tener un formato PNG o JPG.
<b>Postcondición</b>	La imagen importada se mostrará en pantalla y el usuario podrá comenzar a trabajar sobre ella inmediatamente.
<b>Propósito</b>	
Abrir una imagen específica con el programa para poder trabajar sobre ella. El propósito del usuario es retocar esta imagen haciendo uso de modelos 3D que aparecen o no en la imagen.	
<b>Resumen</b>	
El usuario carga del administrador de archivos una imagen válida para ser modificada por el programa y poder generar una imagen nueva retocada.	
<b>Curso Normal</b>	
1	Usuario: Selecciona la opción de importar imagen.
2	Muestra la ventana de selección de archivos.
3	Usuario: Selecciona una imagen y pulsa sobre aceptar.
4	Carga la imagen en memoria.
5	Dibuja la imagen en pantalla.
<b>Cursos Alternos</b>	
4a	El archivo seleccionado no es una imagen o no tiene el formato esperado. El sistema informa del error.

CU-2	
<b>Caso de Uso</b>	Importar modelo 3D
<b>Actores</b>	Usuario
<b>Tipo</b>	Secundario, esencial
<b>Referencias</b>	
<b>Precondición</b>	El modelo que se importará debe de existir y tener un formato PLY.
<b>Postcondición</b>	El sistema lee el archivo y lo almacena en memoria como un modelo 3D. El modelo se muestra en pantalla sobre la imagen y el usuario puede trabajar con él.
<b>Propósito</b>	
Importar un modelo que se haya creado con la propia aplicación o con una aplicación externa de modelado en tres dimensiones para incorporarlo a una imagen.	
<b>Resumen</b>	
El usuario importa un modelo válido en el programa y puede trabajar con él inmediatamente realizando transformaciones sobre el modelo para crear una nueva imagen retocada.	

### Curso Normal

- 1 Usuario: Selecciona la opción de importar modelo 3D.
- 2 Muestra la ventana de selección de archivos.
- 3 Usuario: Selecciona un modelo que existe entre sus arvchivos y pulsa aceptar.
- 4 Carga el modelo en memoria.
- 5 Dibuja el modelo en pantalla sobre la imagen de base.

### Cursos Alternos

- 4a El archivo seleccionado no es un modelo 3D o no tiene el formato esperado (PLY). El sistema informa del error y abre de nuevo la ventana de selección de archivos.

**CU-3**

<b>Caso de Uso</b>	Exportar modelo 3D
<b>Actores</b>	Usuario
<b>Tipo</b>	Primario, esencial
<b>Referencias</b>	
<b>Precondición</b>	El usuario debe haber creado un modelo en tres dimensiones desde la aplicación y debe tenerlo seleccionado.
<b>Postcondición</b>	El programa guarda una copia del modelo en formato PLY en el directorio que el usuario ha elegido.

**Propósito**

Guardar un modelo creado con la aplicación para poder usarlo posteriormente en el mismo programa o en otros programas de modelado y visualización 3D.

**Resumen**

El usuario selecciona un modelo que crea con la aplicación y selecciona la opción de exportar, el sistema guarda una copia en formato PLY del modelo.

**Curso Normal**

- 1 Usuario: Selecciona un modelo presente en pantalla.
- 2 Usuario: Pulsa sobre la opción de exportar modelo.
- 3 Usuario: Selecciona un directorio y da un nombre al modelo.
- 2 Muestra una ventana con los directorios del usuario.
- 4 Guarda el modelo en formato PLY.

**Cursos Alternos**

- 2a El usuario no seleccionó ningún modelo para exportar. El sistema informa del error y no hace nada.

<b>CU-4</b>	
<b>Caso de Uso</b>	Exportar imagen retocada
<b>Actores</b>	Usuario
<b>Tipo</b>	Primario, esencial
<b>Referencias</b>	
<b>Precondición</b>	El usuario ha debido de importar una imagen antes al programa y retocarla con la aplicación.
<b>Postcondición</b>	El programa exporta la imagen retocada en formato PNG o JPG.
<b>Propósito</b>	
Guardar la nueva imagen retocada con la aplicación para tener una copia de esta.	
<b>Resumen</b>	
El usuario después de retocar la imagen de entrada con los modelos que ha creado o importado selecciona la opción de exportar imagen, el sistema guarda una copia de la imagen en el directorio seleccionado en formato PNG o JPG.	
<b>Curso Normal</b>	
1	Usuario: Selecciona la opción de exportar imagen.
2	Muestra una ventana con los directorios del usuario.
3	Usuario: Selecciona un directorio y da un nombre a la imagen.
4	Guarda la imagen en formato JPG o PNG.
<b>Cursos Alternos</b>	
4a	Si el usuario no dio formato a la imagen, por defecto el sistema exportará la imagen en JPG.

**CU-5**

<b>Caso de Uso</b>	Generar modelo 3D por revolución
<b>Actores</b>	Usuario
<b>Tipo</b>	Primario, esencial
<b>Referencias</b>	CU-9
<b>Precondición</b>	En la imagen debe aparecer un modelo que pueda ser creado por revolución. El usuario debe realizar adecuadamente los pasos para la creación de un modelo por revolución.
<b>Postcondición</b>	El sistema crea el modelo 3D asociado a la figura del objeto y el usuario puede realizar transformaciones sobre él.

**Propósito**

Generar un modelo 3D que represente a un objeto que aparece en la imagen. El modelo debe de ser lo más similar posible a la figura que aparece en la imagen.

**Resumen**

El sistema calcula una representación matemática del objeto que aparece en la imagen. El sistema realiza una estimación de la pose que guarda el objeto en la imagen. El sistema extrae la textura del objeto y la pega sobre el modelo.

**Curso Normal**

- 1 Usuario: Selecciona la opción de crear modelo por revolución.
- 2 Usuario: Define la base del modelo.
- 3 Usuario: Define el contorno del objeto.
- 4 Usuario: Selecciona la opción de crear modelo en 3D.
  - 5 Calcula la geometría del modelo
  - 6 Extrae la textura del objeto de la imagen y la pega sobre el modelo.
  - 7 Dibuja el modelo en pantalla sobre la imagen.

**Cursos Alternos**

- 4a El usuario no ha completado todos los pasos para crear una figura por revolución. El sistema informa del error y no hace nada.

<b>CU-6</b>	
<b>Caso de Uso</b>	Generar modelo 3D cuboide
<b>Actores</b>	Usuario
<b>Tipo</b>	Primario, esencial
<b>Referencias</b>	CU-9
<b>Precondición</b>	En la imagen debe aparecer un objeto con forma de cuboide que pueda ser generado por la aplicación.
<b>Postcondición</b>	El sistema crea el modelo 3D asociado a la figura del objeto y el usuario puede realizar transformaciones sobre él.
<b>Propósito</b>	
Generar un modelo 3D que represente a un objeto que aparece en la imagen. El modelo debe de ser lo más similar posible a la figura que aparece en la imagen.	
<b>Resumen</b>	
El sistema calcula una representación matemática del objeto que aparece en la imagen. El sistema realiza una estimación de la pose que guarda el objeto en la imagen. El sistema extrae la textura del objeto y la pega sobre el modelo.	
<b>Curso Normal</b>	
<ol style="list-style-type: none"> <li>1 Usuario: Selecciona la opción de crear modelo cuboide.</li> <li>2 Usuario: Define el modelo con la ayuda del sistema.</li> <li>3 Usuario: Selecciona la opción de crear modelo en 3D.</li> <li>4 Calcula la geometría del modelo</li> <li>5 Extrae la textura del objeto de la imagen y la pega sobre el modelo.</li> <li>6 Dibuja el modelo en pantalla sobre la imagen.</li> </ol>	
<b>Cursos Alternos</b>	
<p>3a El usuario no ha completado todos los pasos para crear la figura. El sistema informa del error y no hace nada.</p>	

**CU-7**

<b>Caso de Uso</b>	Generar modelo 3D esfera
<b>Actores</b>	Usuario
<b>Tipo</b>	Secundario, esencial
<b>Referencias</b>	CU-9
<b>Precondición</b>	En la imagen debe aparecer un objeto con forma de esfera que pueda ser generado por la aplicación.
<b>Postcondición</b>	El sistema crea el modelo 3D asociado a la figura del objeto y el usuario puede realizar transformaciones sobre él.

**Propósito**

Generar un modelo 3D que represente a un objeto que aparece en la imagen. El modelo debe de ser lo más similar posible a la figura que aparece en la imagen.

**Resumen**

El sistema calcula una representación matemática del objeto que aparece en la imagen. El sistema realiza una estimación de la pose que guarda el objeto en la imagen. El sistema extrae la textura del objeto y la pega sobre el modelo.

**Curso Normal**

- 1 Usuario: Selecciona la opción de crear modelo esfera.
- 2 Usuario: Define el modelo con la ayuda del sistema.
- 3 Usuario: Selecciona la opción de crear modelo en 3D.
- 4 Calcula la geometría del modelo
- 5 Extrae la textura del objeto de la imagen y la pega sobre el modelo.
- 6 Dibuja el modelo en pantalla sobre la imagen.

**Cursos Alternos**

- 3a El usuario no ha completado todos los pasos para crear la figura. El sistema informa del error y no hace nada.

<b>CU-8</b>	
<b>Caso de Uso</b>	Combinar modelos para crear un modelo mayor
<b>Actores</b>	Usuario
<b>Tipo</b>	Primario, esencial
<b>Referencias</b>	CU-5, CU-6, CU-7, CU-9
<b>Precondición</b>	El usuario debe haber creado varios objetos y tenerlos sobre la imagen.
<b>Postcondición</b>	El sistema combina los modelos seleccionados para crear un modelo con una geometría más compleja
<b>Propósito</b>	
Crear un modelo 3D más complejo que represente a un objeto que aparece en la imagen para poder trabajar con él.	
<b>Resumen</b>	
El usuario crea varios modelos, los selecciona y elige la opción de combinar modelos. El sistema combina los modelos para crear un objeto más complejo sobre el que el usuario pueda trabajar.	
<b>Curso Normal</b>	
1	Usuario: Selecciona varios objetos de la escena.
2	Usuario: Pulsa sobre la opción de combinar objetos.
3	Combina los modelos y calcula la nueva geometría.
4	Dibuja el modelo en pantalla.
<b>Cursos Alternos</b>	
2a	El usuario no ha seleccionado al menos dos modelos. El sistema informa del error y no hace nada.

**CU-9**

<b>Caso de Uso</b>	Realizar transformaciones sobre el modelo generado
<b>Actores</b>	Usuario
<b>Tipo</b>	Primario, esencial
<b>Referencias</b>	CU-5, CU-6, CU-7, CU-8
<b>Precondición</b>	El usuario debe haber creado o importado un modelo 3D y tenerlo seleccionado
<b>Postcondición</b>	El modelo será transformado (duplicado, escalado, rotado y/o trasladado) y guardará la nueva geometría. El resultado se mostrará en pantalla.

**Propósito**

Transformar un modelo para crear un objeto diferente al inicial.

**Resumen**

El usuario selecciona un modelo y elige una de las opciones de transformación. El sistema realiza la transformación sobre la geometría del modelo y lo dibuja en pantalla.

**Curso Normal**

- 1 Usuario: Selecciona un modelo de la imagen.
- 2 Usuario: Elige una transformación y transforma el modelo.
  - 3 Calcula la nueva geometría del modelo.
  - 4 Dibuja el modelo en pantalla.

**Cursos Alternos**

- 2a El usuario no ha seleccionado ningún modelo. El sistema no realiza ninguna acción.

<b>CU-10</b>	
<b>Caso de Uso</b>	Consultar manual de usuario del programa
<b>Actores</b>	Usuario
<b>Tipo</b>	Primario, esencial
<b>Referencias</b>	
<b>Precondición</b>	Ninguna
<b>Postcondición</b>	El sistema abrirá el manual de usuario.
<b>Propósito</b>	
Conocer cómo se realiza una acción determinada dentro de la aplicación.	
<b>Resumen</b>	
El usuario selecciona la opción de ver manual de usuario. El sistema muestra en pantalla o externamente el manual de usuario.	
<b>Curso Normal</b>	
1	Usuario: Pulsa sobre la opción de ver el manual de usuario
	2 Abre una ventana con el manual de usuario.
<b>Cursos Alternos</b>	

#### 4.2.3. Diagramas de actividad de los casos de uso

Este tipo de diagramas representan la actividad de un caso de uso, la actividad realizada por un conjunto de objetos o los flujos entre varios casos de uso.

Los diagramas de actividad para los casos de uso 5, 6 y 7 son muy similares por lo que se presenta en este documento únicamente el diagrama de actividad para el caso de uso 5 (Figura 4.6).

El caso de uso 10 es trivial por lo que hace su diagrama de actividad innecesario.

