2018

APLICACIÓN MÓVIL DE SUPERVISIÓN REMOTA Y TRAZABILIDAD, A TRAVÉS DE MAQUETA VIRTUAL

ROSAS ALBRECHT, CARLOS ANDRÉS

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE ELECTRÓNICA

VALPARAÍSO - CHILE

“Aplicación móvil de supervisión remota y trazabilidad, a través de maqueta virtual”

CARLOS ANDRÉS ROSAS ALBRECHT

MEMORIA DE TITULACIÓN PARA OPTAR AL GRADO DE INGENIERÍA CIVIL ELECTRÓNICA MENCIÓN TELECOMUNICACIONES

PROFESOR GUÍA: AGUSTÍN J. GONZÁLEZ V.

TUTOR CORREFERENTE: EDMUNDO G. CASAS C.

OCTUBRE – 2017

RESUMEN

En la actualidad la mayoría de las plantas industriales poseen sistemas de monitoreo y diagnóstico centralizados. Sin embargo, en caso de algún incidente, los operadores en terreno deben recurrir a diversas fuentes de información, muchas veces de difícil acceso, para dar respuesta a la emergencia. Por lo anterior existe la necesidad de disponibilizar y centralizar esta información, disminuyendo así los tiempos de búsqueda y mejorando la capacidad de respuesta dentro de la compañía.

El presente trabajo describe el diseño, implementación y funcionamiento de la aplicación TR3, cuyo objetivo es centralizar información relevante de equipos de una planta industrial en una maqueta virtual para consultar y facilitar la toma de decisiones del personal en terreno.

El contenido de este documento va desde la elección y descripción de la plataforma de desarrollo, al diseño y desarrollo de la aplicación TR3, incluyendo las mejoras realizadas en el sistema. Se hace énfasis en la comprensión del proceso de renderizado, explicando su funcionamiento y como se pueden generar cuellos de botella en la comunicación del CPU con el GPU. Esto permite optimizar la aplicación y obtener el mayor provecho a los recursos disponibles.

Es importante tener presente que la aplicación procura aumentar el rendimiento y la productividad, optimizando procesos y recursos, manteniendo intactos los estándares de calidad de la compañía.

Finalmente cabe señalar que el desarrollo y la implementación de este proyecto abre las puertas a nuevos desafíos, como pasar de una aplicación que recoger y despliega información a poder actuar sobre las variables que controlan los elementos de la planta industrial.

ABSTRACT

Currently, most industrial plants have centralized monitoring and diagnostic systems. However, in the event of an incident, the groundworkers in the field must rely upon several sources of information, often difficult to access, to react to the emergency. Therefore, there’s a need to provide and centralize this information, reducing response time and facilitating the operations within the company.

This paper describes the design, implementation and operation of the TR3 application, which aim is to centralize relevant information from industrial plant equipment in a virtual model to consult and facilitate the personnel decision making in the field.

The work contained in this document goes from the choice and description of the development platform, the design and development of TR3 application to the implemented improvements in the system. Emphasis is placed on the understanding of the rendering process, explaining its operation and detecting potential bottlenecks in CPU-GPU communication, to improve the application and optimize available resources.

It is important to understand or realize that the application seeks to increase performance and productivity, optimizing processes and resources, keeping the company's quality standards intact.

Finally it should be noted that the development and implementation of this project opens the doors to new challenges such as going from collecting and displaying information to being able to act over the variables that control the different elements of the industrial plant.

GLOSARIO

Término	Definición
API	Interfaz de programación de aplicaciones
Framework	Entorno de desarrollo
DLL	Biblioteca de enlace dinámico. En el contexto actual se utiliza para agregar funciones que el entorno de desarrollo no trae incorporados.
SQL	Lenguaje de consultas estructurado para base de datos
Query	Consulta a base de datos
Open source	Código abierto
CPU	Unidad central de procesamiento
GPU	Unidad de procesamiento gráfico
Plugin	Complementos para aplicaciones
QR	Módulo para almacenar información en una matriz de puntos.
KKS	Código único de un elemento en secuencia de caracteres alfanuméricos
FBX	Formato de archivo 3D
Asset	Recurso disponible para ser utilizado por la aplicación.
Layout	Estructura de cuadricula que divide espacios.
Layer	Capas para clasificar y trabajar objetos por separado.
Gameobject	Objeto básico dentro de Unity3D

Elemento	Representación dentro de la aplicación de un equipo o maquinaria física
Elemento interactivo	Elemento dentro de la aplicación con el cual el usuario puede interactuar.
Prefab	Conjunto de atributos pre especificados para posterior uso o clonación.
Script	Archivo de ordenes por lotes que el CPU puede interpretar.
Attach	Adjuntar o agregar a otro objeto.
Frame	Un ciclo de procesamiento
Box collider	Elemento cúbico que permite detectar colisiones.
Raycast	Rayo invisible que permite interactuar, medir o detectar objetos.
Input touch	Entrada táctil
Touch events	Eventos táctiles
Gestures	Entradas táctiles que representan un movimiento específico o gesto.
Tap	Tocar la pantalla.
Pan	Deslizar el dedo en la pantalla.
Mesh	Malla de puntos o vértices y bordes que describen superficies y forman estructuras tridimensionales.
Material	Es un conjunto de propiedades físicas y ópticas que se le puede dar a un objeto.
Textura	Representación digital de la superficie de un objeto
Shader	Es un código escrito que da al programador interacción con el GPU

	modificando el renderizado de un pixel o vertex.
Render	Proceso de generar una imagen a partir de una serie de cálculos y representación gráfica de mallas, texturas, iluminación, etc.
Highlight	Destacar. En el contexto, resaltar a través de un efecto de brillo y emisión de color alrededor de un objeto.
Lightmap	Mapa que representa iluminación y sombreado de una escena.
Skybox	Representación de un entorno o cielo a través de un cuboide infinito

ÍNDICE

Contenido

ILUSTRACIONES

TABLAS

1 INTRODUCCIÓN

En este capítulo se presenta una introducción al trabajo de título, detallando los objetivos, motivaciones y enfoque empleados en el desarrollo. Además se exponen las características funcionales con las cuales debe cumplir la aplicación a desarrollar.

1.1 Objetivo y características funcionales de la aplicación TR3

El objetivo de este trabajo es crear TR3, una aplicación para dispositivos móviles que permita centralizar información, datos en tiempo real y documentación relevante de los equipos de la Planta San Isidro de Endesa, en una maqueta virtual de la misma, para la consulta y toma de decisiones de personal en terreno.

Las características funcionales que debe cumplir el desarrollo de la aplicación son: Centralizar información a consultantes mediante dispositivo móvil, entregando orientación técnica o de operación de los equipos presentes dentro de la planta San Isidro de Endesa.

- Tener una representación tridimensional de la planta industrial, navegable e interactiva.
- Tener la capacidad de autorreferencia o geolocalización a través de lectura de códigos únicos instalados en los equipos físicos.
- Tener la capacidad de conectarse y desplegar datos desde los sistemas de información y bases de datos del cliente.
- Tener un buscador que permita al usuario consultar respecto a un elemento a través de su nombre o código.
- La aplicación y maqueta virtual de la planta deben generar un entorno amigable para el usuario.

1.2 Motivación

Actualmente para obtener información sobre el funcionamiento de un equipo físico en la planta San Isidro de Endesa y dar respuesta a posibles fallas en terreno, se debe recurrir a distintas fuentes de información, lo que obliga a una búsqueda amplia y diversa de fuentes ya sean manuales, planos, guías de maniobra, etc.

Si bien las plantas industriales por lo general tienen algún sistemas de monitoreo y diagnóstico del funcionamiento de sus equipos, esta información está generalmente en una oficina central, y por lo tanto de difícil acceso para los operadores en terreno.

Por lo anterior, existe la necesidad de condensar, centralizar y disponibilizar esta información para el operador en terreno, disminuyendo los tiempos de búsqueda y su capacidad de respuesta, facilitando las operaciones dentro de la compañía e impactando positivamente en la optimización de procesos y recursos.

1.3 Enfoque

Parte fundamental del trabajo es entender la estructura de información que maneja y requiere cada cliente.

Es de vital importancia entonces el trabajo en conjunto para lograr convenir y abrir espacios de comunicación tanto en aspectos técnicos, como interpersonales para lograr una solución integral y adaptada a sus necesidades.

Por lo anterior se hace una breve descripción de los sistemas de información que maneja el cliente, *Plant Information System* (*PI system*): primero porque forma parte fundamental del diseño de la arquitectura de la aplicación en cuanto a comunicación se refiere y además porque el poder entender y generar una conexión a sus sistemas de información es lo que finalmente permite adjudicar el proyecto.

2 DESCRIPCION PLATAFORMA DE PROGRAMACION UNITY 3D

En este capítulo se presenta la plataforma de entorno de desarrollo Unity3D, se describen en detalle sus características y virtudes y luego se presentan los elementos fundamentales y otros términos que serán recurrentemente utilizados en el desarrollo. Finalmente se presentan los complementos o *Plugins* que se utilizaran en el proyecto y se explica cómo integrarlos a la plataforma de desarrollo. Estos son extensiones y librerías que permiten dar funcionalidades adicionales al proyecto.

Unity3D es una plataforma de entorno de desarrollo enfocada en la creación de escenarios *3D* y orientada al desarrollo de videojuegos multiplataforma.

Es por esto último que tiene gran compatibilidad con exportación a distintos dispositivos y plataformas. (iOS, Android, Windows Phone, PS4, Xbox, PC, etc).

Acorde a la propia descripción del Software:

> *Unity es una plataforma de desarrollo flexible y poderosa para crear juegos y experiencias interactivas 3D y 2D multiplataforma. Es un ecosistema completo para todo aquel que busque desarrollar un negocio a partir de la creación de contenido de alta gama y conectarse con sus jugadores y clientes más fieles y entusiastas. [1]*

Así, se entiende que las posibilidades otorgadas por el software son de alto alcance en programación como también en la entrega de herramientas para la producción de contenidos amables para el usuario.

A continuación se hace una breve introducción a los principios fundamentales de este *IDE*.

2.1 Elementos Fundamentales de Unity 3D

2.1.1 GameObject

Es el objeto básico dentro de la escena y jerarquía. Este objeto solo posee las características de posición, rotación y escala.

A este se le pueden agregar scripts que se transformarán en los atributos del objeto. Puede haber atributos de forma, *meshes* (malla poligonal), atributos de textura, scripts de movimiento, etc.

2.1.2 Layers

Layers o capas son una manera de clasificación de objetos en Unity3D. Dicha clasificación permite trabajar con uno o múltiples tipos de capas a la vez. Las layers son comúnmente utilizadas por cámaras para renderizar sólo una parte de la escena, por luces para iluminar sólo ciertos objetos y por cualquier script que quiera discriminar por capas.

Así, la clasificación por layers permite trabajar con grupos acotados de objetos, disminuyendo considerablemente los recursos. Este ahorro de recursos es especialmente notorio en el área de renderizado e iluminación de la escena.

2.1.3 Prefabs

Es un objeto o jerarquía de objetos que tienen scripts (o atributos) vinculados y especificados de manera previa, lo que permite la consulta expedita de dichos objetos, sin necesidad de diseñar constantemente los atributos. Por ejemplo, la necesidad de definir dimensiones o volúmenes de un polígono, que posteriormente será utilizado de manera repetida.