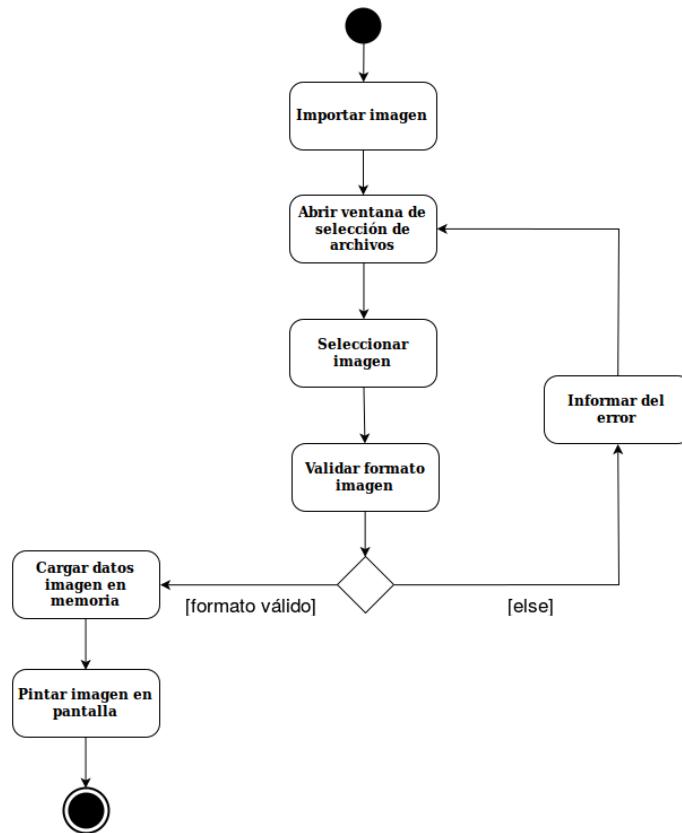


Figura 4.2: Diagrama de actividad para el caso de uso 1

#### 4.3. Tecnología a utilizar

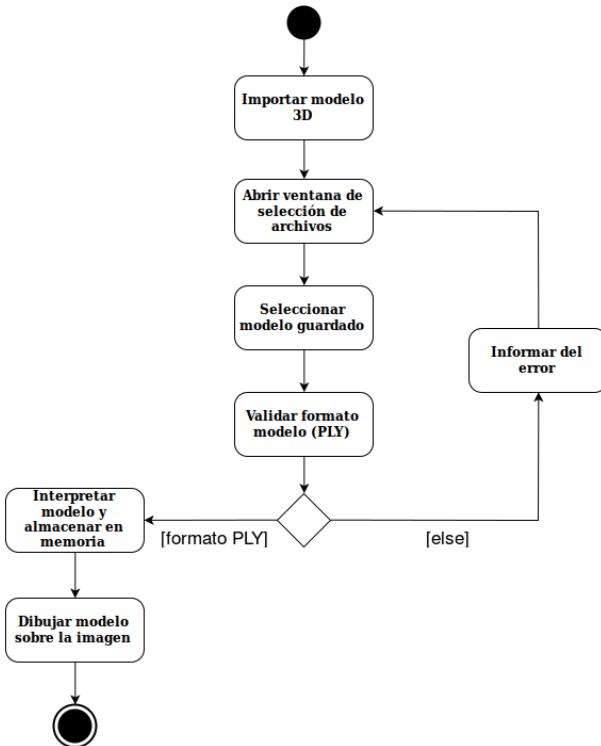


Figura 4.3: Diagrama de actividad para el caso de uso 2

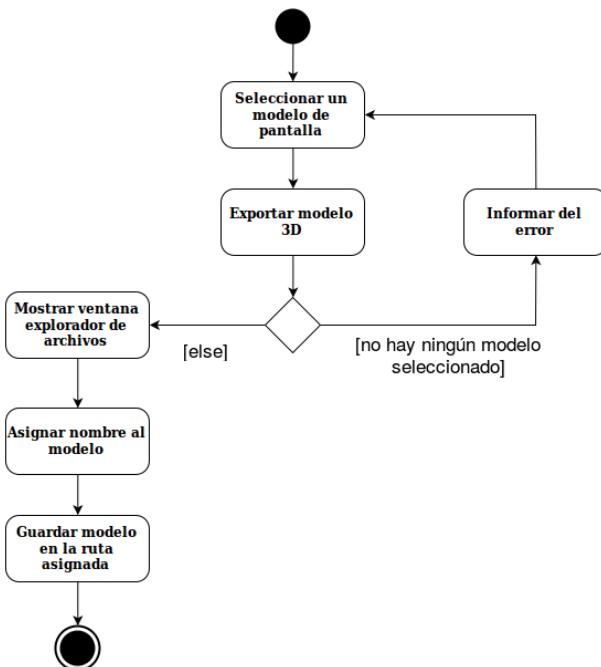


Figura 4.4: Diagrama de actividad para el caso de uso 3

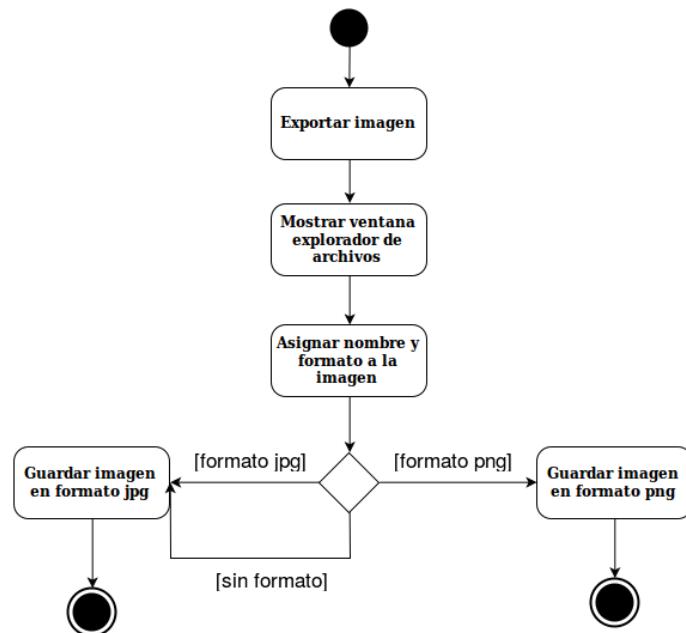


Figura 4.5: Diagrama de actividad para el caso de uso 4

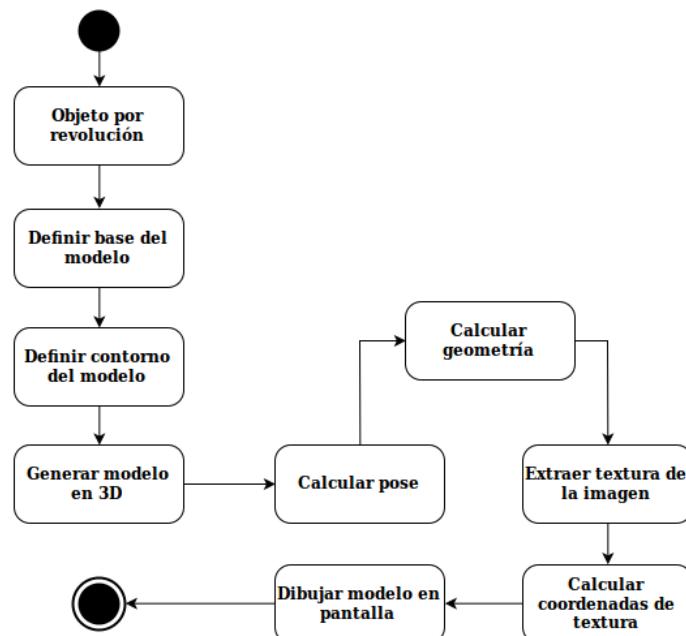


Figura 4.6: Diagrama de actividad para el caso de uso 5

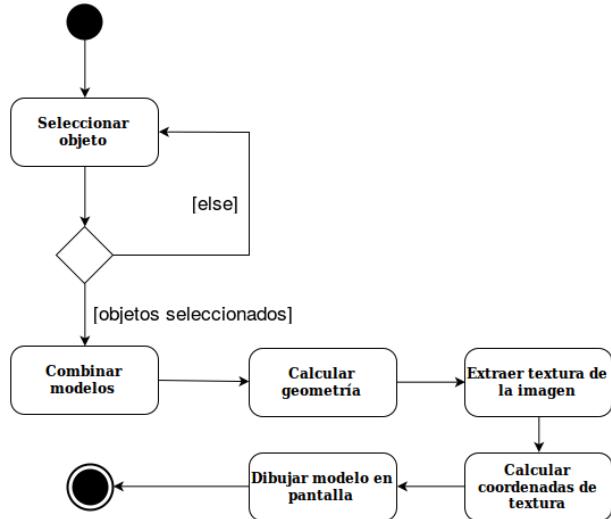


Figura 4.7: Diagrama de actividad para el caso de uso 8

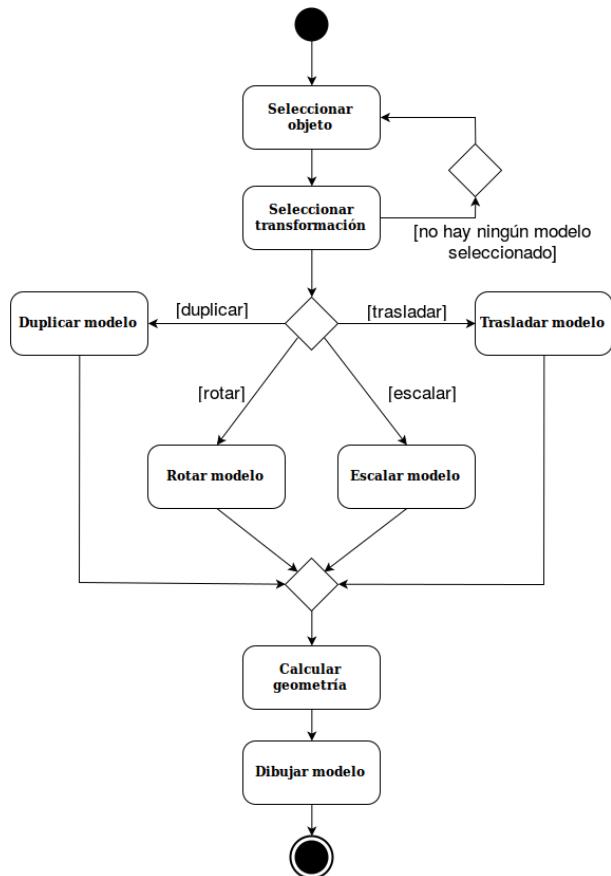


Figura 4.8: Diagrama de actividad para el caso de uso 9

# Capítulo 5

## Diseño

### 5.1. Arquitectura software

La arquitectura software definida para el proyecto es de Modelo Vista Controlador (MVC). Con esta arquitectura se pretende separar los datos y la lógica del sistema de la aplicación de la interfaz de usuario y el controlador encargado de gestionar los eventos y comunicaciones. Este patrón de diseño facilita el desarrollo y posterior mantenimiento de las aplicaciones gracias a la reutilización de código y a la separación de conceptos.

En la Figura 5.1 podemos ver de manera resumida la comunicación entre las distintas capas. Las líneas sólidas indican una asociación directa, y las punteadas una indirecta.

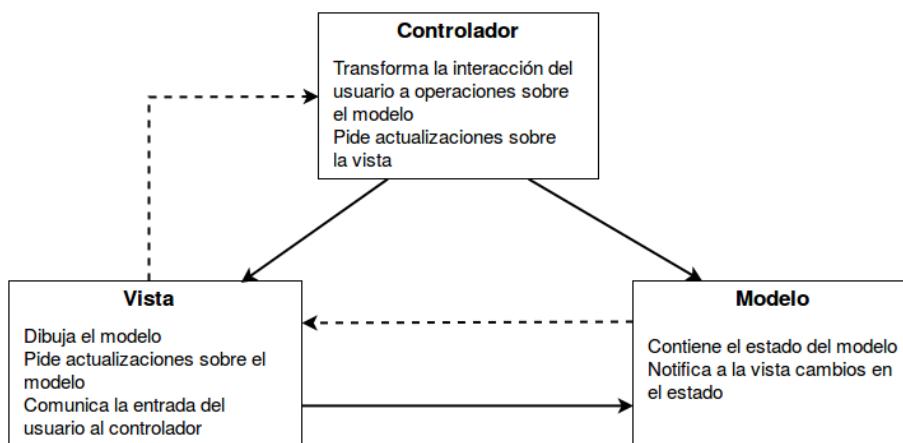


Figura 5.1: Arquitectura MVC

De esta manera, haciendo uso del patrón MVC, la arquitectura se divide en tres tipos de clases fundamentales con las siguientes responsabilidades:

- Clases para almacenar el modelo. Almacena la información de los modelos 3D creados por el usuario así como las figuras relativas a estos modelos.
  - Clases para la gestión de eventos. Principalmente la clase Controlador que gestiona todos los eventos que la interfaz recibe del usuario y los transforma en operaciones sobre el modelo.
  - Interfaz de usuario. La vista que el usuario puede ver. La ventana principal donde se carga la imagen importada y se dibujan los modelos creados.

En el diagrama de clases de la Figura 5.2 se puede ver la envergadura que tendrá el software a desarrollar. Las clases en color amarillo corresponden a la capa de Vista, las de color azul a la capa Controlador y por último, las de color rojo a la capa de Modelo.

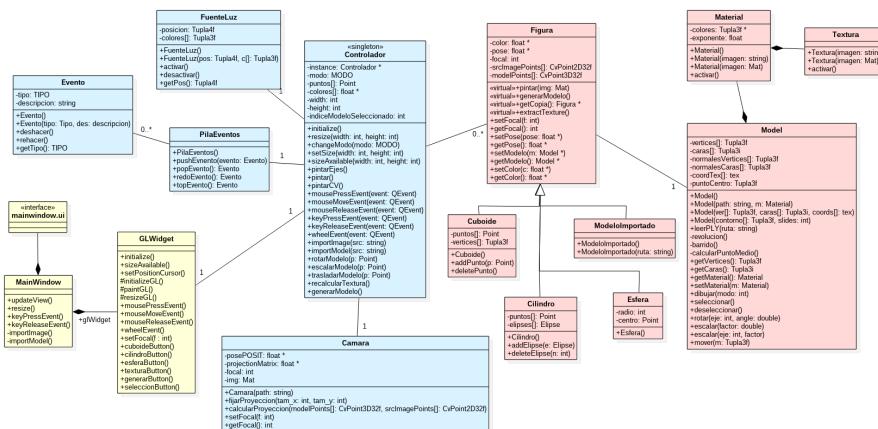


Figura 5.2: Diagrama de clases UML

## 5.2. Estimación de la postura 3D

Uno de los problemas más clásicos en visión por ordenador es la estimación de la posición y orientación que guarda un objeto en una imagen. Esta tarea ha sido caso de estudio durante muchos años y tiene una importante aplicación en robótica. El desarrollo de nuevos algoritmos en el campo de la visión artificial permite dotar a los robots móviles de una mayor autonomía en su funcionamiento.

### 5.2.1. Modelo de cámara Pinhole

Matemáticamente, una cámara puede modelarse como una función de transformación que mapea puntos en tres dimensiones a dos dimensiones. El modelo pinhole [13] [14] suele ser el habitual para modelar cámaras digitales (Figura 5.3).

Un punto  $P$  en tres dimensiones se proyecta sobre el plano de imagen a través del centro óptico. El centro óptico se encuentra a una distancia del plano de la imagen denominada distancia focal. El eje óptico es el rayo que pasa por el centro óptico y es perpendicular al plano de la imagen. El punto proyectado sobre la imagen ( $p$ ) es la intersección de la recta formada por el centro óptico y el punto  $P$  con el plano de la imagen.

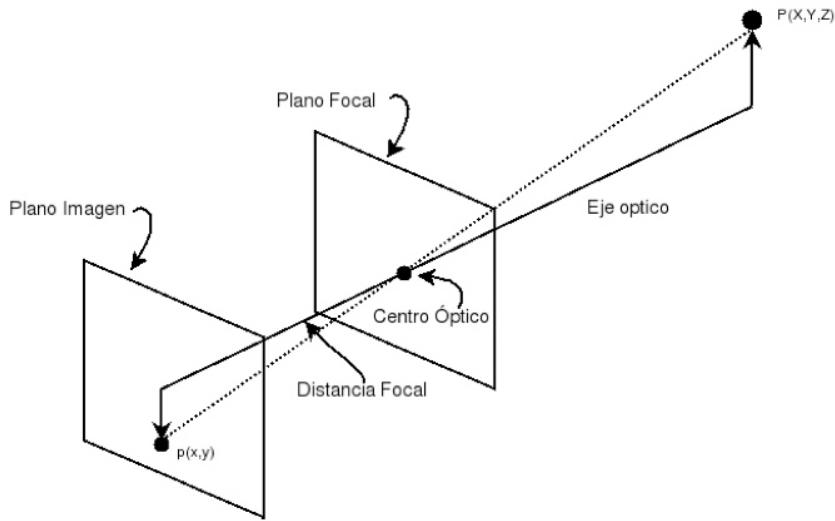


Figura 5.3: Modelo de cámara pinhole

Para  $P = (X, Y, Z)$  y  $p = (x, y)$  podemos obtener la relación 5.1:

$$\begin{cases} \frac{x}{f} = \frac{X}{Z} \\ \frac{y}{f} = \frac{Y}{Z} \end{cases} \rightarrow \begin{bmatrix} n_x \\ n_y \\ n \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (5.1)$$

Donde la matriz  $\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$  se llama matriz de perspectiva.

Normalmente los puntos de una escena se representan en el sistema de coordenadas del mundo y no en el de la cámara. Por lo tanto es necesario hacer una transformación que transforme coordenadas del mundo en coordenadas de la cámara. Esta transformación trata de una rotación y una traslación que se representa mediante una matriz llamada matriz de parámetros extrínsecos 5.2.

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} \quad (5.2)$$

Siendo  $(X, Y, Z)$  las coordenadas de la cámara y  $(X', Y', Z')$  las coordenadas del mundo.

La relación entre las coordenadas de la cámara  $(X, Y, Z)$  y las centrales de la imagen  $(x, y)$  viene definida como la siguiente ecuación 5.3.

$$\begin{bmatrix} n_x \\ n_y \\ n \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (5.3)$$

Por último se debe mencionar la distorsión geométrica causada por las imperfecciones de las lentes de la cámara. Esta distorsión ha de tenerse en cuenta ya que afecta a los puntos del plano de la imagen. Para obtener las coordenadas laterales de la imagen es necesaria la matriz  $K$ , llamada matriz de calibración de la cámara 5.4.

$$\begin{aligned} x_f &= K_x x_d + C_x \\ y_f &= K_y y_d + C_y \end{aligned} \quad (5.4)$$

En conclusión, un punto  $P(X', Y', Z')$  es proyectado en la imagen como  $p(x_f, y_f)$  de tal manera que el modelo final que relaciona ambas coordenadas sin distorsión lo podemos ver en la ecuación 5.5.

$$\begin{bmatrix} n_x \\ n_y \\ n \end{bmatrix} = \begin{bmatrix} K_x f & 0 & C_x & 0 \\ 0 & K_y f & C_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} \quad (5.5)$$

Siendo los parámetros de la matriz de la izquierda de la ecuación  $\begin{bmatrix} K_x f & 0 & C_x & 0 \\ 0 & K_y f & C_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$  los **parámetros extrínsecos** y los de la matriz de la derecha  $\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$  los **parámetros intrínsecos**. La multiplicación de ambas resulta en la denominada **matriz de proyección**.

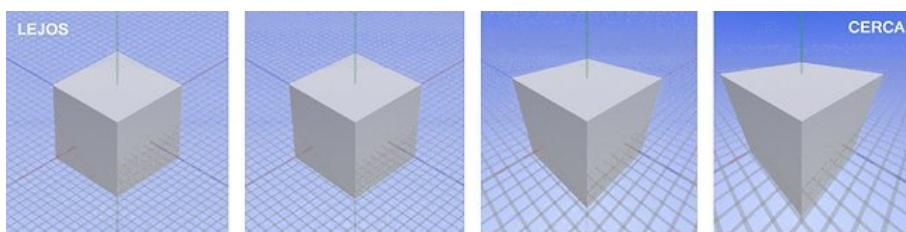


Figura 5.4: Distorsión de la imagen alterando la distancia focal

### 5.2.2. Algoritmo POSIT

El algoritmo **POSIT** (Pose from Orthography and Scaling with Iteration) desarrollado por DeMenthon y Davis [11] es uno de los algoritmos más usados a la hora de calcular la pose de un objeto conocido. Para este fin, el algoritmo POSIT necesita al menos cuatro puntos no coplanares del objeto y se asume que el objeto se encuentra a la misma profundidad. Esto es incorrecto, pero se asume que el objeto está lo suficientemente lejos como para que podamos prescindir del relieve del objeto. A este tipo de perspectivas se las llama *perspectivas débiles*.

La ejecución del algoritmo se divide principalmente en dos partes:

1. La fase **POS** se encarga de encontrar una pose aproximada del objeto. Para esto utiliza los parámetros intrínsecos de la cámara para hallar la perspectiva del objeto y así poder calcular la pose aproximada del objeto. Para encontrar dicha perspectiva el algoritmo necesita las coordenadas de los puntos en el sistema de coordenadas del objeto y sus correspondientes coordenadas de los mismos puntos en la imagen bidimensional. De esta manera es capaz de calcular una matriz de rotación y un vector de escalado que son los parámetros intrínsecos de la cámara.

2. A continuación la fase POSIT, POS con **Iteración**, ejecuta de nuevo la fase POS, pero en lugar de utilizar como datos de entrada los puntos del objeto, utiliza los resultados de POS. De esta forma, retroalimentando el algoritmo se consiguen mejorar los resultados y en cuatro o cinco iteraciones el algoritmo es capaz de converger a la pose real.

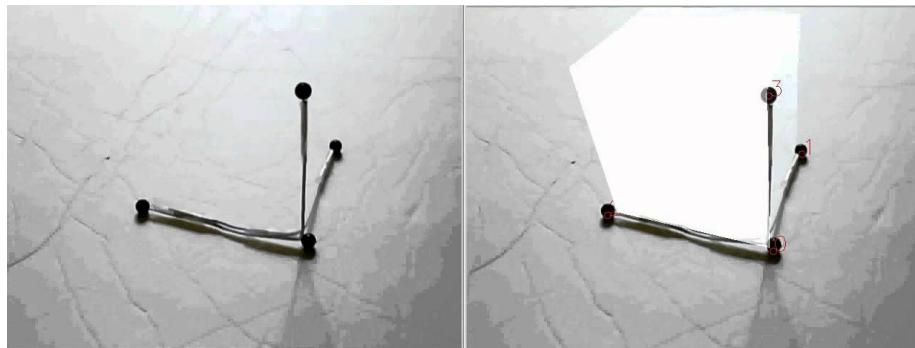


Figura 5.5: Con 4 puntos no coplanares el algoritmo POSIT puede calcular la pose 3D

Para la ejecución del algoritmo POSIT es necesario conocer los parámetros extrínsecos de la cámara que capturó la imagen, estos son la matriz de distorsión y la distancia focal. En la práctica se puede considerar que la imagen no tiene ninguna distorsión obviando los parámetros de la matriz de distorsión, en cambio sí es necesario el conocimiento de la distancia focal para una estimación de la pose adecuada.

Cómo se verá más adelante, en la implementación del algoritmo POSIT, no se conoce la distancia focal y ésta es calculada de forma estimada mediante un algoritmo iterativo para conseguir el mejor ajuste de la figura con la imagen.

### 5.3. Modelado de objetos por revolución

Una superficie de revolución es aquella que se genera mediante la rotación de una curva plana, o generatriz, alrededor de una recta directriz, llamada eje de rotación, la cual se halla en el mismo plano que la curva.

En este proyecto es de gran importancia el modelado de figuras por revolución ya que en la vida cotidiana podemos encontrar multitud de objetos que podemos modelar mediante una superficie de revolución. Por este motivo se ha puesto mucho interés en diseñar una técnica para generar estos modelos a partir de una imagen de forma sencilla. Esta técnica se basa principalmente en dos pasos que el usuario debe ejecutar:

1. **Base del objeto.** La tarea consiste en dibujar un perfil 2D correctamente orientado en 3D. Para simplificar esta tarea, se asume que la base es un círculo, esto reduce el número de parámetros desconocidos. Después, el disco circular puede ser deformado en un disco elíptico basado en una reconstrucción 3D. Dibujando dos líneas rectas, la primera define el diámetro mayor y la segunda línea se arrastra hasta el final del diámetro menor. Esto forma una elipse en la imagen que coincide con la proyección del disco circular. La profundidad del disco es definida como 0. La dirección de la normal y el radio del disco son asignados de acuerdo a la longitud y la orientación de los dos diámetros de la proyección elíptica (ver Figura 5.6).



Figura 5.6: El usuario define la elipse base mediante dos trazas, para definir el eje mayor y el eje menor.

2. **Perfil.** Después de completar el perfil base, el usuario barre una curva que se aproxima al eje principal del eje del modelo 3D. En general esta curva será perpendicular al perfil de la primitiva 3D. El usuario puede ajustar el tamaño y la perspectiva de la elipse generada para que se ajuste al perfil de la figura y definir el perfil tan detallado como desee (ver Figura 5.7 y Figura 5.8). Después de que el usuario dibuje el perfil del objeto sobre la imagen, el ordenador calcula una estimación de la superficie de revolución.

En la Figura 5.8 podemos ver la definición de un perfil de revolución más complejo haciendo uso de esta técnica. Así el usuario con solo el uso del ratón puede dibujar sobre la imagen un perfil de un objeto que posteriormente será



Figura 5.7: La última traza debe de terminar en la base del modelo.



Figura 5.8: El perfil del objeto puede ser más complejo.

calculado. Cuanto más detallado sea el perfil más similitud tendrá el modelo final con la figura que se encuentra en la imagen.

#### 5.4. Modelado de objetos cuboides

Un cuboide es un tipo de cubo pero todas sus caras no tienen porqué ser cuadradas. En geometría, un cuboide es definido como una caja rectangular cerrada que tiene tres pares de caras opuestas. También es llamado un paralelepípedo rectangular porque sus lados tienen una forma rectangular (Figura 5.9).

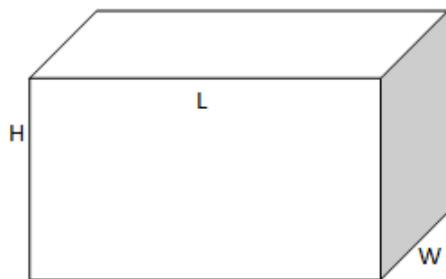


Figura 5.9: Cuboides

Los cuboides genéricos son modelados de manera similar a los cilindros. La principal diferencia se encuentra en la primera fase del modelado del perfil. Los dos trazos que definen el perfil del cuboide siguen los dos ejes de la base de el cuboide en lugar de los diámetros del disco. El usuario crea una base cuadrada para el objeto mediante 2 trazas del ratón que aproximan dos de los lados de la base del objeto y a continuación, con la tercera traza del ratón, ajusta la altura del cubo. El usuario puede definir el perfil del cubo de forma detallada al igual que anteriormente (ver Figura 5.10).

La técnica de modelado explicada arriba gracias a la ayuda de los gestos del usuario provee un conocimiento más inteligente pero menos preciso. Por lo tanto, después del modelado de cada primitiva, se aplica una etapa de post-snapping para ajustar mejor la primitiva a la imagen. Se buscan pequeñas transformaciones (+-10% el tamaño de la primitiva) y cambios del ángulo vertical de visión (+-10°), con esto conseguimos un mejor ajuste de la proyección del objeto.

En muchos casos, el objeto modelado tiene propiedades especiales que pueden ser usadas como prioridades de restricción del modelado. Por ejem-

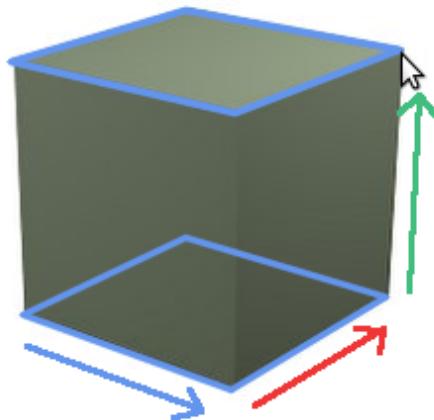


Figura 5.10: Definición de un cubo mediante la técnica de modelado de objetos cuboides.

plo, si sabemos que una parte dada tiene una columna recta, se puede restringir el barrido para que siga una línea recta. Similarmente, se puede restringir el trazo para preservar una restricción o linealidad cambiando el radio del perfil. En este caso, el radio detectado es ajustado a una linea sobre el trazo. También podemos restringir el perfil para que sea un cuadrado o un círculo.

# Capítulo 6

## Implementación

A continuación se detalla el desarrollo de la implementación de un prototipo del sistema donde podemos modelar objetos por revolución, objetos cuboides, exportar las imágenes generadas y exportar los modelos creados.

### 6.1. Licencia

Una de las características de este proyecto es que se trata de software libre. Así cualquier persona puede aportar sus conocimientos al proyecto y mejorar el producto. Se distribuye bajo licencia GPLv3 por los siguientes motivos:

- Es software libre.
- Obliga a la redistribución del proyecto con la misma licencia.
- El material debe de estar de forma ilimitada y gratuita para todo aquel que lo solicite.
- Está aprobada por Open Source Initiative.
- Es utilizable junto con otras licencias.

### 6.2. Lenguaje de programación

Para el desarrollo del proyecto se han estudiado las distintas alternativas en cuanto a lenguajes de programación que nos permitan crear el entorno 2D y 3D necesario para llevar a cabo el proyecto. Se ha elegido implementar el software en **C++** por la experiencia en la programación orientada objetos que se ha obtenido en el grado y de manera autodidacta, además de su

facilidad de trabajo con librerías como OpenGL para crear entornos en tres dimensiones y OpenCV para trabajar con imágenes bidimensionales.

### OpenCV

Para la implementación del software se ha utilizado la biblioteca libre de visión artificial **OpenCV**. Esta librería se ha utilizado en infinidad de aplicaciones dentro del área de la visión artificial. Desde sistemas de seguridad con detección de movimiento, hasta aplicaciones de control de procesos donde se requiere reconocimiento de objetos. Tiene interfaces de programación para C++, C, Python y Java y soporte multiplataforma para Windows, Linux, Mac OS, iOS y Android.

OpenCV dispone de más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos, calibración de cámaras, visión estérea y visión robótica. OpenCV tiene además una amplia documentación de toda la API que podemos consultar online.

### OpenGL

**OpenGL** es una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. La interfaz consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos [15].

En la década de los 80, los programas gráficos se escribían para cada hardware específico, escribiendo directamente sobre la memoria de video, y a finales de esta década, Silicon Graphics era la líder en estaciones gráficas, ofreciendo IrisGL, que era su API propietaria. En 1992, surgió OpenGL como evolución libre de IrisGL, permitiendo abstraer el uso de primitivas gráficas, sin necesidad de escribir directamente en la memoria de video.

OpenGL se centra única y exclusivamente en procesar unos datos, referentes a la escena 3D y generar un buffer de memoria, framebuffer, con la imagen 2D rasterizada. La API no se preocupa si esa imagen va por pantalla, a un archivo o en un holograma. Por tanto, será necesario utilizar una librería de interfaz de usuario distinta a OpenGL para gestionar las ventanas y los eventos, y aquí las posibilidades son casi infinitas: GLU, **Qt**, WxWidgets, Java, FLTK, Tcl-TK, etc.

## Qt

**Qt** es un framework multiplataforma orientado a objetos ampliamente usado para desarrollar programas (software) que utilicen interfaz gráfica de usuario. Qt es desarrollada como un software libre y de código abierto a través de Qt Project.

### 6.3. Integración de OpenGL con Qt

Toda aplicación que use OpenGL, independientemente del sistema operativo, el lenguaje de programación y la librería de gestión de usuario utilizada, necesita de unos pasos básicos o una funcionalidad que debe existir, definiéndose de una forma u otra en función de los requisitos del lenguaje. Este conjunto de pasos se suele llamar *OpenGL Rendering Pipeline*.

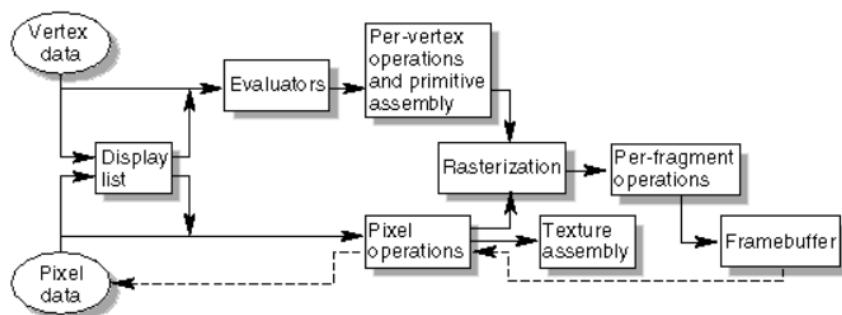


Figura 6.1: Pipeline de OpenGL

En el diagrama de la Figura 6.1 se puede apreciar el orden de operaciones que sigue el pipeline para renderizar. Por un lado está el *vertex data*, que describe los objetos de la escena, y por el otro lado, el *pixel data*, que describe las propiedades de la escena que se aplican sobre la imagen tal y como se representa en el buffer. Ambas se pueden guardar en una *display list* que es un conjunto de operaciones que se guardan para ser ejecutadas en cualquier momento.

En la sección de *primitive assembly*, se hace *clipping* de lo que queda fuera del plano de proyección, entre otros. Por la parte de *pixel data*, están las *pixel operations*. Aquí los píxeles son desempaquetados desde algún array del sistema (como el framebuffer) y tratados (escalados, etc.). Luego si estamos tratando con texturas, se preparan en la sección *texture assembly*.

Ambos caminos convergen en la *rasterization*, donde son convertidos en fragmentos. Cada fragmento será un píxel del framebuffer. Aquí es donde se tiene en cuenta el modelo de sombrado, las líneas, o el antialiasing.

En la última etapa, las *per-fragmet operations*, es donde se preparan los texeles (elementos de texturas) para ser aplicados a cada píxel, la niebla, el z-buffering, el blending, etc. Todas estas operaciones sedembocan en el framebuffer, donde se obtiene el render final.

### 6.3.1. QtCreator

Qt Creator es un entorno de desarrollo integrado (IDE) que proporciona al usuario las herramientas necesarias para diseñar y desarrollar aplicaciones con el framework Qt. Está diseñado para desarrollar aplicaciones e interfaces de usuario y desplegarlas a través del escritorio y sistemas operativos móviles. Proporciona herramientas para llevar a cabo todo el desarrollo de aplicaciones, desde la creación de un proyecto hasta el despliegue de la aplicación en las plataformas de destino (ver Figura 6.2).

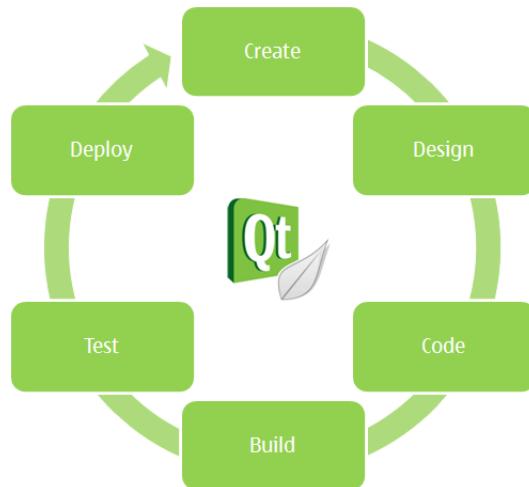


Figura 6.2: QtCreator

Un ejemplo de una aplicación de **Hola Mundo** con Qt en C++:

```

1 #include <QtGui/QApplication>
2 #include <QtGui/QLabel>
3
4 int main(int argc, char *argv[])
5 {
6     QApplication app(argc, argv);
    
```

```
7 QLabel label( QString :: fromUtf8( "Hola Mundo" ) );
8 label.show();
9 return app.exec();
10 }
```

El módulo de Qt OpenGL hace que el uso de OpenGL en aplicaciones basadas en Qt sea muy sencillo. Proporciona una clase widget OpenGL que puede ser utilizado como cualquier otro widget de Qt, excepto que se abre un búfer de pantalla OpenGL en la que se puede utilizar la API de OpenGL para representar el contenido. Desde la creación de interfaces de usuario podemos añadir el Widget a nuestra interfaz y crear una clase heredera de QGLWidget para gestionar las llamadas de OpenGL.