2.2 Interfaz de Usuario de Unity 3D

Untiy3d tiene 5 ventanas principales, las cuales vienen en un *Layout* básico predefinido y se pueden adaptar dependiendo el gusto y necesidades del programador.

Estas 5 ventanas son las siguientes:

- Game (1): en donde se ve la previsualización de la aplicación funcionando.
- Vista Escena (2): el escenario donde se muestran los objetos.
- Jerarquía (3): en el que se despliega la lista de elementos en la escena.
- Inspector (4): en donde se despliegan opciones y scripts asociados a un elemento seleccionado.
- Proyecto (5): se despliegan todas las carpetas con todos los recursos utilizados por el proyecto. (Imágenes, modelos 3D, scripts, *plugins*, etc.)

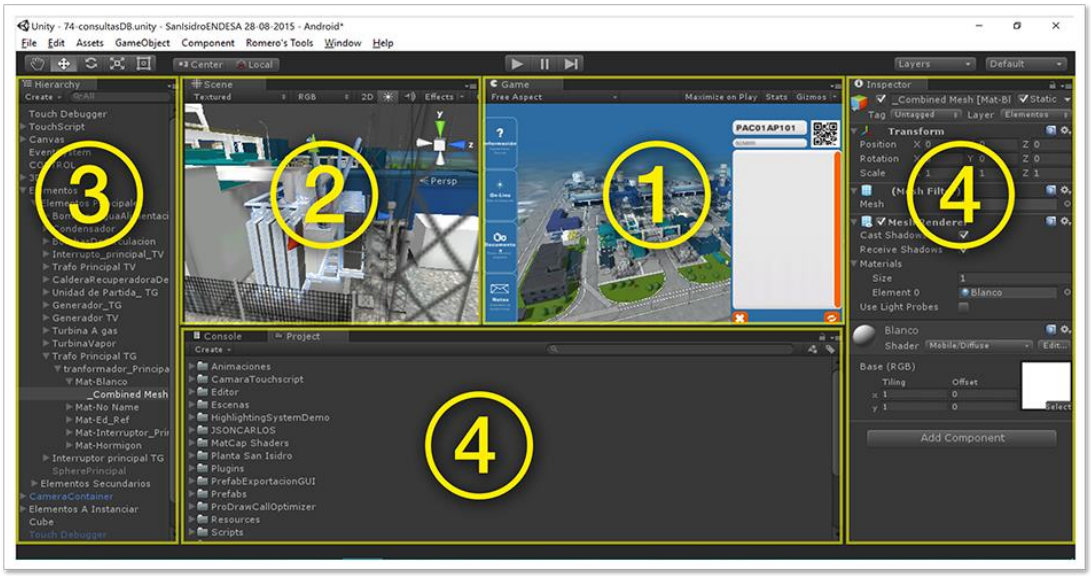

Ilustración 2.1 Interfaz de usuario.

2.2.1 Game

Es una ventana que permite previsualizar la aplicación que se desarrolla. Esto implica la capacidad de probar la aplicación y la interacción que incurre en ella, evaluando en qué nivel de avance se encuentra la aplicación y sus funciones, como también detectar errores en la fase operacional y generar la mejora pertinente.

2.2.2 Vista de Escena

Una escena es sólo un nivel dentro de un juego, un juego puede tener múltiples niveles. La escena es el espacio 3D donde van colocados todos los elementos visuales y de interfaz. Esta vista permite tomar, mover y rotar elementos en el escenario de manera visual y rápida.

De igual manera, tanto modelos 3D como la interfaz de la aplicación pueden ser modificados fácilmente desde esta vista. Un elemento importante en la escena es la cámara. La cámara es el objeto principal de una escena y lo que ve esta cámara es lo que se despliega en pantalla.

2.2.3 Jerarquía

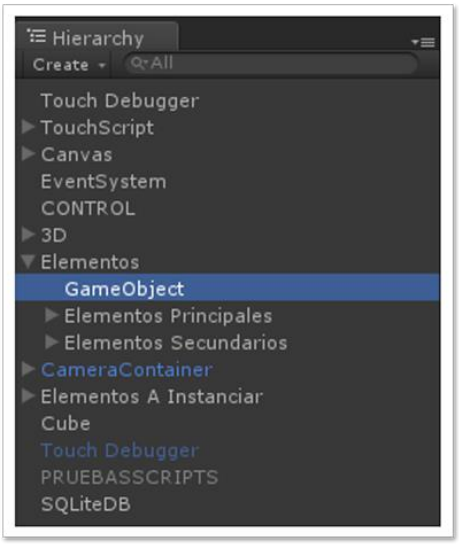

Ilustración 2.2 Jerarquía.

La jerarquía está directamente relacionada con la escena actual, es aquí donde se despliega una lista de todos los elementos presentes en la escena.

Es importante mencionar que una aplicación puede estar compuesta por varias escenas, pero tanto la vista de escena, como la jerarquía solo muestran los objetos relacionados a una sola escena a la vez.

Todo objeto en la jerarquía es un *gameobject*, es la estructura básica y cuenta con características de posición, rotación y tamaño de escala. Por defecto, todo objeto que se agrega a la escena tiene estas características que se heredan de dicho objeto principal.

Los *gameobjects* u objetos en la jerarquía, pueden agruparse. Estando agrupados, el de más arriba es llamado objeto padre y los de dentro, hijos.

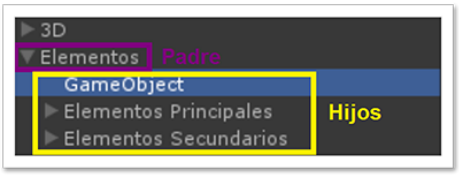

Ilustración 2.3 Jerarquía padres e hijos.

La posición y rotación de los objetos sueltos en la jerarquía hacen referencia al Origen (0,0,0) tanto en posición, rotación y escala.

En cambio en el caso de los hijos la posición rotación y escala son relativas al padre. En términos prácticos, esto significa que al modificar el padre, el hijo será modificado para quedar inmóvil respecto al padre.

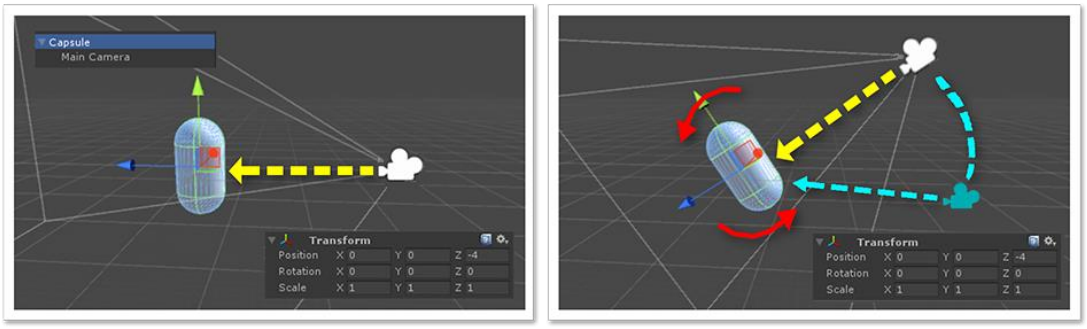

Ilustración 2.4 Movimiento objeto hijo respecto a rotación del padre.

2.2.4 Inspector

El inspector se puede entender como el conjunto de atributos y scripts que tiene un objeto.

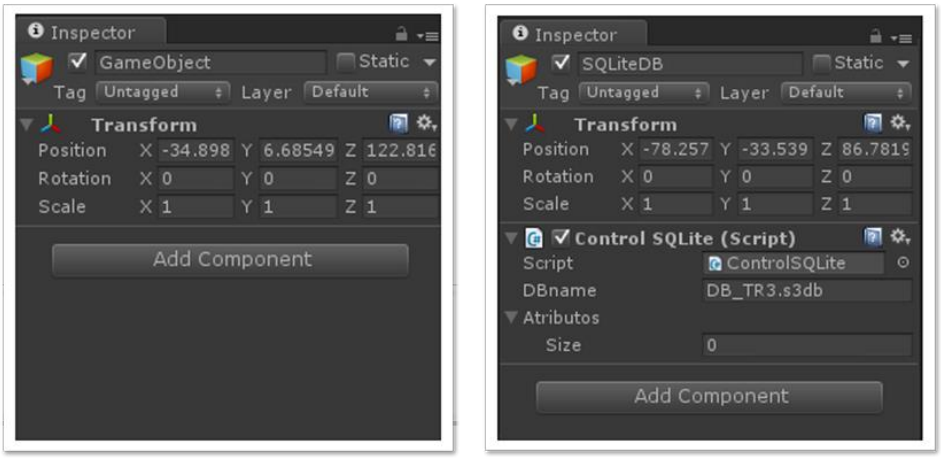

Ilustración 2.5 Inspector.

El objeto por defecto o *GameObject* básico, tiene como atributos el *Transfrom* que en el fondo es posición, rotación y escala. A este elemento básico es posible agregarle todos los scripts que se requiera. Así, los scripts, se definen por la capacidad de dar funcionalidad y atributos a un objeto.

Estos scripts pueden afectar a cualquier objeto dentro de la escena y no únicamente al *GameObject* en el que se encuentra, pero es aconsejable, por temas de orden, que los scripts estén relacionados con el *GameObject* en el que se encuentran.

2.2.5 Proyecto

La ventana de proyecto es donde se encuentran todos los elementos que se utilizaran en la construcción del proyecto, para todas y cada una de las escenas. En esta ventana existe una carpeta llamada *Plugins*, donde se agregan funcionalidades externas o librerías externas a las que trae Unity3D. Además, en esta ventana se ubican todos los

componentes (íconos, bases de datos, modelos 3Ds, scripts) que aseguran los recursos que serán utilizados en el programa.

Esto implica que todo lo que se encuentra en *Proyecto* es susceptible de ser utilizado: se mantiene un universo de información, del cual, la aplicación utiliza sólo los archivos que necesite.

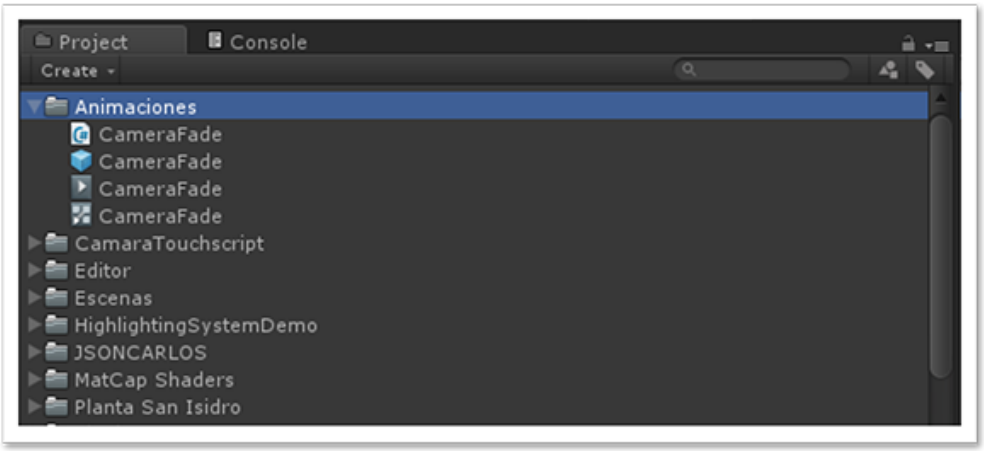

Ilustración 2.6 Proyecto.

2.3 Complementos para Unity 3D

A continuación se describen brevemente las librerías externas que serán requeridas en el desarrollo de la aplicación. Detalles sobre su instalación y funcionamiento están disponibles en el **anexo A**.

2.3.1 Librería TouchScript

Una de las necesidades funcionales de la aplicación es permitir la navegación dentro de un entorno y modelo en 3D. La interacción con la navegación del usuario será a través de gestos táctiles sobre la pantalla, para esto es necesario reconocer estos gestos.

Unity3D, la presente plataforma de desarrollo, permite reconocer **eventos** táctiles, pero no así el reconocimiento de **gestos** táctiles, técnicamente la interpretación de estos eventos reconocidos.

Por lo anterior se decide utilizar una librería externa *TouchScript* compatible con la plataforma de desarrollo, la cual trae implementado el reconocimiento de una variedad de gestos táctiles como gestos de escalado y rotación - de ahora en adelante *Gestures* - y facilita la creación e interpretación de eventos táctiles nuevos.

2.3.1.1 Principales características de la librería TouchScript

Este *framework* fue desarrollado por *Interactive labs* y sus principales características son:

- Compatibilidad con PC, Mac, Android e IOS.
- Compatible con la versión gratuita de Unity3D.
- Permite reconocimiento de gestos simultáneos.
- Trae varios gestos predeterminados y permite la creación de gestos nuevos.
- Trae un emulador de segundo toque para utilizar en el entorno de desarrollo.
- Es *Open source* y licenciado por el MIT.
- Tiene buena documentación [2].

2.3.2 Librería SQLite

La librería *SQLite* es un motor de bases de datos *SQL* autónomo que funciona de manera local, dejando fuera la necesidad de tener una máquina externa corriendo un servicio de base de datos. Es fácil de implementar y totalmente gratuita.

A nivel mundial, cuando se habla de bases de datos para aplicaciones o videojuegos, *SQLite* sin duda alguna es la herramienta de base de datos local más utilizada.

2.3.2.1 Principales características de la librería SQLite

A continuación son detalladas las características principales de este *plugin*, vinculadas a ciertos requerimientos de la aplicación [3]:

- Las transacciones presentan atomicidad, consistencia, aislamiento y durabilidad (ACID por sus siglas en inglés), incluso después de los fallos del sistema y fallas de energía.

- No necesita configuración administrativa.
- Tiene implementación completa de *SQL,* con funciones avanzadas como índices parciales y tablas comunes.
- *API* fácil de utilizar e intuitiva.
- Código fuente abierto y bien comentado.
- No contiene dependencias externas.
- Multi-plataforma. Soporta Android, IOS, Linux, MAC, Solaris, Windows,etc.
- La base de datos completa se guarda en el disco local.

2.3.3 Librería ZXing

ZXing.NET [4] es una librería que permite decodificar y codificar códigos de barras en imágenes. Soporta códigos *QR, EAN, UPC* y matrices de datos entre otros.

2.3.3.1 Principales características de la librería ZXing

La librería viene pre compilada para distintas plataformas entre ellas *.NET 2.0, 3.5* y *4.0, Silverlight 4* y *5*, *Windows Phone*, etc. Además viene pre compilado para Unity3D, esto es equivalente a *.NET* sin utilizar *System.drawing*, lo cual permite compilación multiplataforma.

Zxing es también una librería de código abierto, gratuita y muy fácil de integrar.

3 DESARROLLO DE LA APLICACIÓN TR3

En este capítulo se presenta la arquitectura generada y los requisitos funcionales de programación y la implementación de la aplicación TR3, considerando pruebas de rendimiento y optimización de la aplicación.

Se fundamentan las decisiones tomadas en las distintas etapas, entrando en detalle sobre cada una de las funcionalidades presentes en la aplicación: Creación de una maqueta virtual, sistema de navegación y movimiento en el espacio 3D, reconocimiento de gestos táctiles, descarga y manejo de información, etc.

En la etapa de optimización se describe el proceso de renderizado y como este afecta el desempeño de la aplicación. Además, se explican una serie de técnicas para optimizar este proceso y aumentar así considerablemente el desempeño de aplicaciones 3D.

En relación a los propósitos específicos con que debe cumplir TR3, las características principales que debe poseer son:

- Permitir recorrer la planta de manera virtual.
- Seleccionar y enfocar elementos de la planta industrial.
- Desplegar datos e indicadores en tiempo real de elementos.
- Disponibilizar documentación y planos relacionados con elementos.
- Desplegar alertas de elementos.
- Permitir buscar elementos por identificador o por lectura de código QR.

En cuanto a lo señalado, se hace necesario evidenciar la situación de desarrollo y soporte de la aplicación, tanto a nivel de conceptos como de los requerimientos funcionales para su implementación.

3.1 Arquitectura de la aplicación

A partir de las características principales de la aplicación se genera una arquitectura base definida por la ilustración 3.1.

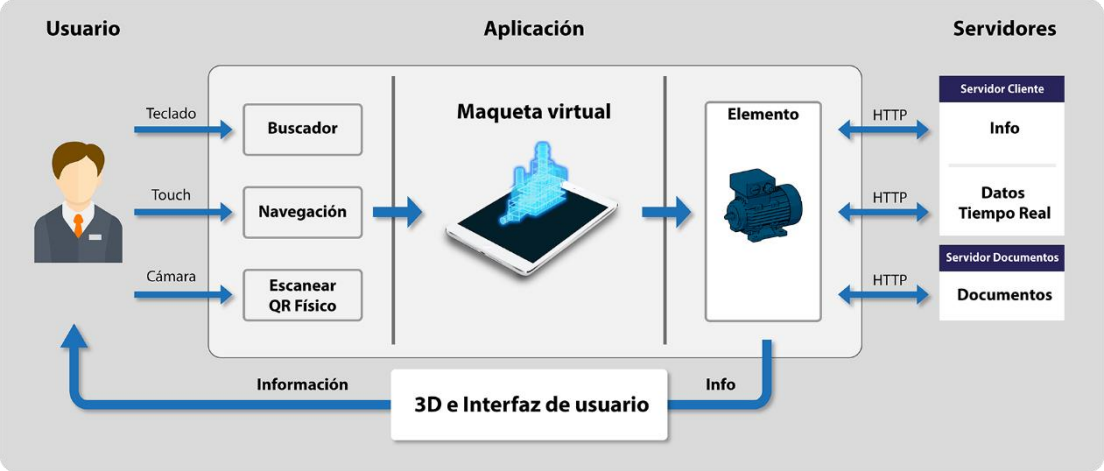

Ilustración 3.1 Diagrama de la aplicación.

La relevancia de la aplicación se encuentra en los elementos y en su vinculación con datos en tiempo real y documentación, para la consulta e interacción de los usuarios. A continuación se define y profundiza en las características del elemento, siendo éste el componente base de la programación.

3.1.1 Definición de elemento

Elemento, para fines de este programa, se define como la representación virtual de equipos dentro de una planta productiva. Estos pueden ser bombas, motores, válvulas, transformadores, ventiladores, compresores, etc.

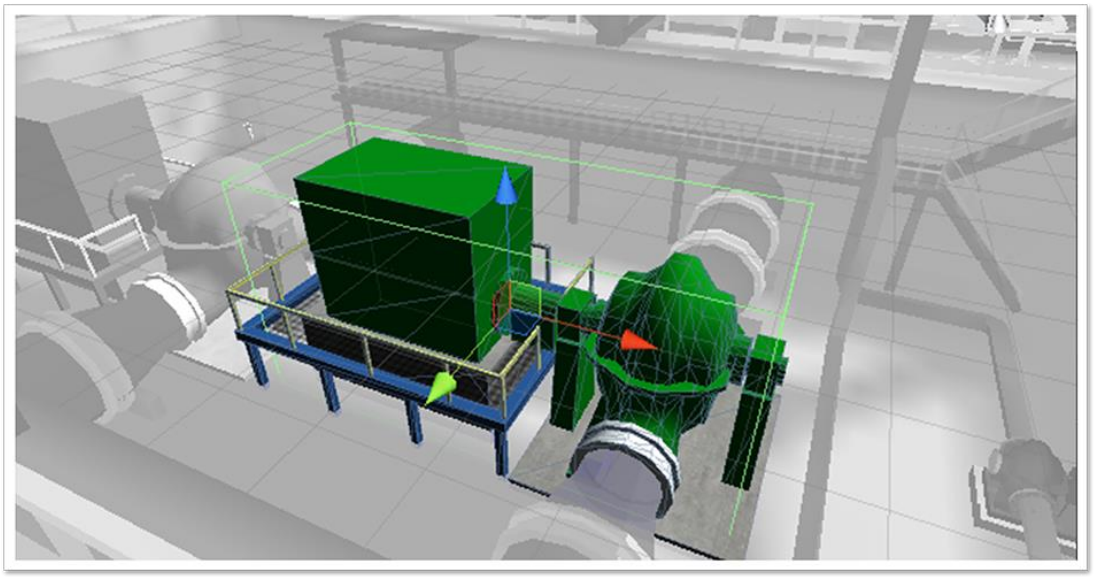

Ilustración 3.2 Elemento de ejemplo.

Estos elementos o equipos son la unidad fundamental para el desarrollo de la aplicación y los principales componentes interactivos dentro de ésta.

En cuanto a la programación, un elemento es un *GameObject* u objeto básico de Unity3D al cual se le tienen que agregar ciertos atributos para considerarlo Elemento.

Para considerar el elemento como tal, se debe en primero lugar agregar a este un *box collider*, que permitirá detectar interacción con el elemento.

Un *box collider* es un componente físico de Unity3D en forma de cubo que permite generar un evento al detectar contacto o colisión con este.

Además se le agrega el script *InfoObjeto* que tiene los datos básicos del objeto, el código *KKS* y el nombre del objeto. De lo anterior, es importante explicar que el *KKS* constituye el principal método de búsqueda y relación de los objetos, mucho más que el nombre. El código es único, el nombre no necesariamente.

Finalmente, se le agrega el script *HighlightObjeto* que permite destacar el objeto con un contorno de color al ser seleccionado. Lo señalado se puede ver en las dos secciones destacadas a continuación.