La clase definida en el proyecto que hereda de QGLWidget es la siguiente:

```
1 // glwidget.h
2
3 #ifndef GLWIDGET_H
4 #define GLWIDGET_H
5
6 #include <QGLWidget>
7
8 class GLWidget : public QGLWidget
9 {
10     Q_OBJECT
11
12 public:
13     explicit GLWidget(QWidget *parent = 0);
14     ~GLWidget();
15
16 protected:
17     void initializeGL();
18     void paintGL();
19     void resizeGL(int width, int height);
20
21     void mousePressEvent(QMouseEvent *event);
22     void mouseMoveEvent(QMouseEvent *event);
23     void mouseReleaseEvent(QMouseEvent *event);
24     void wheelEvent(QWheelEvent *event);
25     void keyPressEvent(QKeyEvent *event);
26     void keyReleaseEvent(QKeyEvent *event);
27
28 private:
29     Controlador * controlador;
30     MainWindow * main;
31 };
32 }
```

<sup>33</sup>  
<sup>34</sup> **#endif // GLWIDGET\_H**

QGLWidget proporciona la funcionalidad para la visualización de gráficos OpenGL integrados en una aplicación Qt. Es muy fácil de usar. Se hereda de él y utiliza la subclase como cualquier QWidget. QGLWidget proporciona tres funciones prácticas virtuales que se pueden implementar de nuevo en la subclase para realizar las tareas típicas de OpenGL:

- **paintGL()** Representa la escena descrita con OpenGL. Se llama cada vez que el widget necesita ser actualizado.
- **resizeGL()** Establece la ventana gráfica de OpenGL, proyección, tamaño, etc. Se llama cada vez que el usuario cambia el tamaño de la ventana y también cuando se muestra por primera vez para inicializar la vista.
- **initializeGL()** Establece el contexto de renderizado OpenGL, define las listas de la pantalla, etc. Se llama una vez antes de la función resizeGL() y paintGL().

Si se necesita actualizar la vista desde otra función que no sea paintGL() se debe de llamar a la función updateGL() de QGLWidget.

En el proyecto las funciones que se redefinen en QGLWidget comunican los eventos a la clase Controlador que gestiona las acciones del usuario y las transforma en acciones sobre el modelo.

## 6.4. Configuración de la escena 3D

El paso más importante del proceso que sigue este software es el de la configuración de la escena 3D. Es necesario realizar una estimación correcta de la posición y orientación de la cámara así como el correcto modelado de los objetos para conseguir plasmar la escena en tres dimensiones de la mejor manera posible.

Este proceso incluye varias tareas de mayor o menor complejidad que se tratarán en las siguientes secciones.

### 6.4.1. Imágenes bidimensionales

Partimos de una imagen real o sintética proporcionada por el usuario. En esta imagen podemos suponer que hay varios objetos y el usuario quiere

generar uno o más de ellos como un objeto 3D para transformarlo o simplemente exportarlo y ser usado en otro software.

La imagen es la base que ayuda al usuario a que el software pueda interpretarla para generar un modelo 3D a partir de sus trazas e interacciones. La única información que el sistema obtiene de la imagen es una matriz de ancho x alto de colores en formato rgb. En esta imagen el usuario conoce la pose del objeto que quiere generar y debe ayudar al sistema a conocer los puntos significativos que le permitan crear una serie de dependencias para llevar el objeto de 2D a 3D.

Es importante recalcar que al tratarse de una sola imagen 2D no conocemos al completo la superficie del objeto ya que hay caras ocultas en la imagen. Por este motivo es imposible obtener una representación exacta del modelo, el software realiza una estimación dadas las caras vistas por el usuario.

## OpenCV

OpenCV nos proporciona una manera muy sencilla de trabajar con imágenes bidimensionales. A través de la clase *cv::Mat* podemos almacenar y trabajar con las imágenes que el usuario importa al sistema [16].

La clase Mat representa un array de n dimensiones que puede tener un canal o múltiples canales. Este contenedor lo podemos usar para almacenar la imagen de entrada. Podemos cargar una imagen de un archivo con:

```
1 Mat img = imread( filename )
```

Una vez que tengamos la imagen cargada en memoria el sistema debe ser capaz de pintarla en pantalla. Para mostrar la imagen usamos nuestro QGLWidget y pintamos los pixeles de la imagen con OpenGL de la siguiente manera:

```
1 glDisable(GL_DEPTH_TEST);
2 glDrawPixels(image.size().width, image.size().height,
               GLBGR, GLUNSIGNED_BYTE, image.ptr());
3 glEnable(GL_DEPTH_TEST);
```

Las trazas que el usuario dibuja en la interfaz de usuario son directamente dibujadas sobre la matriz de imagen. Cada vez que se actualiza la pantalla

debemos de pintar los pixeles de nuestra imagen de nuevo ya que puede contener cambios o dibujos de las trazas del usuario.

### 6.4.2. Cámara

La estimación de la posición y orientación de la cámara es fundamental. OpenCV ya nos proporciona una implementación del algoritmo POSIT anteriormente descrito (*cvPOSIT()*). La pose P de un objeto 3D es la combinación de su orientación R (una matriz 3D de rotación) y su posición T (un vector 3D de traslación) relativos a la cámara.

Con el algoritmo *cvPOSIT* [17] podemos calcular los parámetros extrínsecos de la cámara dados una serie de puntos en tres dimensiones y sus correspondientes coordenadas de proyección sobre la imagen bidimensional y la distancia focal de la cámara. A continuación se detalla un ejemplo de uso del algoritmo para la estimación de la pose de un cubo.

### Vértices del modelo 3D

Antes de nada, debemos de crear un objeto POSIT con los vértices del modelo, para el ejemplo se usan cuatro esquinas del cubo. El primer punto del vector que se ha de pasar a la función *cvCreatePOSITObject* debe ser (0,0,0). Este punto es conocido como el punto de referencia del objeto. El algoritmo POSIT devolverá la traslación de la cámara a este punto.

```

1 float cubeSize = 10.0;
2 std::vector<CvPoint3D32f> modelPoints;
3 modelPoints.push_back(cvPoint3D32f(0.0f, 0.0f, 0.0f));
4 modelPoints.push_back(cvPoint3D32f(0.0f, 0.0f, cubeSize));
5 modelPoints.push_back(cvPoint3D32f(cubeSize, 0.0f, 0.0f));
6 modelPoints.push_back(cvPoint3D32f(0.0f, cubeSize, 0.0f));
7 CvPOSITObject *positObject = cvCreatePOSITObject( &
    modelPoints[0], static_cast<int>(modelPoints.size()) );

```

### Correspondencias en la imagen

Debemos de crear un vector de puntos 2D que corresponden a los vértices del modelo que se han especificado anteriormente. Estos puntos deben de estar situados en el vector en el mismo orden que el vector de vértices

del modelo. En otras palabras, el primer punto de este vector debe de corresponder con la proyección de el primer vértice del vector de vértices. El origen de coordenadas de la imagen es situado en el centro.

```

1 std :: vector<CvPoint3D32f> projectedPoints;
2 projectedPoints . push_back ( cvPoint2D32f( x0 , y0 ) );
3 projectedPoints . push_back ( cvPoint2D32f( x1 , y1 ) );
4 projectedPoints . push_back ( cvPoint2D32f( x2 , y2 ) );
5 projectedPoints . push_back ( cvPoint2D32f( x3 , y3 ) );

```

### Estimación de la pose

Ahora que tenemos los vértices del modelo y sus correspondencias en la imagen podemos calcular la pose:

```

1 CvMatr32f rotation_matrix = new float [9];
2 CvVect32f translation_vector = new float [3];
3 CvTermCriteria criteria = cvTermCriteria(CV_TERMCRIT_EPS |
   CV_TERMCRIT_ITER, 100, 1.0e-4f);
4 cvPOSIT( positObject , &imagePoints[0] , FOCALLENGTH,
   criteria , rotation_matrix , translation_vector );

```

### Matriz de vista

Para poder dibujar el modelo usando OpenGL debemos de construir la matriz de vista a partir de la matriz de rotación y el vector de traslación que obtenemos con cvPOSIT.

```

1 for ( int f=0; f<3; f++)
2   for ( int c=0; c<3; c++)
3     posePOSIT [ c*4+f ] = rotation_matrix [ f*3+c ]; // transposed
4
5 posePOSIT [ 3 ] = 0.0;
6 posePOSIT [ 7 ] = 0.0;
7 posePOSIT [ 11 ] = 0.0;
8 posePOSIT [ 12 ] = translation_vector [ 0 ];
9 posePOSIT [ 13 ] = translation_vector [ 1 ];
10 posePOSIT [ 14 ] = translation_vector [ 2 ];
11 posePOSIT [ 15 ] = 1.0;

```

## Matriz de proyección

Podemos construir la matriz de proyección con los parámetros intrínsecos:

- la distancia focal
- el punto principal de la imagen, el cual consideramos que es el centro de la imagen
- la resolución de la imagen, ancho y alto en píxeles
- los valores de clipping farPlane y nearPlane

```

1 cv::Mat intrinsics(3, 3, CV_32F);
2 intrinsics.at<double>(0,0) = focal;
3 intrinsics.at<double>(1,1) = focal;
4 intrinsics.at<double>(0,2) = width * 0.5;
5 intrinsics.at<double>(1,2) = height * 0.5;
6 intrinsics.at<double>(2,2) = 1.0;
7
8 double farPlane = 1000.0;
9 double nearPlane = 1.0;
10
11 projectionMatrix[0] = 2.0 * intrinsics.at<double>(0,0) /
12     width;
12 projectionMatrix[1] = 0.0;
13 projectionMatrix[2] = 0.0;
14 projectionMatrix[3] = 0.0;
15
16 projectionMatrix[4] = 0.0;
17 projectionMatrix[5] = 2.0 * intrinsics.at<double>(1,1) /
18     height;
18 projectionMatrix[6] = 0.0;
19 projectionMatrix[7] = 0.0;
19
20 projectionMatrix[8] = 2.0 * ( intrinsics.at<double>(0,2) /
21     width ) - 1.0;
22 projectionMatrix[9] = 2.0 * ( intrinsics.at<double>(1,2) /
22     height ) - 1.0;
23 projectionMatrix[10] = -( farPlane+nearPlane ) / ( farPlane
24     - nearPlane );
24 projectionMatrix[11] = -1.0;
25
26 projectionMatrix[12] = 0.0;
27 projectionMatrix[13] = 0.0;
28 projectionMatrix[14] = -2.0 * farPlane * nearPlane / (
29     farPlane - nearPlane );
29 projectionMatrix[15] = 0.0;
```

### Dibujar el modelo con OpenGL

Con las matrices de proyección y de vista calculadas podemos utilizar OpenGL para dibujar el modelo de la siguiente manera [18]:

```
1 glClear( GL.COLOR_BUFFER_BIT | GL.DEPTH_BUFFER_BIT );
2 glViewport(0, 0, imageWidth, imageHeight);
3 glMatrixMode( GL_PROJECTION );
4 glLoadMatrixd( projectionMatrix );
5 glMatrixMode( GL_MODELVIEW );
6 glScalef( 1.0f, 1.0f, -1.0f );
7 glMultMatrixf( posit.posePOSIT );
8 drawModel();
```

El motivo por lo que realizamos un escalado de -1 en el eje Z antes de dibujar es porque a diferencia de OpenGL, OpenCV usa la regla de la mano izquierda en lugar de la mano derecha.

#### 6.4.3. Modelado en tres dimensiones

Uno de los objetivos principales del proyecto es abstraer al usuario de todo el proceso de modelado de objetos en tres dimensiones facilitándole una herramienta fácil de usar. El usuario desea generar rápidamente un objeto que aparece en una imagen para trabajar con él en el mismo sistema o con otro software. A continuación veremos la implementación de las técnicas descritas en el capítulo de diseño de modelado de objetos por revolución y modelado de cuboides.

Antes de profundizar en las técnicas de modelado desarrolladas cabe mencionar la estructura de datos que se usa para guardar los modelos tridimensionales así como las figuras en dos dimensiones que crea el usuario.

Para la implementación del software se almacenan todas los eventos que realiza el usuario sobre el sistema para así poder convertir estos eventos de ratón y teclado en información útil. Estas traducciones las realiza la clase Controlador. Cada traza que el usuario realiza con el ratón sobre nuestro GLWidget es almacenada y transformada en una acción sobre el modelo de datos.

Para representar las figuras bidimensionales que el usuario dibuja sobre la imagen para crear el contorno del modelo se ha implementado una jerar-

quía de clases (Figura 6.3).

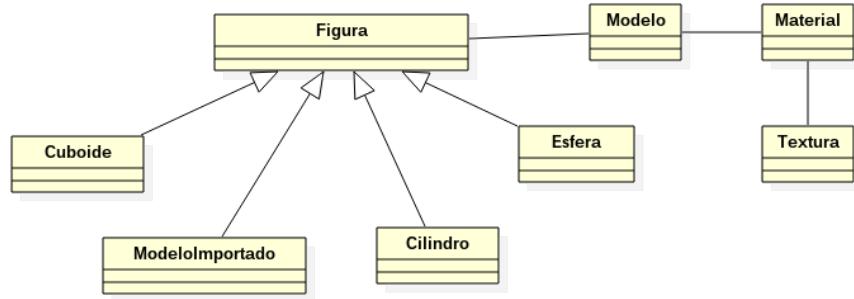


Figura 6.3: Jerarquía de clases para almacenar los modelos

En el diagrama de clases resumido de la Figura 6.3 podemos ver que la clase padre es **Figura**, de esta clase heredan las clases **Cuboide**, **Cilindro**, **Esfera** y **ModeloImportado**, para representar las distintas figuras con las que trabaja el sistema. La implementación de la clase **Figura** es la siguiente:

```

1 #ifndef FIGURA_H
2 #define FIGURA_H
3
4 #include <iostream>
5 #include <opencv2/opencv.hpp>
6 #include "modelo.h"
7
8 class Figura
9 {
10
11 protected:
12
13     Model * modelo;
14     float * color;
15     float * pose;
16     float focal;
17     std::vector<cv::Point2D32f> srcImagePoints;
18     std::vector<cv::Point3D32f> modelPoints;
19
20 public:
21     virtual void pintar(cv::Mat &img)=0;
22     virtual void generarModelo()=0;
23     virtual Figura * getCopia()=0;
  
```

```

24     virtual void extractTexture()=0;
25     void setModelo(Model * m) { modelo = m; }
26     Model * getModelo() { return modelo; }
27     void setColor(float * c) { color = c; }
28     float * getColor() { return color; }
29     void setPose(float * p)
30     {
31         pose = new float[16];
32         for(int i=0; i<16; i++) pose[i] = p[i];
33     }
34     float * getPose() { return pose; }
35     virtual void setFocal(float f)=0;
36     float getFocal(){ return focal; }
37 };
38
39 #endif // FIGURA.H

```

Los métodos definidos como virtual no están implementados en la clase *Figura* ya que estos los sobreescriben las clases que heredan. Cada instancia de la clase *Figura* almacena una instancia de la clase *Modelo* que es donde se almacena el modelo tridimensional resultante de la figura en dos dimensiones.

La clase controlador guarda un vector de figuras para representar todas y cada una de los contornos que el usuario crea.

Para la implementación de la clase *Modelo* se ha reutilizado el código realizado para la asignatura de Informática Gráfica del Grado de Ingeniería Informática. Esta representación del modelo consiste en almacenar:

- **vértices del modelo**, los vértices del modelo se almacenan en un array de tamaño  $N \times 3$ , donde  $N$  es el número de vértices del modelo. Cada vértice consta de 3 valores float, para representar sus coordenadas en el espacio 3D ( $x, y, z$ ).
- **caras del modelo**, las caras se almacenan como triángulos. Cada cara guarda tres índices a la tabla de vértices que hacen referencia a los vértices que forman el triángulo.

Podemos ver gráficamente la estructura usada en la Figura 6.4.

Además para cada modelo guardamos su Material y la Textura asociada al material.