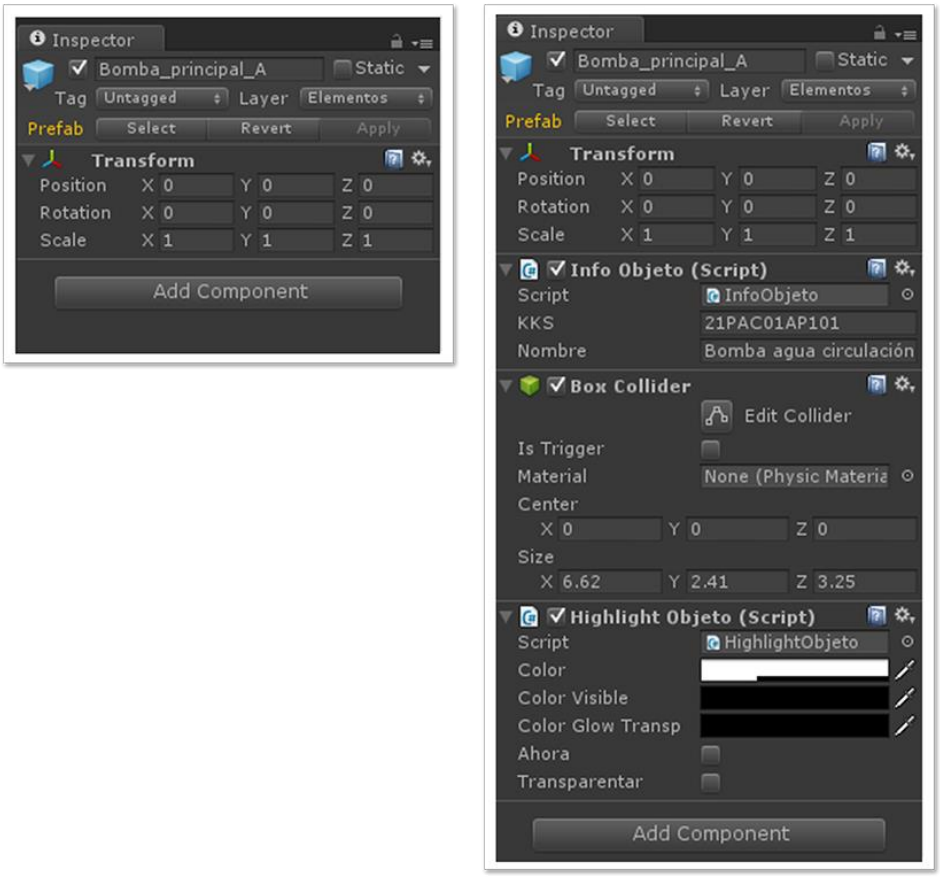

Ilustración 3.3 *GameObject* básico vs Elemento interactivo.

3.2 Requisitos funcionales de programación

Para lograr cumplir con las características de la aplicación TR3, es necesario considerar los siguientes requisitos funcionales de programación:

- Creación y montado de la maqueta virtual de la planta.
- Sistema de movimiento y navegación dentro de la planta.
- Interacción con elementos dentro de la maqueta virtual.
- Descarga, manejo y despliegue de información.

- Sistema de búsqueda por teclado y lectura de códigos QR.

A continuación son desarrollados cada uno de los puntos, junto a las definiciones conceptuales pertinentes.

3.3 Creación y montado de la maqueta virtual de la planta

El primer paso y más importante es tener una maqueta virtual de la planta productora que se quiere recorrer en busca de información de elementos.

3.3.1 Modelado de la planta

Para realizar la reconstrucción de planta es necesario hacer un recorrido real tomando referencias fotográficas y de video, esto además puede ser apoyado por fotografías aéreas y planos de construcción para temas de escala.

Luego, el equipo de diseño, comienza a modelar la planta y cada uno de sus elementos en algún programa de modelado en 3D. El programa debe tener como opción de exportación el formato *.FBX*, esto debido a que Unity3D, la plataforma de desarrollo, tiene *.FBX* como formato estándar para materiales y modelos 3D.

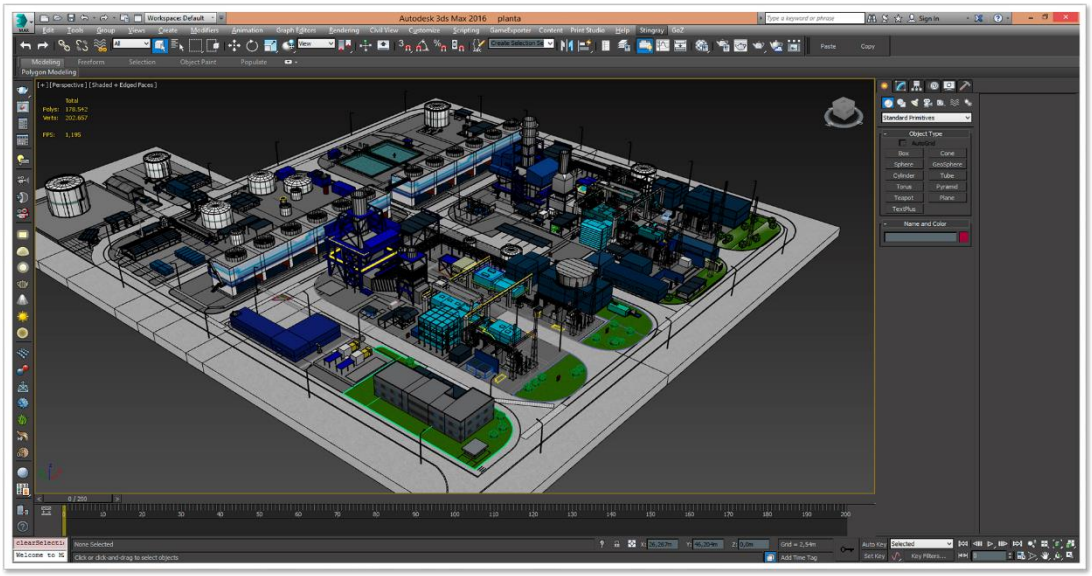

Ilustración 3.4 Modelo de la planta 3D en 3DS Max.

Dentro de los programas orientados a generación de modelos 3D, se destaca *3DS Max* [5] y *Maya* [6]. El proceso de modelado, dependiendo el nivel de detalle, puede llevar de 3 a 6 meses de desarrollo.

3.3.2 Importación del modelo 3D

Una vez que la planta está modelada, lista en un programa externo y exportada en formato *.FBX*, es bastante simple el proceso de importación. Solo se debe arrastrar el archivo *.FBX* de la maqueta virtual a Unity3D. Una vez Importado todos los parámetros se ajustan por defecto, pero hay dos parámetros que es importante revisar.

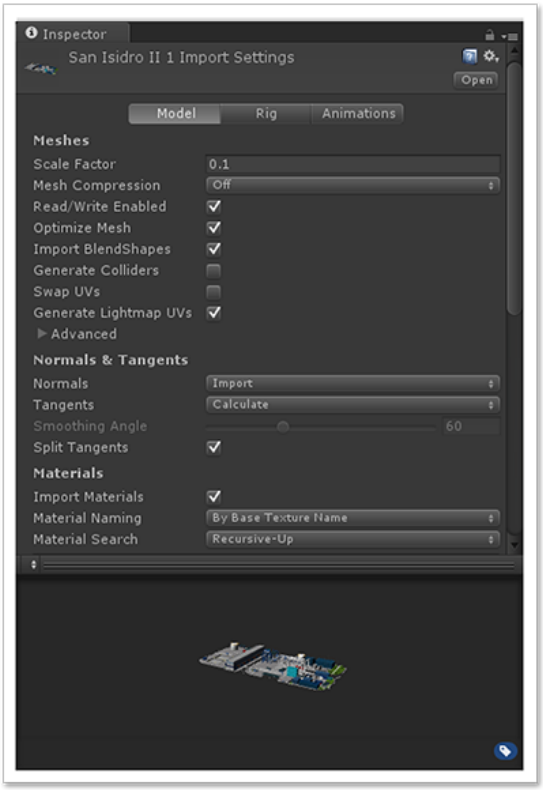

Ilustración 3.5 Propiedades modelo 3D (.FBX).

Primero que esté *Generate Lightmaps UVs* activado, esto permite generar *Lightmaps* o mapa de iluminación del modelo reduciendo en gran medida los recursos utilizados por la aplicación.

Segundo es el factor de escala, si bien Unity3D puede trabajar con cualquier tamaño de escala es muy recomendado escalar los elementos al tamaño unitario de Unity3D para que todos los cálculos posteriores, de movimiento, iluminación, etc, sean calculados en la unidad fundamental.

Una manera fácil de ajustar el factor de escalamiento es poniendo en escena un *Cube* que es un elemento primitivo básico de Unity3D y que tiene dimensiones 1x1x1. Luego comparar visualmente el modelo 3D importado con *Cube* y ajustar el factor de escala para que tenga un tamaño proporcionalmente correcto.

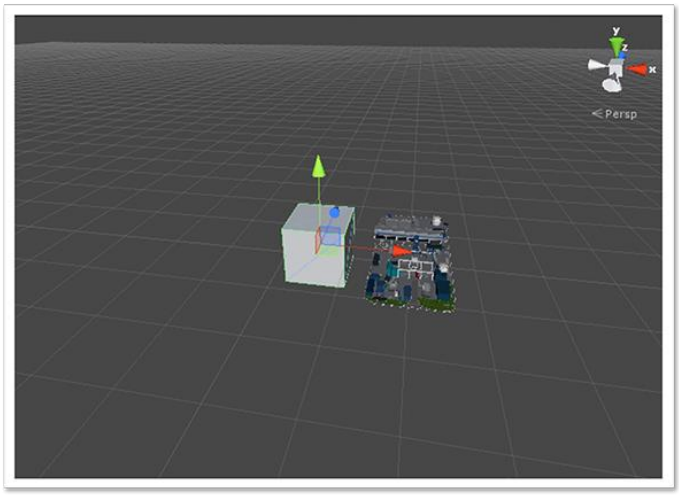

Ilustración 3.6 Comparación inicial de modelo 3D con cubo unitario.

Ilustración 3.7 Modelo 3D ajustado para que puerta mida 2 cubos (2mts) de alto.

3.3.3 Ambiente e iluminación

3.3.3.1 Ambiente

Para dar mayor realismo al modelo 3D de la planta es aconsejable agregar elementos ambientales a ésta.

La manera más fácil de realizar esto es agregando un *Skybox.*

3.3.3.1.1 ***Skybox***

Un *skybox* es un método para crear *backgrounds*. Cuando un *skybox* se aplica a la escena, esta es encerrada en un cuboide infinito. Montañas, edificios, cielo y otros objetos inalcanzables son proyectados en las caras de este cubo dando la sensación de estar inmerso en un entorno o mundo tridimensional.

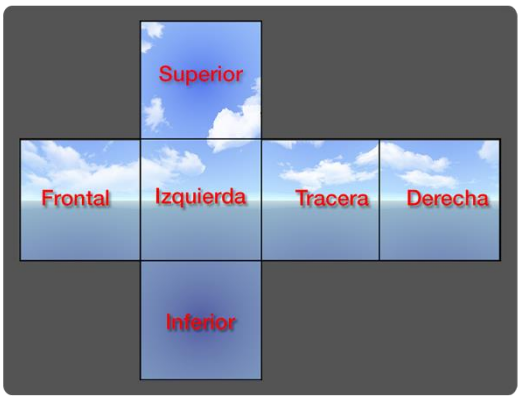

Ilustración 3.8 Ejemplo de *Skybox*.

3.3.3.1.2 *Agregar Skybox a la escena*

Unity3D trae una colección de *skyboxes* que se pueden utilizar. Para importar estos *skyboxes* hay que ir a *Assets->Import Package->Skyboxes*. Esto agregará una carpeta al proyecto con varios *skyboxes* para seleccionar, luego para agregarlos a la escena es preciso ir a *Edit->Render Settings* y en la pantalla vincular el *Skybox* que se desea aplicar.

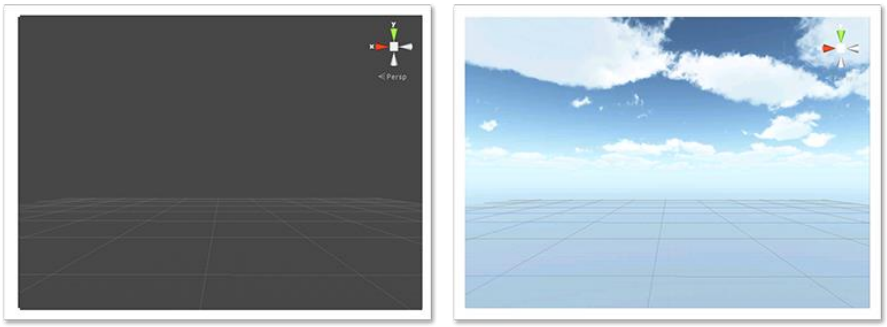

Ilustración 3.9 Escena con y sin *Skybox*.

3.3.3.2 Iluminación.

Una escena por defecto no tiene fuentes de luz, esto significa que todo objeto agregado a la escena se verá sumamente oscuro, por lo que se hace necesario agregar algún tipo de iluminación.

3.3.3.2.1 *Iluminación dinámica*

Las fuentes de luz en Unity3D por defecto son dinámicas, esto quiere decir que se va calculando la iluminación de un lugar en tiempo real, lo que permite mover la fuente de luz y ver como esto repercute en cada uno de los objetos alcanzados por esta luz.

Si bien la iluminación dinámica es muy cómoda al momento de generar un escenario iluminado y proyectar sombras, tiene un muy alto gasto de recursos, ya que por cada cuadro está calculando toda la iluminación en la escena, independiente si la fuente de luz se mueve o es estática.

3.3.3.2.2 *Generacion de Lightmaps*

Un *lightmap* es una estructura o textura que almacena la información de iluminación de un objeto. Esto permite pre-calcular la iluminación y almacenarla, de esta manera se reduce una gran cantidad de recursos en comparación a tener iluminación en tiempo real.

Para generar *Lightmaps* lo primero es agregar una luz dinámica en escena y adaptarla según conveniencia, color, intensidad, dirección, etc.

Como la información de iluminación queda guardada de manera fija, es solo aplicable a elementos fijos que no se moverán en la aplicación. Para esto se seleccionan los objetos deseados, y en el *Inspector* se marcan como estáticos.

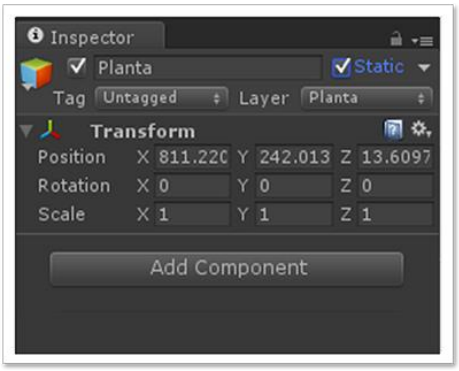

Ilustración 3.10 Marcar objeto como estático.

Una vez marcados todos los objetos deseados como estáticos y teniendo una luz correctamente dirigida a estos objetos, se procede al quemado de las luces.

Para esto, es preciso ir a *Window->Lightmapping* donde se puede elegir en la primera pestaña que objetos serán utilizados para generar el *Lightmap*. Lo más sencillo es poner "*All*", lo que seleccionará todos los objetos estáticos que tengan *"Generate Lightmaps UVs"* activado como objetivo del *Lightmapping*.

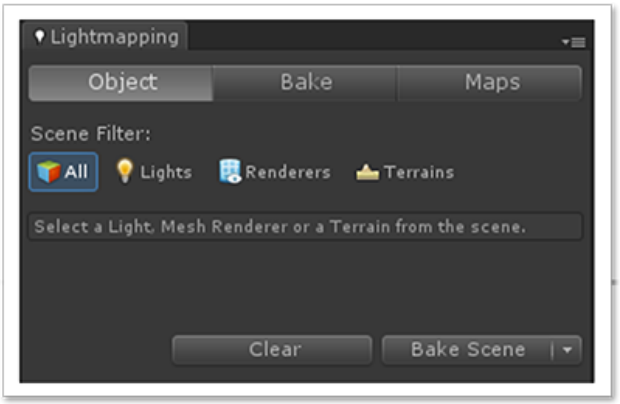

Ilustración 3.11 Selección de objetos para *Lightmapping*.

Luego en la opción *"Bake"* se despliega una lista de parámetros para calcular las sombras e iluminación, como la calidad, el número de rebotes, el color del cielo, etcétera.

Una vez definido estos parámetros se da la opción *"Bake Scene"* con lo que se inicia el proceso de *"Bake"* o quemado de luces. Dependiendo los parámetros y la cantidad de elementos puede tardar horas. Finalizado el proceso de quemado es importante desactivar la fuente de luz, ya que la iluminación de los objetos es parte de la textura del mismo y ya no hay que calcularla.

3.3.4 Comentarios

Hasta ahora, ha sido presentada la maqueta virtual en la aplicación, permitiendo la navegación en la escena tridimensional.

Es importante saber cómo ambientar de correcta manera un escenario tridimensional, por un tema estético pero también muy ligado a los recursos utilizados. La idea es siempre mejorar el ambiente utilizando la menor cantidad de recursos posibles. Es aquí donde los *Skyboxes* y el *Lightmapping* juegan un papel fundamental.

3.4 Sistema de navegación

Una vez modelada y puesta en escena la maqueta virtual, es necesario dar al usuario la capacidad de navegar y moverse a través de ella. Para ello se requiere crear controles para manipular y manejar la cámara a través de la maqueta virtual.

3.4.1 Movimiento de cámara

Como se señala en el **capítulo 2**, la cámara es el dispositivo a través del cual el usuario ve este mundo virtual. Es entonces esta última la que debe navegar la maqueta virtual.

Para la navegación es necesario permitir al usuario las capacidades de desplazarse, acercarse y alejarse de los objetos (*zoom*), rotar la cámara respecto a un foco (orbitar), y finalmente centrarse en un elemento.

3.4.1.1 Funcionamiento básico de movimiento

Para los movimientos de navegación *zoom* y rotación orbital se requiere un objeto de referencia. Es debido a esto que se opta por dejar a dicho objeto, de ahora en adelante llamado *pivote*, como controlador de los movimientos y foco fijo de cámara.

Para lograr los movimientos de una manera sencilla, es utilizada la jerarquía de hijos dejando al pivote como padre de la cámara. De esta manera, donde quiera que se mueva el objeto pivote, la cámara lo seguirá como punto focal.

Así mismo, el script de control de movimiento estará adjunto (*attached*) al pivote y no directamente a la cámara.

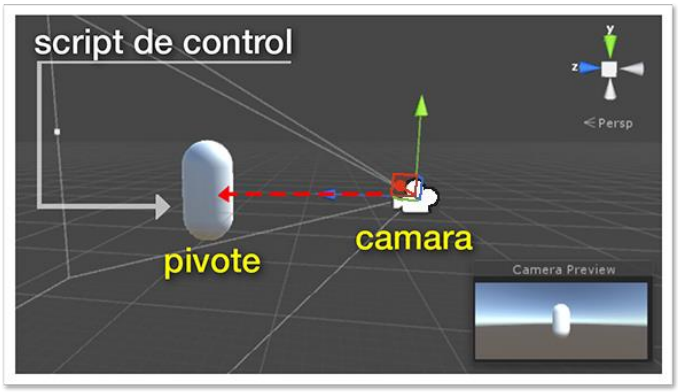

Ilustración 3.12 Esquema básico de movimiento de cámara.

3.4.1.2 Zoom

Ya que el objeto pivote es el foco de la cámara, el *zoom* se lograra con solo alejar o acercar la cámara al pivote.

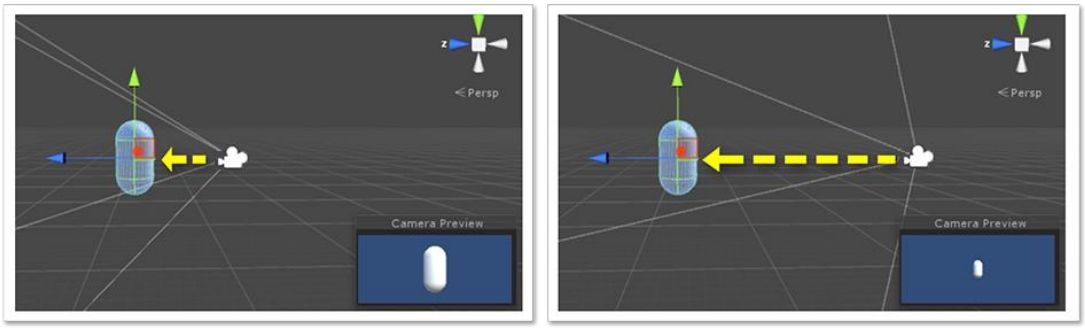

Ilustración 3.13 *Zoom-In Zoom-Out.*

Se debe tomar en cuenta que la velocidad de movimiento de la cámara dependerá de la distancia al pivote, esto por un tema de percepción y proporción.

Si el usuario se aleja del objeto pivote una distancia pequeña, estando cerca de este, el cambio será notorio. Si en cambio el usuario se aleja la misma distancia del pivote estando a gran distancia de éste, parecerá que no hubo ningún movimiento. Es un tema de proporciones.

Para mantener la percepción de zoom constante, se utiliza la función matemática *Mathf.Clamp* que acota un valor entre un máximo y un mínimo. En este caso se utiliza para aumentar o disminuir la velocidad de movimiento, dependiendo de la distancia al pivote.

```
void Zoom(float _wheel){
    float distanciaPivote = Mathf.Abs(camara.transform.localPosition.z);
    float currentZoomSpeed = Mathf.Clamp(maxZoomSpeed * (distanciaPivote / maxZoom), minZoomSpeed, maxZoomSpeed);

    camara.transform.Translate(Vector3.forward * (_wheel * currentZoomSpeed));
}
```

Ilustración 3.14 Función zoom, velocidad respecto a distancia al pivote.

3.4.1.3 Rotación Orbital

Como el objeto pivote está en el foco de la cámara, la función de orbitar alrededor de un punto, no es nada más que rotar alrededor del pivote, esto es particularmente sencillo cuando la cámara es hija del pivote, ya que como se explica en la jerarquía de Unity3D, al mover al objeto padre, el hijo se mantendrá estático respecto a este.

Por lo tanto para lograr este movimiento orbital tan solo se requiere una función de rotación para el pivote.

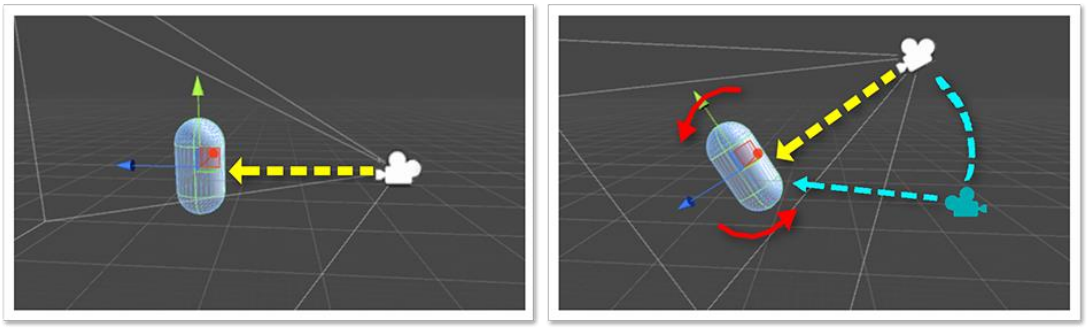

Ilustración 3.15 Movimiento orbital.