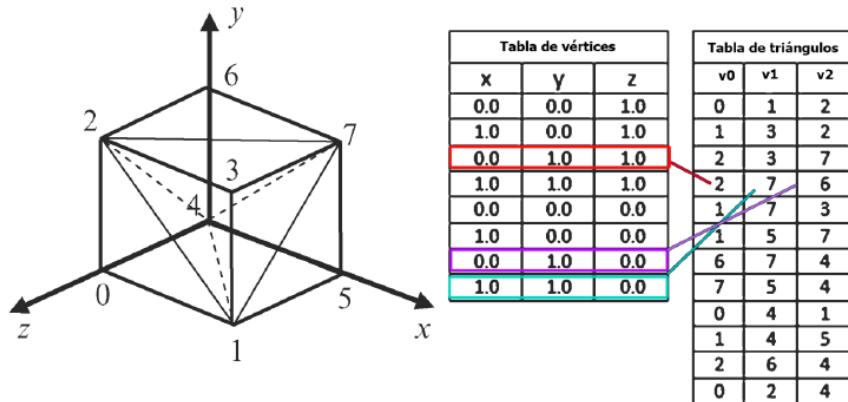


Figura 6.4: Tabla de vértices y triángulos

Para pintar el modelo haciendo uso de OpenGL es inmediato con la función *glDrawElements*:

```

1 glEnableClientState(GL_VERTEX_ARRAY);
2 glEnableClientState(GL_NORMAL_ARRAY);
3 glVertexPointer(3, GL_FLOAT, 0, vertices.data());
4 glNormalPointer(GL_FLOAT, 0, normalesVertices.data());
5 glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
6 glDrawElements(GL_TRIANGLES, caras.size() * 3,
    GL_UNSIGNED_INT, caras.data());

```

### Modelado de objetos por revolución

El modelado de objetos mediante revolución es una de las principales características del sistema. Para la implementación de esta característica se han tenido en cuenta las siguientes restricciones:

- El objeto que se generará tendrá una base circular.
- El eje principal del objeto será una línea recta. Esto es que el eje de rotación o recta directriz debe de formar un ángulo de 90° con la base del modelo. Esta restricción hace que el sistema sea incapaz de modelar objetos de revolución con un cuerpo curvo. Por la complejidad que conllevaría tener en cuenta que el eje sea curvo se ha obviado en esta implementación del sistema, pudiendo ser añadida esta funcionalidad en el futuro.

Lo primero que necesita conocer el sistema para generar el modelo es la

base de éste. Para simplificar esta tarea, se asume que la base es un círculo, esto reduce el número de parámetros desconocidos. Éste círculo puede tener una desformación debido a su proyección en la imagen por lo que puede resultar en una elipse. El usuario debe de aproximar esta elipse dibujando dos líneas rectas, la primera define el diámetro mayor y la segunda línea se arrastra hasta el final del diámetro menor. Esto forma una elipse en la imagen que coincide con la proyección del disco circular.

Con estos pasos el sistema ya conoce 3 de los 4 puntos necesarios para reconstruir la escena 3D mediante el algoritmo POSIT. El punto medio entre el primer punto y el segundo será el centro del eje de coordenadas  $(0, 0, 0)$ . El segundo punto será la proyección del punto  $(0.5, 0, 0)$  y el tercer punto la proyección del punto  $(0, 0, 0.5)$  (ver Figura 6.5).



Figura 6.5: Proyección de los puntos

Tras esto, el usuario puede definir el perfil del objeto hacia abajo, creando múltiples elipses que se ajusten al modelo como podemos ver en la Figura 6.6.



Figura 6.6: Definición del perfil

El último punto que el usuario define será el punto restante para calcular la proyección con el algoritmo POSIT, este punto de la imagen corresponde a la proyección del punto 3D  $(0, \text{altura}, 0.5)$ . La altura se puede calcular mediante una relación de las distancias, considerando que la distancia entre el primer punto y el segundo equivale en 3D a 1, podemos calcular la distancia entre el tercer punto definido y el último para conocer la altura del objeto.

Con los 4 puntos citados anteriormente podemos obtener una estimación de la posición y orientación de la cámara mediante el algoritmo POSIT explicado anteriormente. Para calcular la geometría del objeto se procede de la siguiente manera:

1. Para cada elipse generada, se calcula la distancia que hay con la elipse definida como la base del modelo. De esta manera obtenemos la coordenada  $y$  del vértice.

2. Para obtener la coordenada  $x$  lo haremos mediante el tamaño del eje mayor de la elipse, el cual el usuario puede ajustar mediante el scroll del ratón.

Una vez tenemos los puntos que conforman el perfil inicial (Figura 6.7) podemos proceder a crear el modelo mediante revolución. Se toma como eje de rotación el eje  $Y$  y si  $N$  es número lados longitudinales, se obtienen los puntos o vértices del sólido poligonal a construir multiplicando cada punto del perfil por  $N$  sucesivas transformaciones de rotación con respecto al eje  $Y$ . Se obtiene un conjunto de  $N \times M$  vértices agrupados en  $N$  perfiles (Figura 6.7).

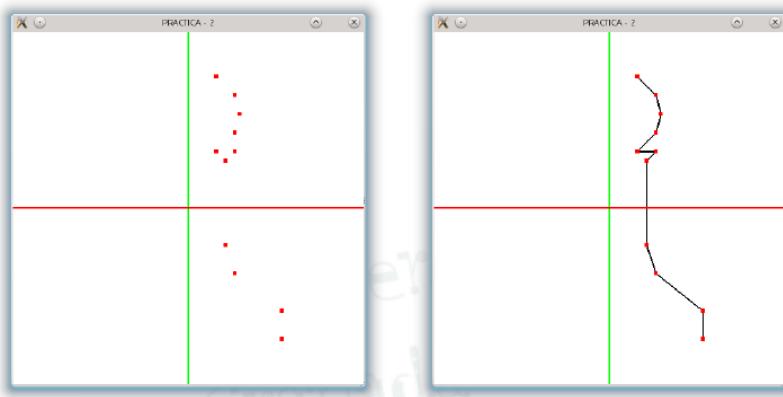


Figura 6.7: Perfil inicial

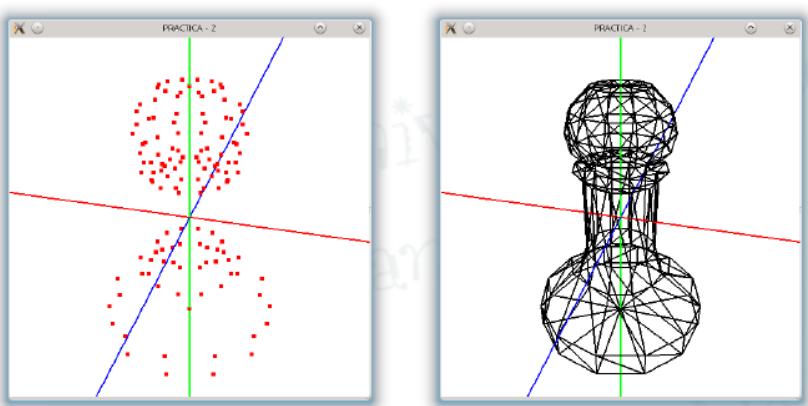


Figura 6.8: Sólido generado por revolución

Después de generar los puntos del modelo debemos de calcular la textura con la que se pintará el objeto. Para ello utilizaremos la misma imagen base

que el usuario importó al sistema. Las coordenadas de textura para cada vértice serán las proyecciones de estos puntos sobre la imagen.

#### 6.4.4. Exportación de los modelos 3D

La exportación de los modelos generados se hace en formato PLY. El formato PLY nos permite almacenar un modelo 3D especificando en formato ASCII la información del modelo. El sistema guarda:

- **Vértices del modelo:** vuelca en el archivo PLY la lista de vértices del modelo. Para cada vértice escribe sus coordenadas  $x$ ,  $y$  y  $z$ .
- **Caras del modelo:** vuelca en el archivo PLY la lista de caras del modelo. Para cada cara especifica el tamaño de la cara, en este caso siempre será 3 ya que las caras son triángulos, y los tres índices que apuntan a la lista de vértices que forman la cara.
- **Imagen de textura:** se guarda el nombre de la imagen de textura.
- **Coordenadas de texture:** para cada vértice se especifican sus coordenadas de textura en la imagen (texel).

Así el formato del fichero PLY exportado es el siguiente:

```

1 ply
2 format ascii 1.0
3 comment TextureFile textura.png
4 element vertex 17
5 property float x
6 property float y
7 property float z
8 property float s // u
9 property float t // v
10 element face 24
11 property list uchar uint vertex_indices
12 property list uchar float texcoord
13 end_header
14 0.707106 1 0 0.721154 0.278846
15 ...
16 lista de vertices
17 ...
18 0 0 0.532692 0.530769
19 3 1 3 4 6 0.721154 0.278846 0.561538 0.348077 0.561538 0.348077
20 ...
21 lista de caras
22 ...
23 3 16 8 11 6 0.532692 0.530769 0.353846 0.540385 0.555769 0.615385

```

Este archivo PLY puede ser leído por otro software de visualización o modelado 3D como Blender, 3DStudio, etc.

#### 6.4.5. Interfaz gráfica de usuario

El diseño de la interfaz gráfica de usuario del prototipo se puede ver en la Figura 6.9

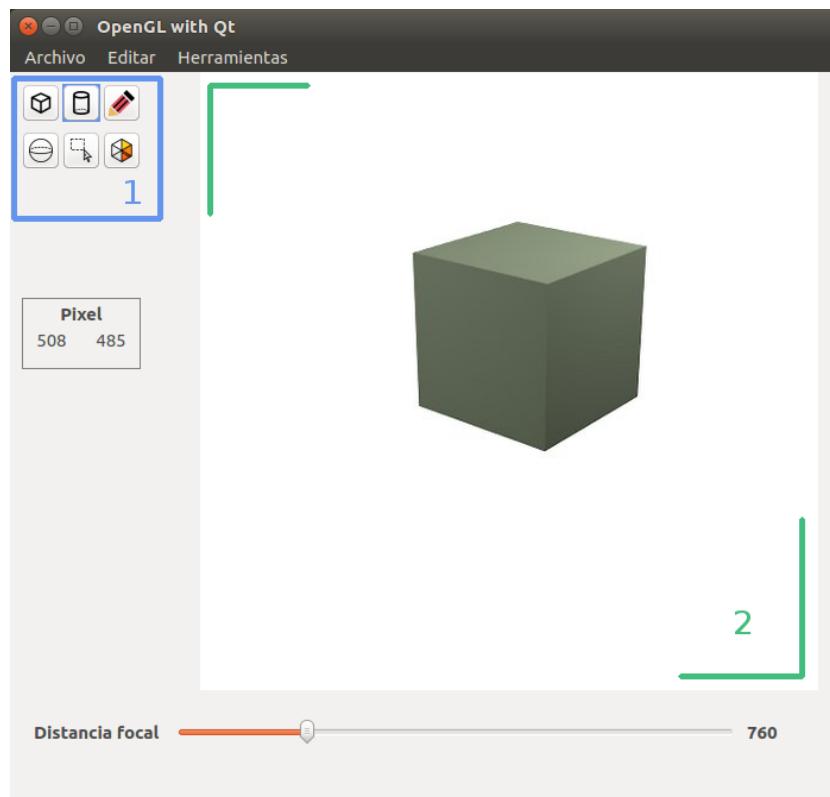


Figura 6.9: GUI prototipo

Se trata de una interfaz simple y fácil de usar. Desde el panel de herramientas de la izquierda podemos seleccionar las distintas herramientas de las que dispone el prototipo. Sobre el widget de la derecha el usuario podrá dibujar y trabajar con los modelos creados. También cuenta con un slider para ajustar la distancia focal de la cámara si así se desea. Para más información de cómo usar la interfaz de usuario se puede consultar el Apéndice A (**Manual de usuario**).



# Capítulo 7

## Pruebas

A continuación se muestran una serie de ejemplos creados con el sistema.

### 7.1. Modelado de objetos cuboides

En la Figura 7.1 se presenta un ejemplo de generación de un cubo a partir de la imagen de base de la izquierda. Como resultado final se obtiene el modelo trasladado y escalado que se puede observar a la derecha de la imagen.

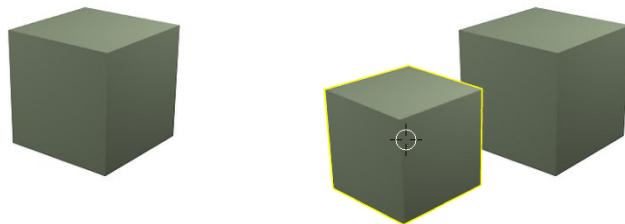


Figura 7.1: Modelado de un cubo.

El siguiente ejemplo se puede visualizar en la Figura 7.2. A la izquierda la imagen de base que se carga al sistema y a la derecha el modelo generado escalado sobre el eje  $y$ . De esta forma se puede generar una imagen diferente sin necesidad de conocimientos en retocado de fotografías.

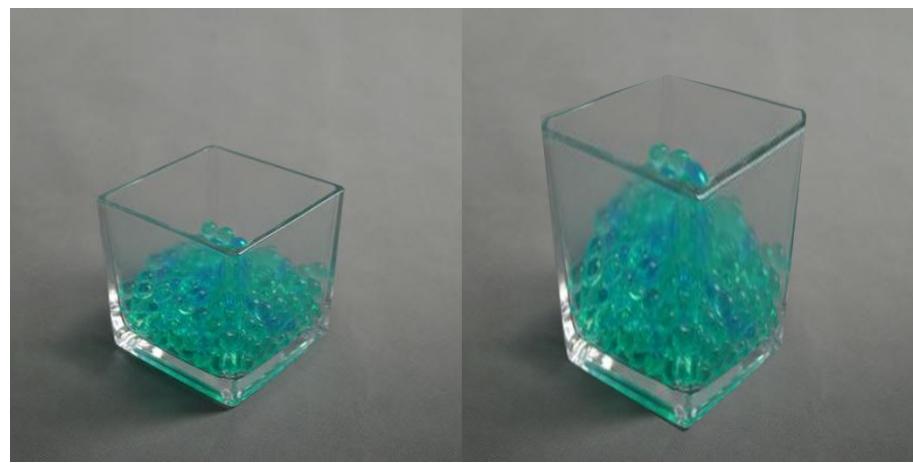


Figura 7.2: Modelado de un jarrón cuadrado de cristal.

En la Figura 7.3 se encuentra un ejemplo del modelado de una papelera con base cuadrada. A la izquierda la imagen de base y a la derecha la imagen resultante después de generar el modelo y realizar transformaciones sobre éste además de duplicarlo.



Figura 7.3: Modelado de una papelera cuadrada.

## **7.2. Modelado de objetos por revolución**

Podemos ver un ejemplo de generación de un objeto básico mediante revolución en la Figura 7.4. A la izquierda la imagen de base y a la derecha podemos ver el objeto resultante escalado.



Figura 7.4: Modelado de un cilindro.

En la Figura 7.5 se muestra un ejemplo de un cilindro generado por revolución con una textura más compleja. A la izquierda la imagen de base y a la derecha el modelo resultante tras aplicar algunas transformaciones sobre el mismo.



Figura 7.5: Modelado de una lata de tomate.

Un ejemplo muy parecido al anterior en la Figura 7.6. Se ha generado el modelo mediante revolución y se ha duplicado y transformado varias veces.



Figura 7.6: Modelado de una lata de cerveza.

En la Figura 7.7 se encuentra un ejemplo de un objeto generado por revolución con un perfil un poco más complejo. En la imagen de la derecha se puede observar como el modelo se ha creado perfectamente con la textura de la imagen.



Figura 7.7: Modelado de una pieza de ajedrez.

En la Figura 7.8 otro ejemplo más. A la izquierda la imagen de base y a la derecha el modelo generado trasladado y escalado sobre la imagen.



Figura 7.8: Modelado de una botella de crial.

### 7.3. Exportación de modelos 3D

Los modelos generados con el sistema pueden ser exportados en formato PLY con la textura.

Tras generar el modelo por revolución que se muestra en la Figura 7.9 podemos seleccionar la opción de exportar modelo (Figura 7.10).