3.4.1.4 Foco en un elemento

Para hacer foco en un elemento lo primero que se debe hacer es tener elementos interactivos (**Apartado 3.1.1**). Desde el punto de vista del movimiento esto significa desplazar el pivote en dirección del objeto seleccionado. Al igual que en el caso del *zoom*, se recomienda variar la velocidad de acercamiento dependiendo de la distancia a la que uno se encuentre del elemento, nuevamente por un tema de proporcionalidad.

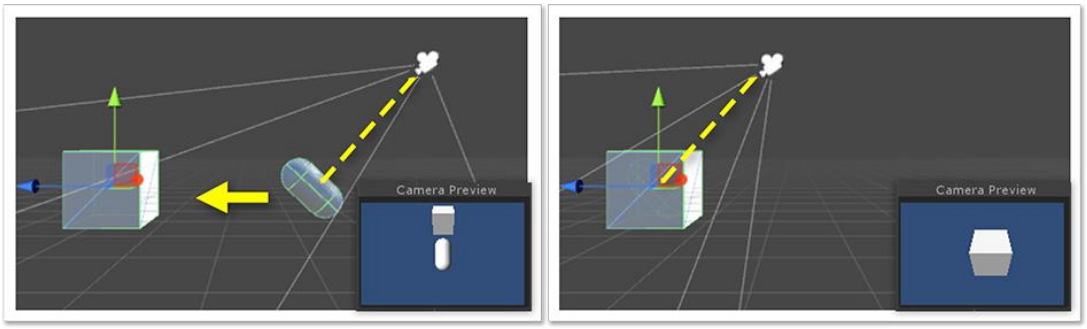

Ilustración 3.16 Función foco.

Aquí otra vez se usa la función *Mathf.Clamp* para limitar la velocidad de movimiento respecto a la distancia del objeto seleccionado. Por otro lado, para generar el desplazamiento en dirección al objeto seleccionado se utiliza la función *Vector3.MoveToward* que viene hecha para resolver esta necesidad.

```
public Transform target;
public float speed;
void Update() {
        float step = speed * Time.deltaTime;
        transform.position = Vector3.MoveTowards(transform.position, target.position, step);
}
```

Ilustración 3.17 Uso básico función *MoveToward*.

3.4.1.5 Desplazamiento

El desplazamiento de la cámara se logra generando movimiento bidimensional de la cámara respecto al objeto pivote. Lo cual traspasado al sistema local de coordenadas del pivote se interpreta como los vectores locales *Up-Down* y *Left-Right*.

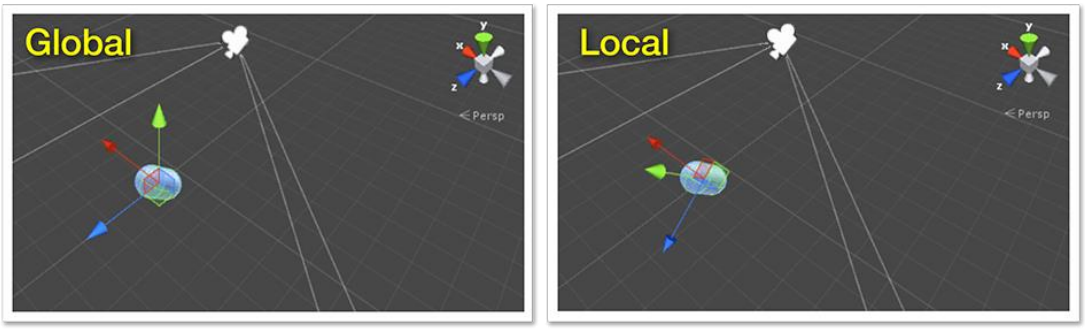

Ilustración 3.18 Eje de referencia Global vs Local.

Para esto se utiliza la función de traslado *Transform.Translate* y los vectores locales *Vector3.Up* y *Vector3.Right*.

```
void Pan(float _x,float _y){
    float distanciaPivote = Mathf.Abs( transform.localPosition.z);
    float currentPanSpeed = Mathf.Clamp(maxPanSpeed * (distanciaPivote / maxZoom), minPanSpeed, maxPanSpeed);

    Vector3 translateX = Vector3.right * (_x * currentPanSpeed) * -1;
    Vector3 translateY = Vector3.up * (_y * currentPanSpeed) * -1;

    pivote.transform.Translate(-translateX);
    pivote.transform.Translate(translateY);
}
```

Ilustración 3.19 Script básico de paneo.

Gracias a esto, se obtienen todas las funcionalidades necesarias para la cámara, que permitirá recorrer la maqueta virtual de manera óptima. A continuación se hace necesario generar los inputs adecuados para dar al usuario la capacidad de navegación.

3.4.2 Gestos táctiles

Ya que la aplicación está pensada para dispositivos móviles, es de especial interés el poder reconocer gestos táctiles como *input* para el control de movimiento de cámara.

Si bien Unity3D trae reconocimiento de eventos táctiles no trae incorporada una librería de gestos, esto es la interpretación de eventos táctiles.

Para este fin se incorpora la librería *TouchScript* (**apartado 2.3.1**), que trae varios gestos táctiles (*Gestures)* reconocibles e incluso la opción de generar nuevos.

3.4.2.1 Implementación y uso de TouchScript

A continuación se describe el proceso para incorporar la detección de gestos *touch* en la aplicación.

3.4.2.1.1 *TouchManager*

Lo primero es agregar a la escena el *prefab TouchScript* que viene dentro de la carpeta con el mismo nombre.

Este *prefab* es un *GameObject* que trae adjunto el script principal del *plugin TouchManager*, el cual, como fue señalado anteriormente, es el encargado de detectar y procesar los *input touch*.

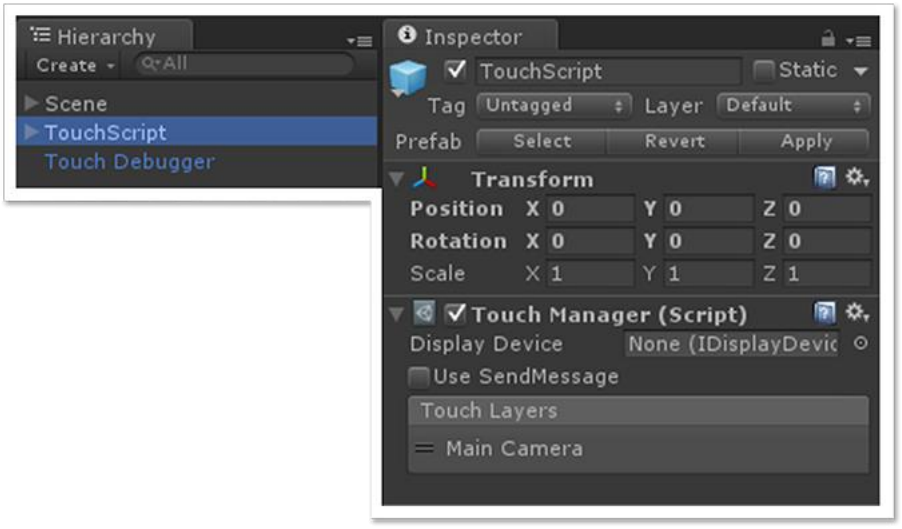

Ilustración 3.20 Prefab *TouchScript* con *TouchManager*.

3.4.2.1.2 *Detección de gestos en pantalla completa*

Como se explicó en la sección complementos de Unity3D *TouchScript* es necesario agregar los objetos interesados en recibir *Gestures*. En el presente caso, se desea controlar la cámara y por ende, detectar los *Gestures* en la totalidad de la pantalla. Con el fin de realizar dicha tarea, se selecciona un *GameObject* vacío para agregarle un script que viene en el *plugin*, llamado *FullScreenLayer*, luego se ancla a la cámara con la que trabajará.

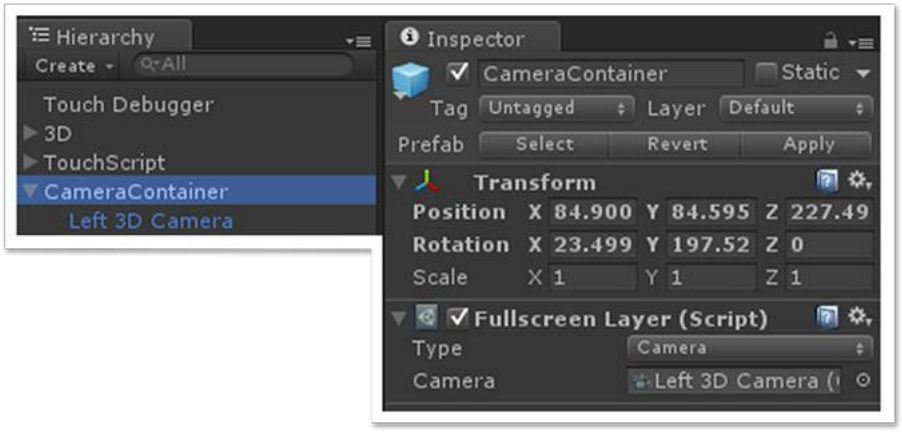

Ilustración 3.21 Agregar reconocimiento de gestos en pantalla completa.

FullscreenLayer permite cambiar la interacción sobre objetos específicos a interacción a pantalla completa.

3.4.2.1.3 *Gestos a reconocer*

Como se presenta en la sección anterior, se necesita asignar *Gestures* para las acciones de *Zoom*, Desplazamiento básico, Rotación orbital y Foco en un elemento.

Esto se logra adjuntando (*attaching*) scripts de *Gestures* específicos para cada uno de los movimientos a realizar. Estos script tienen que adjuntarse al objeto que contiene *FullscreenLayer*.

Para esto se escogen los siguientes *Gestures*: *zoom*, desplazamiento básico, rotación orbital, foco en elemento.

3.4.2.1.3.1 Zoom

Para el zoom se utiliza el script *ScaleGesture*.

El movimiento consiste en interactuar con dos dedos en pantalla, acercando o alejando los dedos entre sí. Este gesto se utilizará para acercar o alejar la cámara dentro de la aplicación.

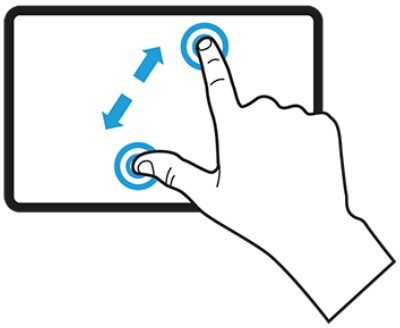

Ilustración 3.22 *ScaleGesture.*

3.4.2.1.3.2 Desplazamiento básico

Para el desplazamiento se utiliza *PanGesture*.

Al tocar y desplazar el dedo sobre la pantalla, se realiza el gesto de *Pan*. Este gesto será utilizado para lograr el desplazamiento básico dentro de la navegación de la maqueta virtual.

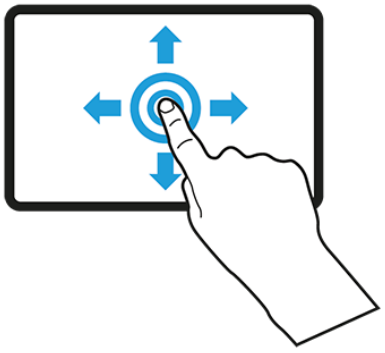

Ilustración 3.23 *PanGesture.*

3.4.2.1.3.1 Rotación orbital

Para la rotación orbital también se utiliza *PanGesture*. Con la diferencia que posteriormente, en el script de control de cámara, además de ver si es un *PanGesture* valido, solo actuará si esta se realiza con dos dedos.

Para realizar el gesto de rotación, tiene que estar dos dedos presentes sobre la pantalla y desplazarlos juntos de forma vertical u horizontal. Este gesto permitirá rotar orbitalmente al rededor del foco de la cámara.

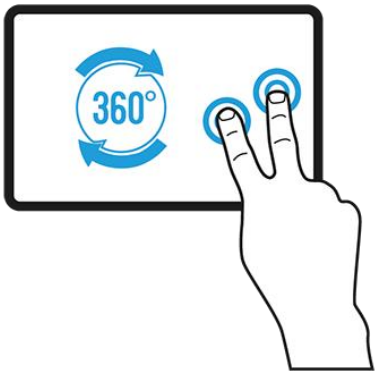

Ilustración 3.24 *PanGesture* con 2 toques.

3.4.2.1.3.1 Foco en elemento

Para enfocar un elemento se utiliza *LongPressGesture*

Al tocar la pantalla y dejar el dedo fijo en ella por 1 segundo, se logra detectar el gesto de *LongPress*. Este gesto permitirá que el pivote de la cámara se mueva hacia el elemento sobre el cual se hizo el gesto.

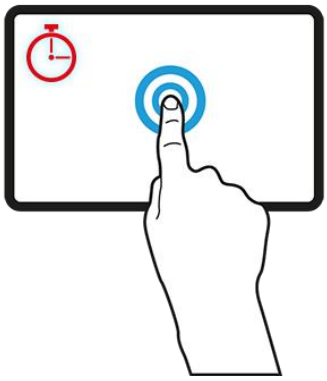

Ilustración 3.25 *LongPress.*

Además el elemento presionado en este caso tiene que ser un elemento interactivo, de lo contrario se descartará el gesto.

Una vez adjuntos estos scripts al objeto, es necesario modificar el script de movimiento de cámara para recibir estos eventos como inputs.

3.4.3 Vinculación gestos touch y movimiento de cámara

Para generar interacción entre la detección de *gestures* y los movimientos de cámara, es necesario primero agregar al script de movimiento, *MovimientoCamara*, una suscripción a los eventos de *Gestures* y a partir de ellos generar los inputs de movimiento.

3.4.3.1 Suscripción a eventos de Gesture

El script de movimiento de cámara, *MovimientoCamara*, tiene que suscribirse o escuchar los eventos que despachan los *Gestures* previamente agregados.

Es una buena práctica dar de baja los eventos cuando no se les necesita más, es por esto que se agregan dos funciones una para suscribirse a los eventos y otra para de-suscribirse.

```
void EnableGestures(){
    GetComponent<PanGesture>().Panned += panStateChangeHandler;


    GetComponent<ScaleGesture>().Scaled += scaleStateChangeHandler;
    GetComponent<LongPressGesture>().StateChanged += longPressStateChangedHandler;
}

void DisableGestures(){
    GetComponent<PanGesture>().Panned -= panStateChangeHandler;


    GetComponent<ScaleGesture>().Scaled -= scaleStateChangeHandler;
    GetComponent<LongPressGesture>().StateChanged -= longPressStateChangedHandler;

}
```

Ilustración 3.26 Funciones para agregar y remover eventos.

Como se ve en la ilustración 3.26:"Funciones para agregar y remover eventos", será agregado un *listener* para cada tipo de *gesture* presente y que se desee detectar. Además, existe la opción de habilitar y deshabilitar la escucha de *Gestures* con las funciones *EnableGestures()* y *DisableGestures()*.

3.4.3.2 Disparo y detección de eventos

Una vez detectado alguno de los 3 eventos que se estan escuchando se dispara *PanStateChangeHandler(), scaleStateChangeHandler()* o *longPressStateChangedHandler()* , que son los encargados de pasar la información relevante como input para el movimiento de cámara.

3.4.3.2.1 *PanStateChangeHandler()*

Esta función es la encargada de recibir y manejar el *PanGesture*. Es necesario recordar que se está utilizando *PanGesture*, tanto para el desplazamiento como para la rotación orbital. Esto dependiendo de si el evento se realiza con uno o dos dedos.

Con este fin, se añade una línea de código que permita reconocer la cantidad de dedos que se están utilizando, para ver si se utiliza una u otra función.

```
private void panStateChangeHandler(object sender, EventArgs e){
    PanGesture _MyGesture=sender as PanGesture;
    bool _Son2Dedos;
    switch(_MyGesture.ActiveTouches.Count){
    case 1:
        _Son2Dedos=false;
        break;
    case 2:
        _Son2Dedos=true;
        break;
    default:
        break;
    }
```

Ilustración 3.27 Función para detectar si el gesto está hecho con 1 o 2 toques.

Además, es necesario saber en qué estado está el gesto. Si el gesto está cambiando o en estado de cambio, se debe hacer movimiento, de lo contrario no, por lo que interesa el estado *_MyGesture.State.*

Si esto es así, se debe comparar la posición normalizada actual del gesto con la posición normalizada anterior del gesto, y este delta pasarlo como una coordenada bidimensional al script de movimiento de rotación o desplazamiento dependiendo si el gesto está hecho con uno o dos dedos.

```
if (_MyGesture.State == PanGesture.GestureState.Changed) {
    Vector2 _PanSpeed=Vector2.zero;
    Vector2 _RotateSpeed = Vector2.zero;
    //PAN
    if (!_Son2Dedos){
        _PanSpeed=_MyGesture.NormalizedScreenPosition-_MyGesture.PreviousNormalizedScreenPosition;
        _PanSpeed=_PanSpeed*10;
        if(!UnityEngine.EventSystems.EventSystem.current.IsPointerOverGameObject(-1))
            Pan(_PanSpeed.x,_PanSpeed.y);

        //ROTATION
    }else{
        _RotateSpeed=_MyGesture.NormalizedScreenPosition-_MyGesture.PreviousNormalizedScreenPosition;
        _RotateSpeed=_RotateSpeed*10;
        if(!UnityEngine.EventSystems.EventSystem.current.IsPointerOverGameObject(-1))
            OrbitArround(_RotateSpeed.x,_RotateSpeed.y);
    }
}
```

Ilustración 3.28 Detección y envío de coordenadas para desplazamiento.

3.4.3.2.2 *ScaleStateChangeHandler()*

Esta función es la encargada de recibir y manejar el *ScaleGesture*.

De igual modo al caso anterior se necesita saber en qué estado está el gesto, en caso de que el gesto este en ejecución se debe pasar la información a la función de *zoom*.

Nuevamente se observa si *_MyGesture.State* está en estado de cambio. Si esto es así se captura el valor *LocalDeltaScale* que entrega el gesto. Es importante tomar en cuenta que *LocalDeltaScale* tiene valor central uno, es por esto que se le aplica un desfase de -1 para centrarlo en 0 y pasarlo a la función encargada del *zoom*.

```
private void scaleStateChangeHandler(object sender, EventArgs e){
    ScaleGesture _MyGesture=sender as ScaleGesture;
    float _DeltaScale=0;

    if(_MyGesture.State==ScaleGesture.GestureState.Changed)
    {
        _DeltaScale=_MyGesture.LocalDeltaScale;
        if(!UnityEngine.EventSystems.EventSystem.current.IsPointerOverGameObject(-1))
            Zoom(_DeltaScale-1);
    }
}
```

Ilustración 3.29 Detección y envío de delta para Zoom.

3.4.3.2.3 *LongPressStateChangeHandler()*

Esta función es la encargada de recibir y manejar el *LongPressGesture*.

En este caso el *LongPressGesture* es un *Gesture* discreto por lo que solo se dispara como reconocido y desaparece.

```
private void longPressStateChangedHandler(object sender, GestureStateChangeEventArgs e)
{

    if (e.State == Gesture.GestureState.Recognized)
    {
        if(!UnityEngine.EventSystems.EventSystem.current.IsPointerOverGameObject(-1))
            TargetingRaycast(true);
    }
}
```

Ilustración 3.30 Reconocimiento de *LongPress.*

Sin embargo este gesto será utilizado para hacer foco en un elemento, por lo que luego de ser reconocido, habrá que revisar si efectivamente este gesto está sobre un elemento seleccionable o no.

Esto se verá en detalle en la siguiente sección, Interacción con elementos dentro de la maqueta virtual.

3.5 Interacción con elementos dentro de la maqueta virtual

A continuación se explican las tres etapas que permiten la interacción del usuario con los elementos dentro de la maqueta virtual.

3.5.1 Detección y selección de un elemento

Con el fin de generar interacción entre el usuario y los elementos dentro de la maqueta virtual, se utiliza la función foco. (**Apartado 3.4.1.4**)

Sin Embargo, como fue señalado con anterioridad los gestos táctiles son hechos sobre *FullScreenLayer*, una especie de vidrio sobre el cual se detectan *gestures*. Este “vidrio”

si bien permite hacer gestos sobre toda la pantalla, es una barrera que impide llegar a los elementos que están detrás de ella.

Por lo anterior se hace necesario utilizar otro mecanismo para llegar a los elementos. Para ello se utiliza *Physics.Raycast*: una función de Unity3D que permite lanzar un rayo de colisión y devolver información sobre la colisión. *Physics.Raycast* se lanzará en la dirección del vector que se genera entre el punto de origen de la cámara y la posición tocada en pantalla, dando la sensación de tocar el objeto.

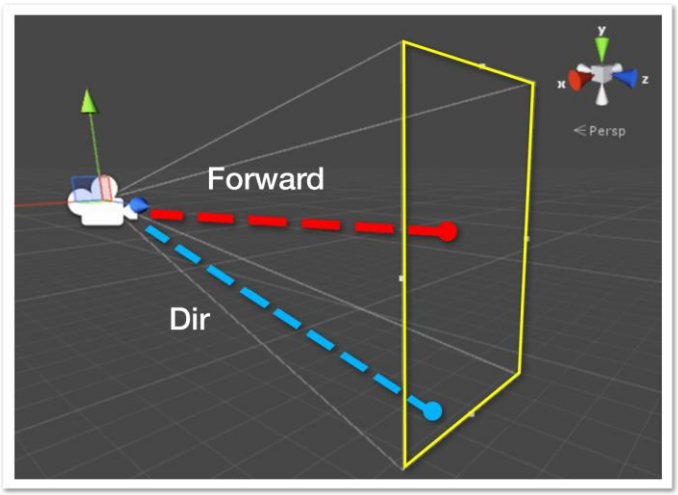

Ilustración 3.31 Vector de dirección de *Raycast*.

Physics.raycast requiere que los objetos tengan una malla de colisión para interactuar, sin embargo no tiene la facultad de discriminar entre uno u otro con esta característica, por lo que es recomendable realizar la discriminación por *Layer*.

Es por lo anterior que los elementos ya cuentan con una malla de colisión, *Box Collider*, y están asignados a la *Layer* "*Elementos*".

Finalmente *raycast* devuelve el elemento con que colisionó. Esto permitirá posteriormente desplegar información respecto al elemento, enfocarlo, resaltarlo y cualquier otra acción que se quiera realizar con el elemento.

3.5.2 Enfoque y menú del elemento

Una vez que *raycast* devuelve el elemento, entrega las coordenadas al script de movimiento de cámara para que pueda enfocarlo.