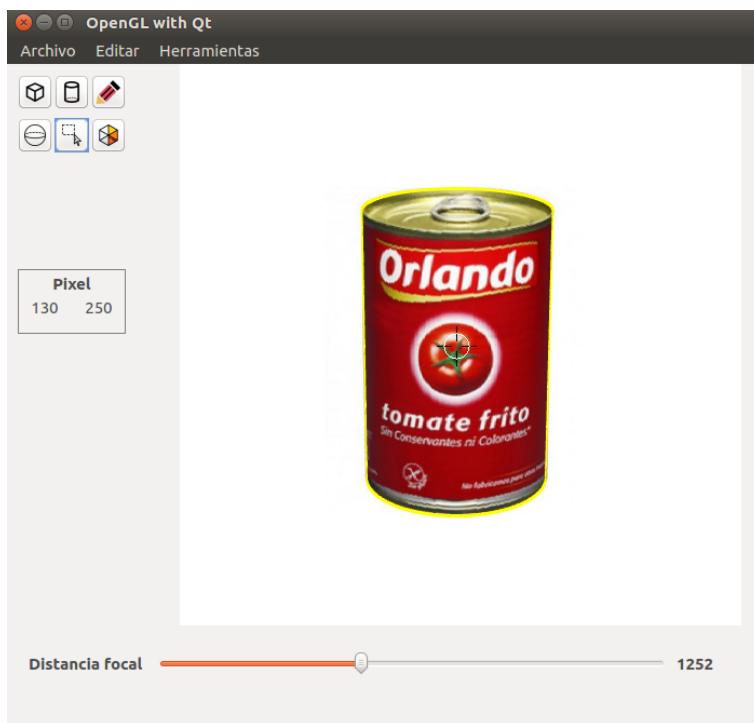


Figura 7.9: Modelo generado de una lata de tomate con el software.

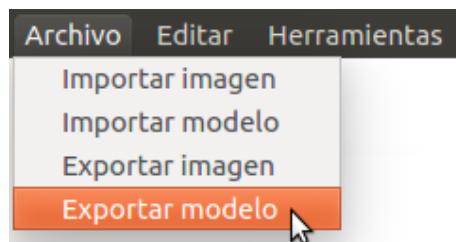


Figura 7.10: Exportar modelo

El archivo PLY guardado tendrá un formato parecido al de la Figura 7.11.

```

1  ply
2  format ascii 1.0
3  comment TextureFile lata.png
4  element vertex 1390
5  property float x
6  property float y
7  property float z
8  property float s // u
9  property float t // v
10 element face 2768
11 property list uchar uint vertex_indices
12 end_header
13 0.486663 1.4643 -1.81057e-07 0.659615 0.259615
14 0.486663 1.4643 -1.81057e-07 0.659615 0.259615
15 0.486663 0 -1.81057e-07 0.648154 0.728846
16 0.486663 0 -1.81057e-07 0.648154 0.728846
17 0.486584 1.4643 -0.00907925 0.659615 0.261538
18 0.486584 1.4643 -0.00907925 0.659615 0.261538
19 0.486584 0 -0.00907925 0.648077 0.730769
20 0.486584 0 -0.00907925 0.648077 0.730769
21 0.486346 1.4643 -0.0181554 0.659615 0.261538
22 0.486346 1.4643 -0.0181554 0.659615 0.261538
23 0.486346 0 -0.0181554 0.648077 0.732692
24 0.486346 0 -0.0181554 0.648077 0.732692
25 0.48595 1.4643 -0.0272256 0.659615 0.261538
26 0.48595 1.4643 -0.0272256 0.659615 0.261538
27 0.48595 0 -0.0272256 0.648077 0.732692
28 0.48595 0 -0.0272256 0.648077 0.732692
29 0.485394 1.4643 -0.0362873 0.659615 0.263462
30 0.485394 1.4643 -0.0362873 0.659615 0.263462
31 0.485394 0 -0.0362873 0.648077 0.734615

```

Figura 7.11: Archivo PLY exportado

Los modelos exportados mediante la aplicación pueden ser importados con otros software de visualización o modelado en 3D. Por ejemplo en la Figura 7.12 podemos ver el modelo creado anteriormente importado en MeshLab.

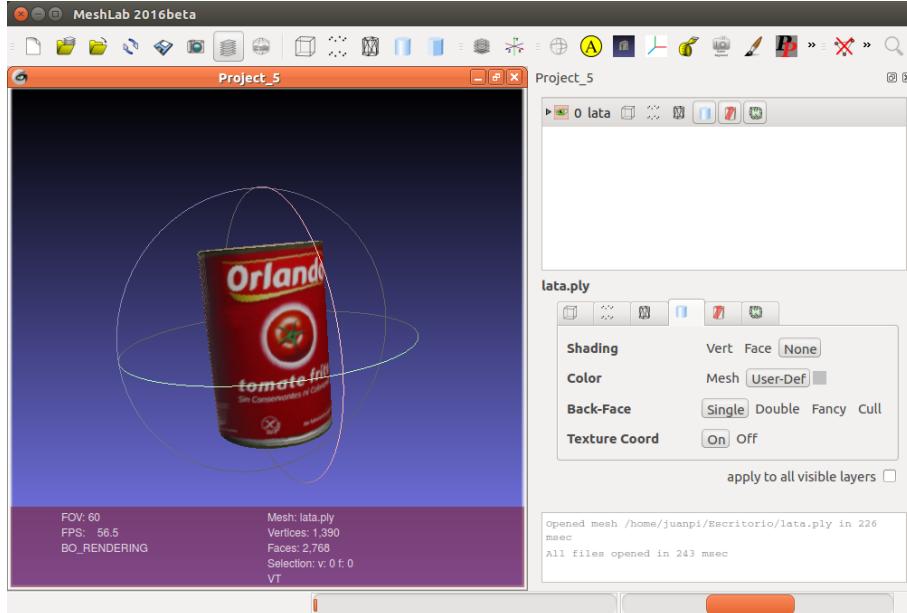


Figura 7.12: Archivo PLY importado con MeshLab



## Capítulo 8

# Conclusiones y trabajo futuro

### 8.1. Conclusiones

Tras el desarrollo de las distintas partes de este proyecto he podido introducirme en el desarrollo de aplicaciones relacionadas con la **informática gráfica**. Se ha comprendido la potencia de las librerías OpenCV y OpenGL para su uso en aplicaciones profesionales, además del framework Qt para la creación de interfaces gráficas de usuario entre otras funcionalidades.

Entre las metas iniciales del proyecto se encontraban algunas de **gran complejidad** como el modelado de formas complejas o la detección óptima del contorno del objeto en condiciones no ideales. Durante el desarrollo del proyecto se ha comprendido que alcanzar estas metas no era posible debido a que requerían más tiempo y conocimientos de los que no disponía.

El sistema desarrollado finalmente se ajusta a los objetivos planteados en un inicio para el proyecto y puede ser la base de una **aplicación profesional** en un futuro. Se ha llegado a la conclusión de que puede resultar más útil el modelado de objetos desde la aplicación para su posterior exportación que el objetivo de modificar la imagen desde el propio sistema. En comparación con el software 3-Sweep, este proyecto puede acabar siendo una alternativa válida para el modelado en tres dimensiones a partir de una sola imagen.

El uso de tecnologías libres y que el software desarrollado tiene una **licencia libre** otorga la posibilidad de que otros desarrolladores aporten sus

conocimientos al proyecto para formar una comunidad de desarrollo entorno a él.

## **8.2. Trabajo futuro**

El sistema desarrollado podría sufrir muchos cambios en el futuro. Los objetivos futuros si se continúa con el desarrollo de la aplicación serán:

1. **Analizar, diseñar e implementar** una técnica para el modelado de objetos con formas más complejas.
2. **Mejorar el modelado de objetos por revolución** teniendo en cuenta que se pueden crear objetos con un eje de revolución curvo.
3. **Ampliar los formatos soportados.** La posibilidad de importar imágenes en cualquier formato puede ampliar el uso de la aplicación a muchos más usuarios. También se tiene en cuenta la posibilidad de exportar los modelos generados a un formato más usado como puede ser OBJ.
4. **Mejorar la interfaz gráfica de usuario.** Diseñar e implementar una interfaz gráfica de usuario acorde con una aplicación de carácter profesional.

# Bibliografía

- [1] Steve Marschner Peter Shirley, Michael Ashikhmin. *Fundamentals of Computer Graphics*. AK Peters/CRC Press, 2009.
- [2] William Johnston and Wing Nip. Geometric analysis, visualization, and conceptualization of 3d image data. <http://froggy.lbl.gov/papers/Reports/LBL.35329.html>.
- [3] Curless B. Seitz S. M. *A comparison and evaluation of multi-view stereo reconstruction algorithms*. Conference on Computer Vision and Pattern Recognition, 2006 IEEE Computer Society, Vol. 1, 519-528, 2006.
- [4] François Chaumette Andrew I. Comport, Éric Marchand. *Robust model-based tracking for robot vision*. IEEE/RSJ Int. Conference on Intelligent Robots and Systems, IROS'04, Sendai, Japan, September 2004.
- [5] P.Eisert P. Fechteler. *Adaptive Color Classification for Structured Light Systems*. Image Processing Department, Einsteinufer 37, D-10587 Berlin, Germany.
- [6] Th. Heinrich A. Kirchner. *Model based detection of road boundaries with a laser scanner*. Proc.IEEE Int. Conf. on Intelligent Vehicles, Vol. 1, Stuttgart, Germany, pags. 93-98, 1998.
- [7] E.P. Kroktov. *Active Computer Vision by Cooperative Focus and Stereo*. Springer-Verlag, NY, 1989.
- [8] N. Ahuja. *Active Stereo: Integrating Disparity, Vergence, Focus, Aperture and Calibration for Surface Estimation*. IEEE Trans. on Pattern Ana. & Mach. Intell. Vol. 15, No. 10, Octubre 1993.
- [9] P. Cignoni C. Rocchini. *A low cost 3D scanner based on structured Light*. Istituto di Scienze e Tecnologie dell'Informazione (ISTI), 2001.
- [10] Thao Chen & Zhe Zhu & Ariel Shamir & Shi-Min Hu & Daniel Cohen-Or. 3-sweep: Extracting editable objects from a single photo.

- [11] Daniel F. DeMenthon and Larry S. Davis. Model-based object pose in 25 lines of code.
- [12] Juan Pablo Porcel Porcel. Repositorio del proyecto 3dphotomodeling. <https://github.com/JPPorcel/3DPhotoModeling>.
- [13] Matt Young. The pinhole camera. <http://inside.mines.edu/~mmyoung/PHCamera.pdf>.
- [14] Richard Szeliski. Computer vision: Algorithms and applications. <http://szeliski.org/Book/>, 2010.
- [15] W. Carithers D. Hearn, P. Baker. *Computer Graphics with Open GL*. Prentice Hall, 2010.
- [16] Adrian Kaehler Gary Bradski. *Learning OpenCV*. O'Reilly Media, Inc., 2008.
- [17] Javir Barandiaran. Posit tutorial with opencv. <http://code.opencv.org/projects/opencv/wiki/Posit>.
- [18] Foley & Van Dam & Feiner & Hughes. *Computer Graphics: Principles and Practice in C*. Addison-Wesley, 1997.

# Apéndice A

## Manual de usuario

En este apéndice se pretende hacer conocer al usuario final del software cómo se maneja el mismo.

### A.1. Interfaz Gráfica de Usuario

En la Figura A.1 podemos ver la interfaz del prototipo del sistema.

En el cuadro verde la Figura A.1 se encuentra nuestro GLWidget. En este widget se dibuja la imagen de base que carga el usuario y las trazas que realiza sobre ella. También se dibujarán aquí los modelos 3D generados.

El cuadro azul de la Figura A.1 se encuentra el menú de herramientas:

- **Crear cuboide:**  Herramienta para crear el contorno de un objeto cuboide, cuando se pulsa este botón se cambia al modo cuboide automáticamente y el usuario puede crear un objeto cuboide desde el widget de dibujado.
- **Crear objeto por revolución:**  Herramienta para crear el contorno de un objeto de revolución, cuando se pulsa este botón se cambia al modo revolución automáticamente y el usuario puede crear el objeto desde el widget de dibujado.
- **Crear esfera:**  Herramienta para crear una esfera.

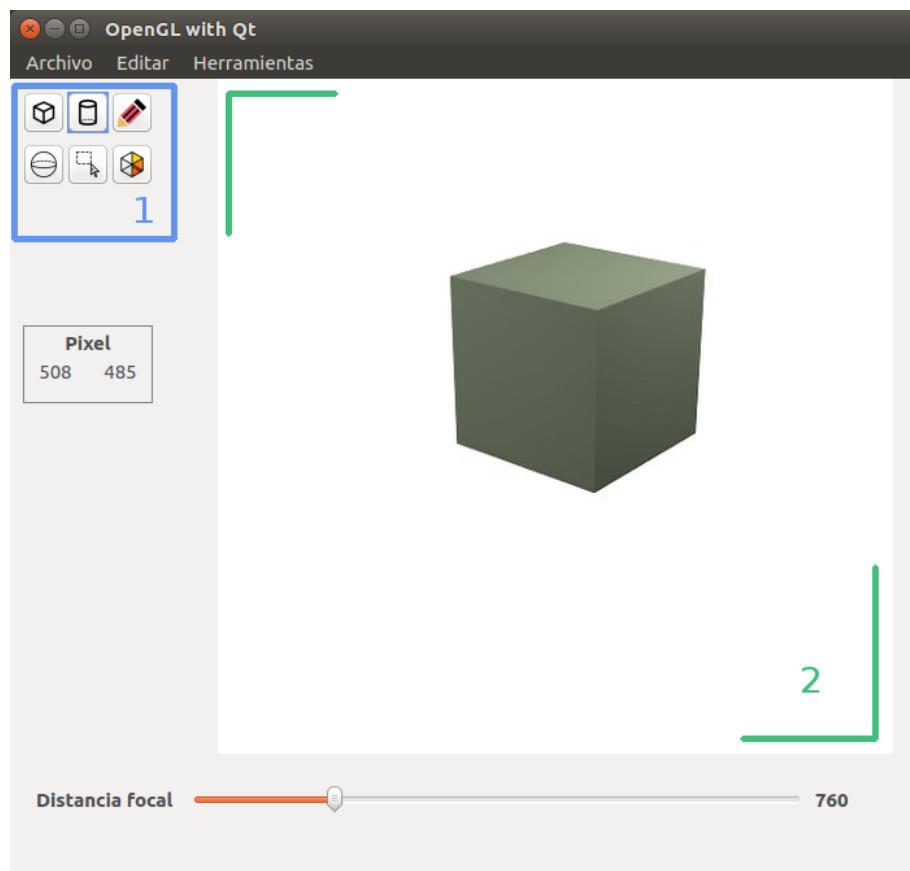


Figura A.1: Interfaz

- **Selección:**  Herramienta de selección. Mediante esta herramienta se pueden seleccionar los distintos modelos creados en la imagen desde el widget de dibujado.
  
- **Generar modelo en 3D:**  Herramienta para generar la figura que se ha dibujado en tres dimensiones. Tras pulsar sobre este botón, el sistema calcula el modelo 3d y cambia al modeo selección.
  
- **Recalcular textura:**  Herramienta extraer de nuevo la textura del modelo de la imagen.

## A.2. Importar una imagen al sistema

Mediante la opción de **Importar imagen** en el menú de **Archivo** (ver Figura A.2) se nos abrirá una venta para seleccionar un archivo JPG (ver Figura A.3). La imagen será cargada por el sistema y dibujada en pantalla.

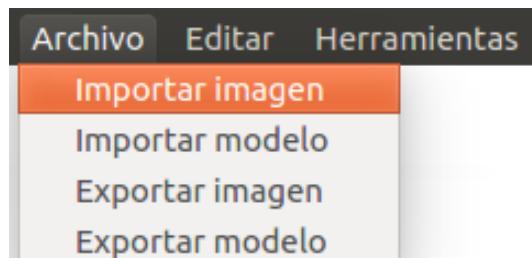


Figura A.2: Archivo - Importar imagen

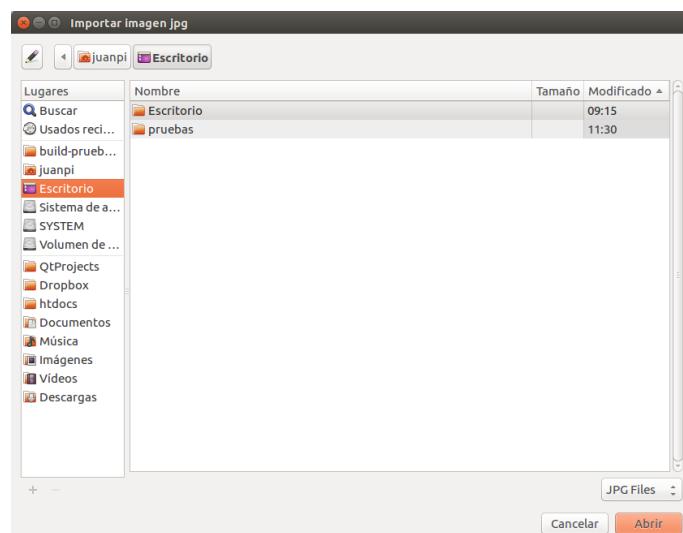


Figura A.3: Ventana de selección de archivos

## A.3. Crear un objeto mediante revolución

Pasos para crear un objeto por revolución desde la GUI:



1. Seleccionar la herramienta de creación de objetos por revolución .
2. Definir el eje mayor de la elipse que forma la base principal del objeto (Figura A.4). Para ello con el ratón pulsaremos en el inicio del eje (1)

y de nuevo en el final del eje (2). Podemos realizar esta acción con la ayuda de las teclas del teclado SHIFT y CTRL para dibujar la traza entre los dos puntos y ajustar a un ángulo recto respectivamente.

3. Definir el eje menor de la elipse (Figura A.4). Tras definir el eje mayor, llevamos el ratón al final del eje menor de la elipse para crear la base del modelo (3).

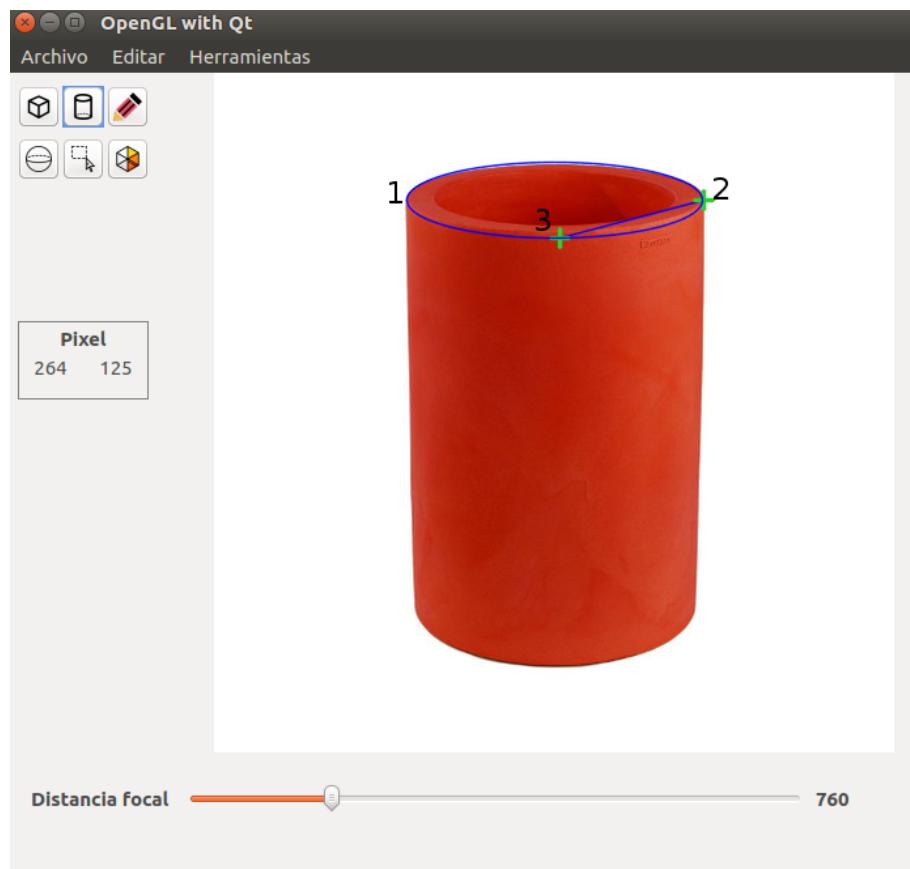


Figura A.4: Primeros pasos para crear un objeto por revolución

4. Arrastrar el ratón hasta el final del perfil y ajustar mediante el scroll del ratón el tamaño de la nueva elipse (Figura A.5). Este paso se puede repetir tantas veces como se quiera para definir el perfil del objeto.

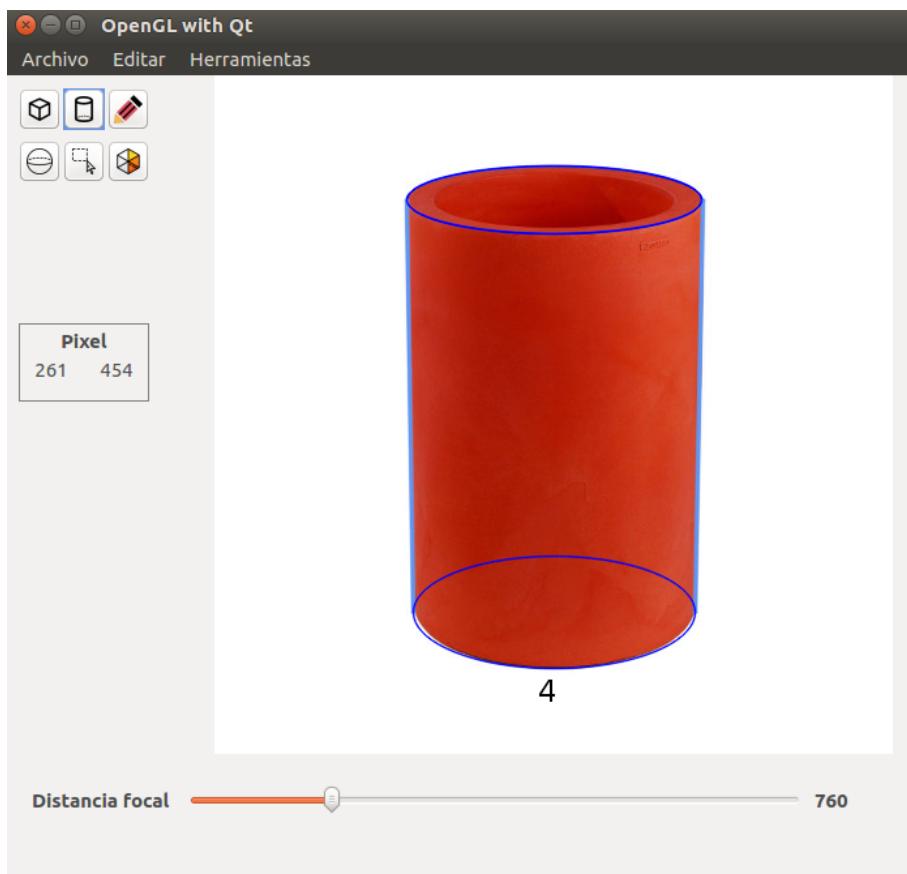


Figura A.5: Definir el perfil del objeto

5. Tras esto, pulsamos ESC y salimos del modo de crear el perfil. Pulsemos sobre  para generar el modelo 3D del perfil creado (Figura A.6).

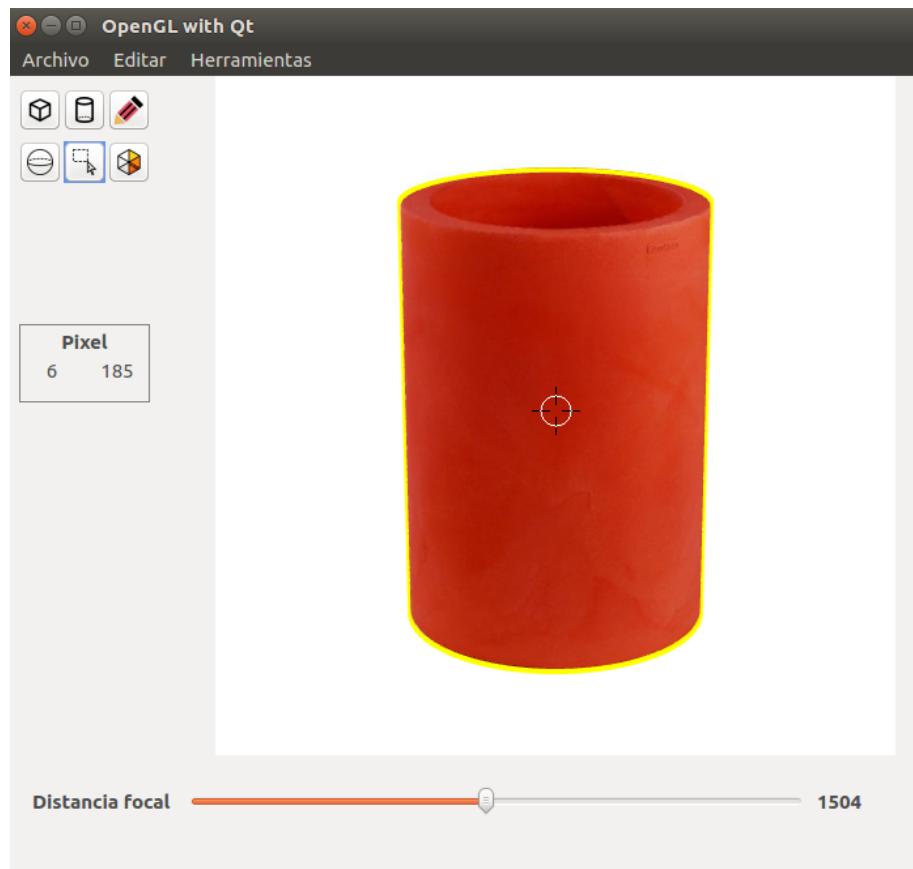


Figura A.6: Modelo 3D del perfil definido

## A.4. Crear un objeto cuboide

Pasos para crear un objeto por revolución desde la GUI:

1. Seleccionar la herramienta de creación de objetos cuboides .
2. Definir la forma del objeto (Figura A.7). Definimos la base con dos trazas del ratón pinchando sobre las esquinas del cubo, de (1) a (2) y de (2) a (3). Con la tercera traza, de (3) a (4) definimos la altura del modelo.

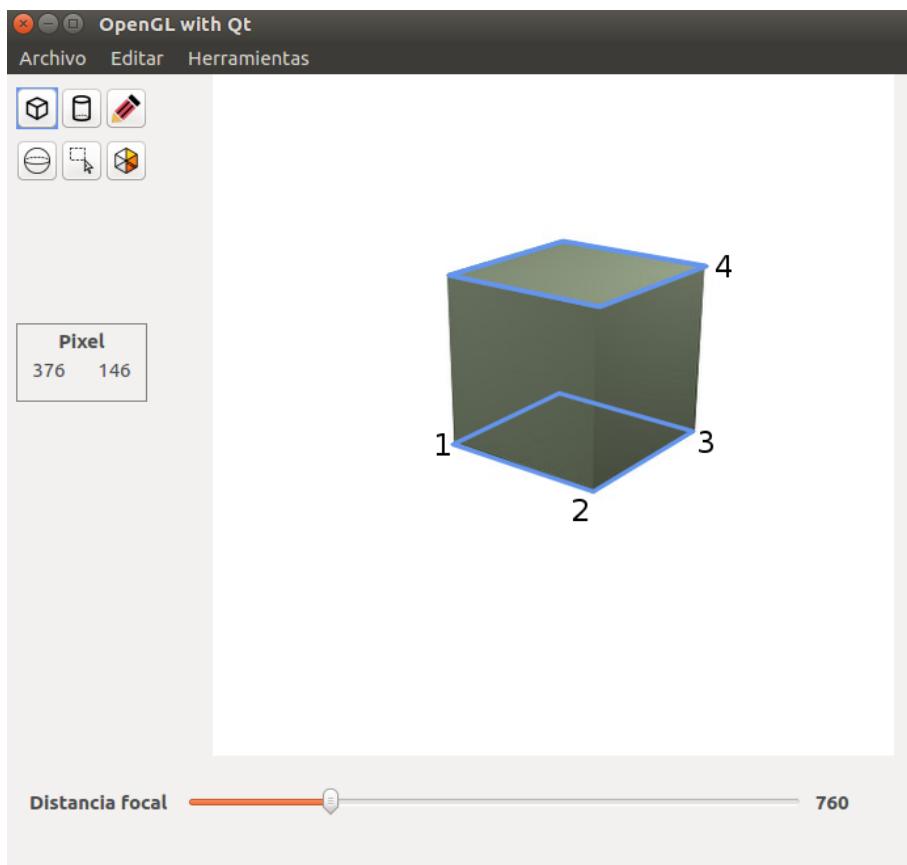


Figura A.7: Definir la forma del cubo

3. Pulsamos ESC y pulsamos sobre  para generar el modelo 3D (ver Figura A.8).

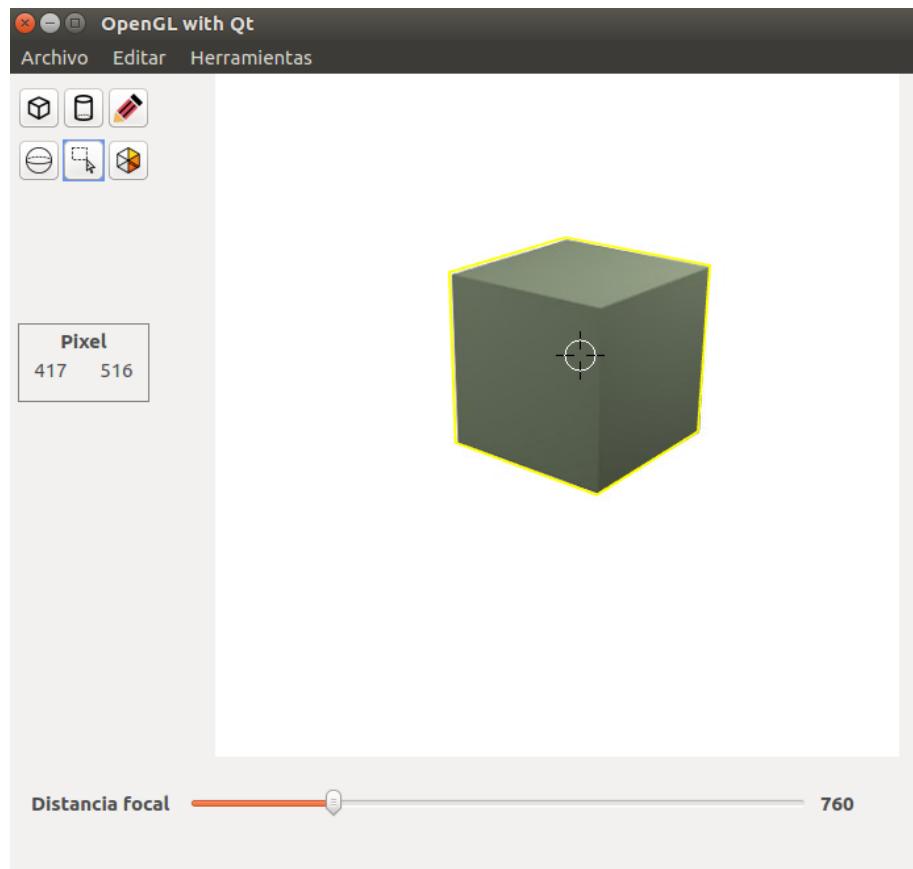


Figura A.8: Modelo 3D

## A.5. Transformar modelo

Podemos realizar varias transformaciones sobre los modelos generados:

- **Rotar:** pulsando la tecla del teclado *r* y con un modelo seleccionado, entramos al modo de rotación. Podemos usar las teclas *x*, *y* y *z* para cambiar el eje de rotación sobre el que rotará el modelo. Con el movimiento del ratón se aumentará y disminuirá el ángulo de rotación (ver Figura A.9).

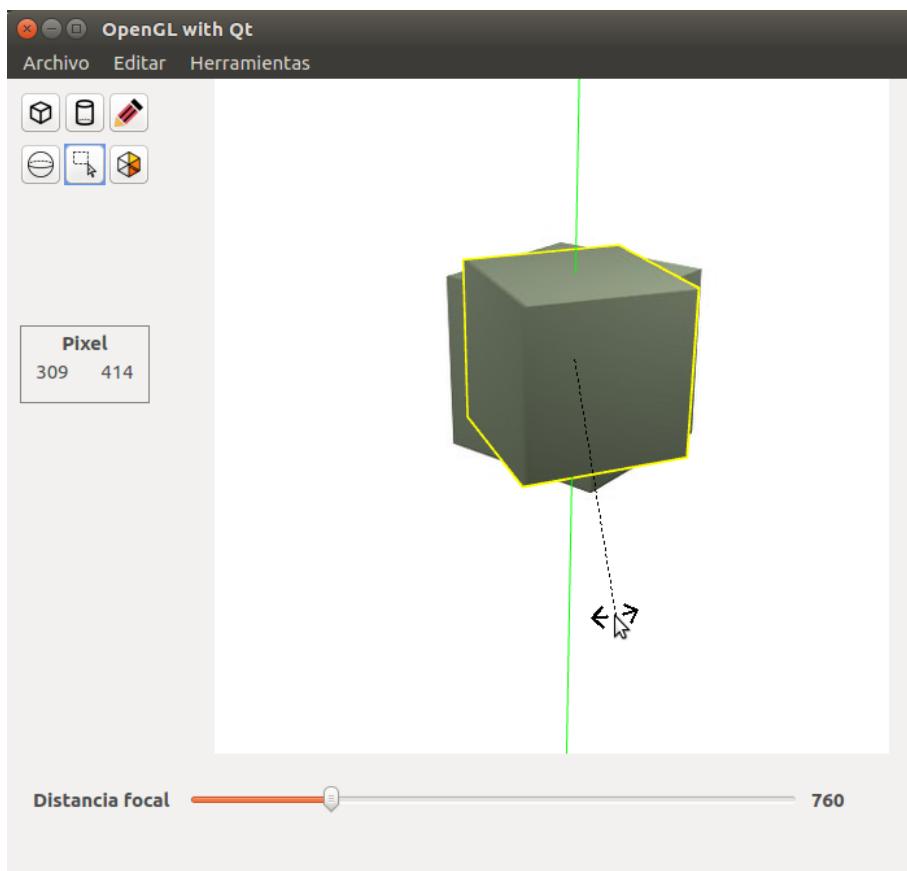


Figura A.9: Rotar un modelo

- **Escalar:** pulsando la tecla del teclado *s* y con un modelo seleccionado, entramos al modo de escalado. Podemos usar las teclas *x*, *y* y *z* para cambiar el eje de escalado sobre el que se escalará el modelo. Con el movimiento del ratón se aumentará y disminuirá el factor de escalado (ver Figura A.10).

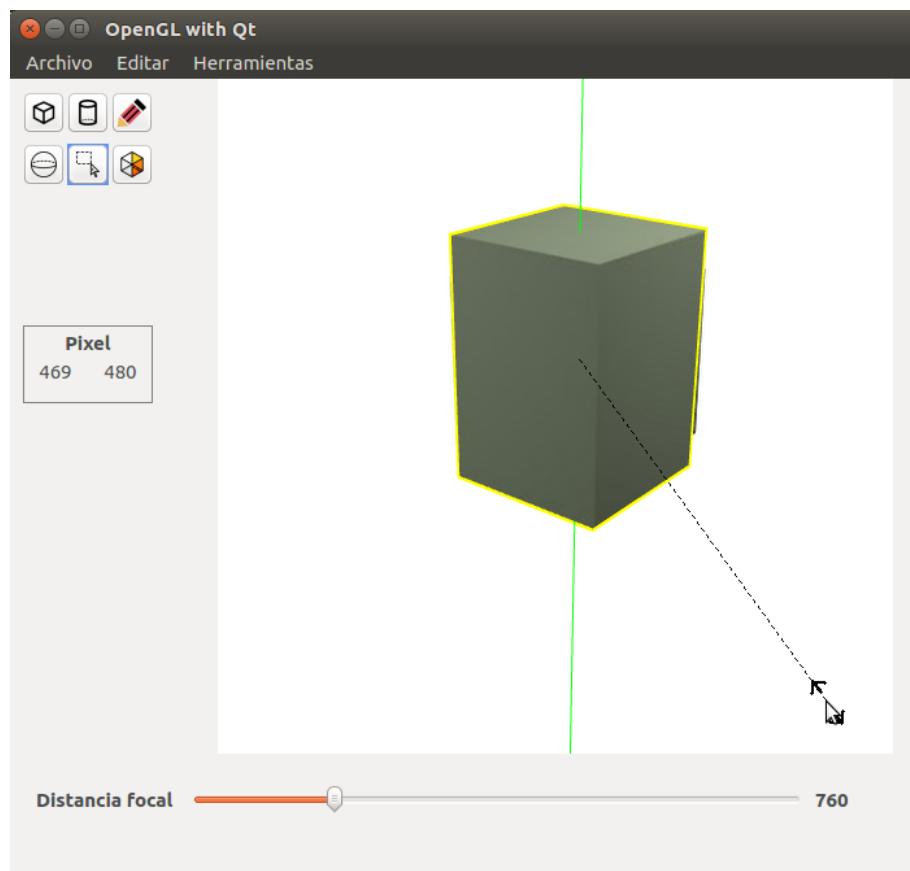


Figura A.10: Escalar un modelo

- **Trasladar:** pulsando la tecla del teclado  $t$  y con un modelo seleccionado, entramos al modo de translación. Podemos usar las teclas  $x$ ,  $y$  y  $z$  para cambiar el eje de translación sobre el que se traslada el modelo. Con el movimiento del ratón se aumentará y disminuirá el factor de translación (ver Figura A.11).

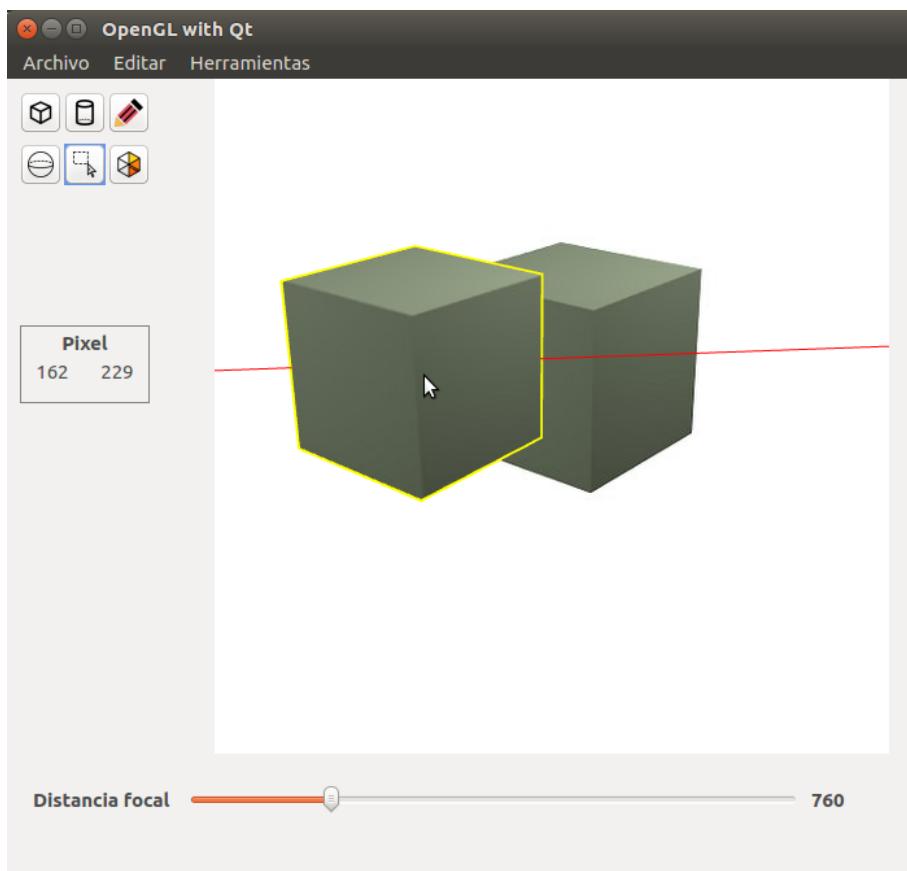


Figura A.11: Trasladar un modelo

- **Duplicar:** pulsando las tecla del teclado  $CTRL+c$  y de nuevo  $CTRL+v$ , duplicamos el modelo seleccionado (ver Figura A.12).

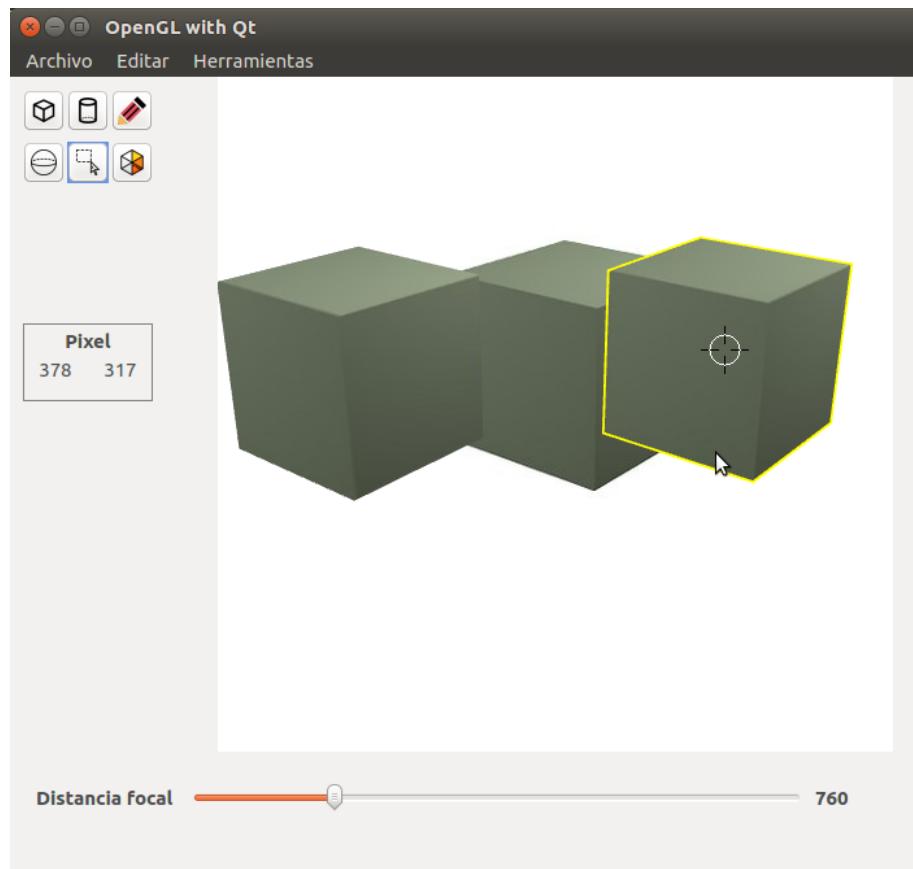


Figura A.12: Duplicar un modelo

## A.6. Exportar imagen en PNG

Mediante la opción de ***Exportar imagen*** en el menú de ***Archivo*** (ver Figura A.13) se nos abrirá una ventana para seleccionar dónde queremos guardar la imagen resultante. La imagen se almacenará en el directorio y con el nombre seleccionados en formato PNG.

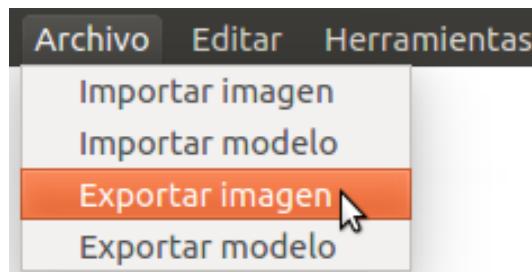


Figura A.13: Archivo - Exportar imagen

## A.7. Exportar modelo en PLY

Mediante la opción de ***Exportar modelo*** en el menú de ***Archivo*** (ver Figura A.14) se nos abrirá una ventana para seleccionar dónde queremos guardar el modelo. El modelo se almacenará en el directorio y con el nombre seleccionados en formato PLY.

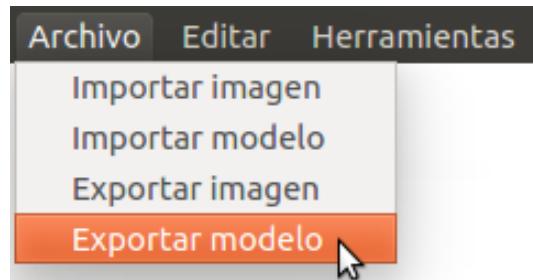


Figura A.14: Archivo - Exportar modelo



# **Lista de figuras**

1.	Reconstrucción 3D de un objeto. A la izquierda: la imagen de base que el usuario carga en la aplicación. A la derecha: la imagen resultante tras generar el modelo y realizar transformaciones sobre este (escalar y trasladar). . . . .	4
2.	3D reconstruction of an object. On the left: the image that the user loads into the application. Right: the resulting image after generating the model and make transformations on it (scale and traslate). . . . .	6
1.1.	Representación del contorno de un objeto. . . . .	2
1.2.	Modelo basado en enumeración espacial. Fuente: [2] . . . . .	3
1.3.	Izquierda: barrido translacional. Derecha: barrido rotacional. .	4
1.4.	Representación de una superficie curva. . . . .	4
1.5.	Operaciones booleanas en CSG. . . . .	5
1.6.	Funcionamiento del escáner 3-D de luz estructurada no sincronizado. Imagen: Taubin Lab. Fuente: Universidad Brown. . . . .	7
1.7.	Ejemplo de uso del software 3-Sweep . . . . .	10
1.8.	(a) Imagen de entrada (b) extracción de ejes (c) modelado de un componente del objeto (d) el modelo completo (e) transformaciones sobre el modelo. Imagen extraída de [10] . . . . .	11
1.9.	Objetos modelados con 3-Sweep. Arriba la imagen de base. Abajo el modelo extraído de la imagen. [10] . . . . .	11
3.1.	Diagrama de Gantt iteraciones 1 y 2 . . . . .	25
3.2.	Diagrama de Gantt iteraciones 3 y 4 . . . . .	28
4.1.	Diagrama de casos de uso . . . . .	32
4.2.	Diagrama de actividad para el caso de uso 1 . . . . .	43

4.3. Diagrama de actividad para el caso de uso 2 . . . . .	44
4.4. Diagrama de actividad para el caso de uso 3 . . . . .	44
4.5. Diagrama de actividad para el caso de uso 4 . . . . .	45
4.6. Diagrama de actividad para el caso de uso 5 . . . . .	45
4.7. Diagrama de actividad para el caso de uso 8 . . . . .	46
4.8. Diagrama de actividad para el caso de uso 9 . . . . .	46
 5.1. Arquitectura MVC . . . . .	47
5.2. Diagrama de clases UML . . . . .	48
5.3. Modelo de cámara pinhole . . . . .	49
5.4. Distorsión de la imagen alterando la distancia focal . . . . .	51
5.5. Con 4 puntos no coplanares el algoritmo POSIT puede calcular la pose 3D . . . . .	52
5.6. El usuario define la elipse base mediante dos trazas, para definir el eje mayor y el eje menor. . . . .	53
5.7. La última traza debe de terminar en la base del modelo. . . . .	54
5.8. El perfil del objeto puede ser más complejo. . . . .	54
5.9. Cuboides . . . . .	55
5.10. Definición de un cubo mediante la técnica de modelado de objetos cuboides. . . . .	56
 6.1. Pipeline de OpenGL . . . . .	59
6.2. QtCreator . . . . .	60
6.3. Jerarquía de clases para almacenar los modelos . . . . .	68
6.4. Tabla de vértices y triángulos . . . . .	70
6.5. Proyección de los puntos . . . . .	71
6.6. Definición del perfil . . . . .	72
6.7. Perfil inicial . . . . .	73
6.8. Sólido generado por revolución . . . . .	73
6.9. GUI prototipo . . . . .	75
 7.1. Modelado de un cubo. . . . .	77
7.2. Modelado de un jarrón cuadrado de cristal. . . . .	78
7.3. Modelado de una papelera cuadrada. . . . .	78

---

7.4. Modelado de un cilindro . . . . .	79
7.5. Modelado de una lata de tomate. . . . .	79
7.6. Modelado de una lata de cerveza. . . . .	80
7.7. Modelado de una pieza de ajedrez. . . . .	80
7.8. Modelado de una botella de crital. . . . .	81
7.9. Modelo generado de una lata de tomate con el software. . . . .	82
7.10. Exportar modelo . . . . .	82
7.11. Archivo PLY exportado . . . . .	83
7.12. Archivo PLY importado con MeshLab . . . . .	83
A.1. Interfaz . . . . .	90
A.2. Archivo - Importar imagen . . . . .	91
A.3. Ventana de selección de archivos . . . . .	91
A.4. Primeros pasos para crear un objeto por revolución . . . . .	92
A.5. Definir el perfil del objeto . . . . .	93
A.6. Modelo 3D del perfil definido . . . . .	94
A.7. Definir la forma del cubo . . . . .	95
A.8. Modelo 3D . . . . .	96
A.9. Rotar un modelo . . . . .	97
A.10. Escalar un modelo . . . . .	98
A.11. Trasladar un modelo . . . . .	99
A.12. Duplicar un modelo . . . . .	100
A.13. Archivo - Exportar imagen . . . . .	100
A.14. Archivo - Exportar modelo . . . . .	101



# Índice de tablas

3.1. Listado de tareas detalladas (I) . . . . .	26
3.2. Listado de tareas detalladas (II) . . . . .	27
4.1. Descripción del actor usuario . . . . .	32