Además, es necesario desplegar las opciones de acción que estarán disponibles respecto al elemento. Esto se logra desplegando un menú de opciones genérico, que es alimentado por el *KKS* y nombre del elemento seleccionado, almacenados en *InfoObjeto*, script adjunto (*attached*) al elemento.

El pasar la referencia de *KKS* al menú del elemento facilita enormemente las relaciones de búsqueda posteriores.

Ilustración 3.32 Elemento seleccionado y menú de elemento.

3.5.3 Enfatizar elemento

Además de enfocar el elemento y desplegar un menú interactivo con información relevante de este, que se refiere a la parte funcional de la aplicación, es necesario

agregar un par de toques estéticos, que son importantes para visualmente destacar la selección de un elemento.

La primera mejora visual es destacar el elemento seleccionado, esto se logra con el script *HighlightObjeto*, el cual activa un *shader*, que permite generar un *glow* o brillo de color alrededor del objeto.

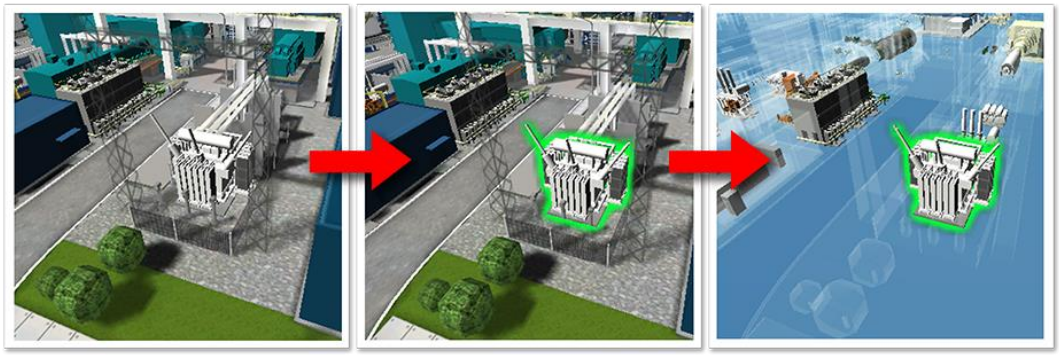

Ilustración 3.33 Mejora visual de selección, con *Highlight* y transparencia de entorno.

La segunda mejora visual es hacer transparentes todos los objetos que no están marcados como elemento. Esta transparencia parcial se logra cambiando el material a todos los objetos de la escena por un material de alta transparencia con bordes blanqueados.

Para ello se hace un barrido de todos los objetos de la escena, llenando dos listas con los siguientes criterios: una lista con todos los objetos marcados como elementos, y una segunda lista con todos los otros objetos que tengan además un material asociado.

Al activar el efecto de transparencia, simplemente se aplica el material semitransparente a dichos objetos, guardando su material original para poder deshacer los cambios posteriormente.

Como la lista de elementos no se ve afectada por este cambio, además de destacar el elemento seleccionado, en pantalla sólo se verán objetos interactivos facilitando enormemente la navegación.

3.6 Descarga, manejo y despliegue de información

La utilidad básica de la aplicación está en desplegar información característica y en tiempo real de los equipos de la central termoeléctrica, para ser visualizada en terreno. Para esto se necesita llevar la información que el cliente maneja a la aplicación.

Es importante entender que si bien en principio es responsabilidad del cliente disponibilizar estos datos, se hace necesario trabajar en conjunto para lograr enlazar el sistema del cliente con la aplicación.

Para ello es preciso entender el funcionamiento de sus sistemas de datos.

A continuación se presenta brevemente las estructuras principales del sistema de información del cliente.

3.6.1 PI System

PI System [7] es una suite de productos de software utilizados para la recolección, análisis, entrega y visualización de datos. Se comercializa como una infraestructura empresarial para la gestión de datos y eventos en tiempo real.

3.6.1.1 Estructura básica

La estructura básica de *PI system* consta de tres partes esenciales:

Data Source: fuentes de información o inputs, que pueden ser mediciones en tiempo real de datos variables de equipos como temperatura por ejemplo, datos fijos o incluso otras bases de datos ajenas a *PI*.

Pi Server: que a su vez se divide en dos partes, *PI Data Archive*, estructura principal que permite almacenar y disponibilizar todos los datos y mediciones provenientes de cualquier *Data Source*. Y *PI Asset Framework* (*PI AF*) que permite al administrador generar estructuras relacionales y árboles de información respecto a todos o parte de los datos almacenados en *PI Server*.

Pi Clients: Distintos softwares diseñados para analizar, desplegar y gestionar eventos provenientes de *PI System*. Entre ellos *PI Datalink* que permite analizar datos

en planillas como Excel y *PI Coresight* una herramienta web que permite acceso rápido y seguro a los datos de *PI system*, permitiendo generar, visualizar y compartir información de manera gráfica.

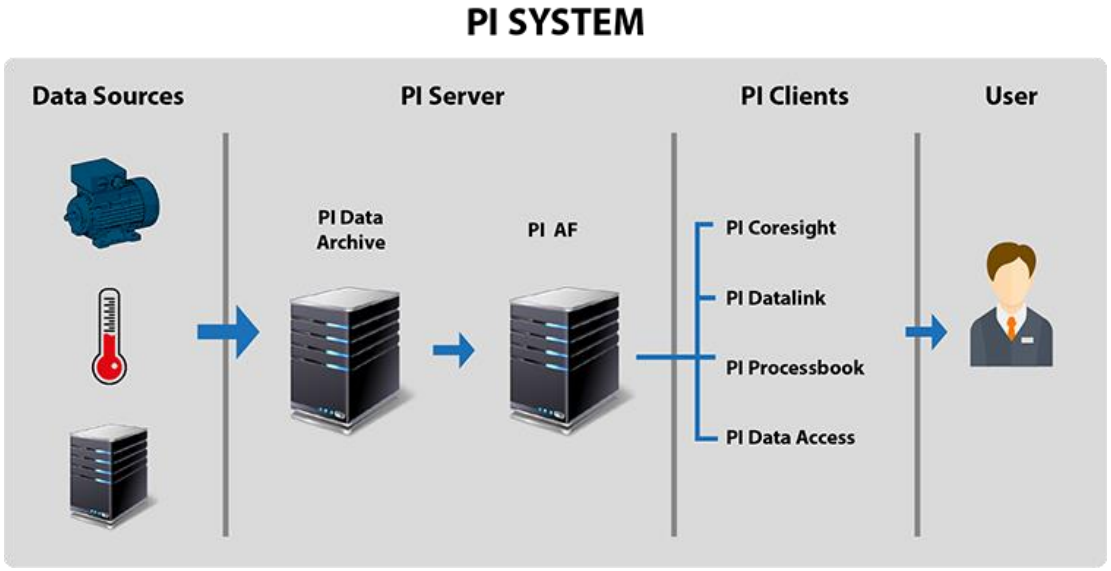

Ilustración 3.34 Estructura básica Pi System.

Lo bueno de esta estructura es que permite a los usuarios de gama baja, que son la gran mayoría de usuarios, desentenderse del sistema y simplemente a nivel de usuario ver datos con alguno de los clientes *PI*.

3.6.1.2 PI AF

Como se vio, una de las características principales de *PI System* es ser capaz de tomar cualquier fuente de información, de cualquier lugar y centralizarla en *PI Server*. Muchas fuentes de información y bases de datos pueden ser centralizadas y monitoreadas desde *PI Server*.

PI Data Archive centraliza y almacena la información tal cual la envían las *Data Sources*. En otras palabras es como una bóveda donde se almacenan todos los datos recibidos de *PI Data Sources* pero sin un orden aparente. Por lo que son difíciles de buscar y manipular.

PI AF es la parte que se encarga de ordenar y organizar los datos a nivel de usuarios. *PI AF* permite generar jerarquías o árboles de información estableciendo una nomenclatura de datos y unidades de medida.

Para distintos procesos dentro de la industria uno puede crear árboles, jerarquías y nomenclatura de datos invocando datos directamente desde *PI Data Archive* o desde otra estructura guardada de *PI AF.*

Finalmente los *PI Clients* se conectan a *PI AF* en la que la información ya está ordenada en vez de tener que descifrar datos desde *PI Data Archive*.

3.6.1.3 PI Interfaces

Las interfaces de *PI* son las que hacen a este Software tan importante.

Si bien la capacidad de *PI System* de recopilar información en tiempo real de distintas fuentes y centralizarlas pareciera ser el fuerte del software, son las interfaces de *PI* lo que realmente hacen a este software la solución industrial preferida.

Como se ve en la ilustración 3.35:"Estructura básica de interfaces PI", las interfaces *PI* son las encargadas de recolectar los datos en tiempo real de fuentes externas y llevarlas al sistema *PI*.

Existen más de 400 *PI Interfaces* para todo tipo de equipos y maquinarias, creadas para la gran mayoría de los procesos productivos de plantas industriales. Entre ellas:

Honeywell 620 Series PLCs, Siemens S7-200 PLCs y *Mitsubishi electric MELSEC* (*NTI)* [8].

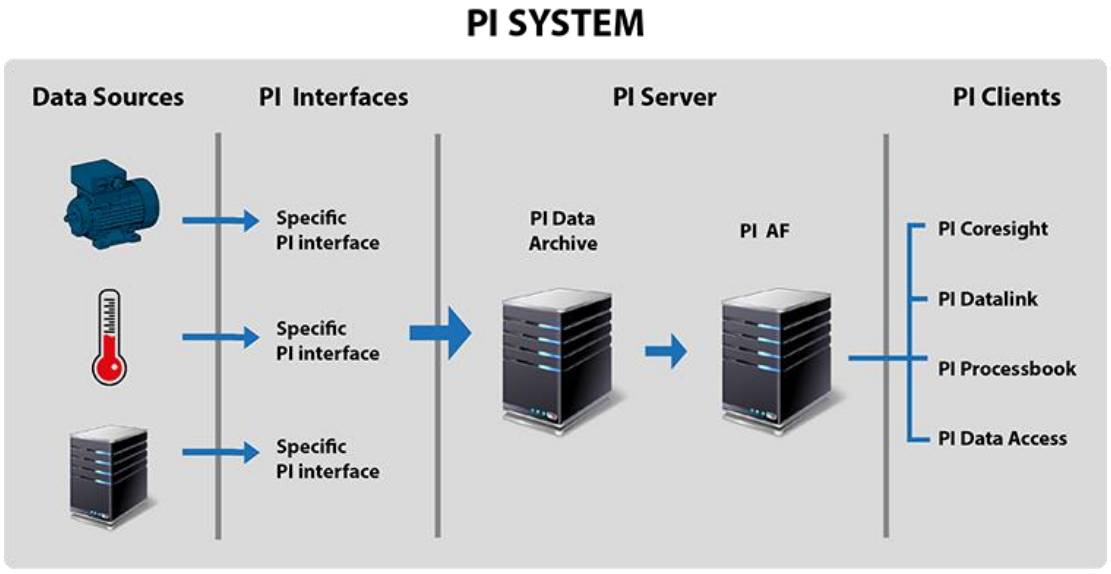

Ilustración 3.35 Estructura básica de interfaces PI.

Tener interfaces para la mayoría de las grandes marcas y maquinaria industrial es lo que hace realmente poderosa a esta suite de productos.

3.6.2 Orden y discriminación de información

Con el fin de disponer de la información necesaria para la aplicación, se trabajó en conjunto con el equipo del *CMD* (Centro de Monitoreo y Diagnostico) de ENDESA. El trabajo con este equipo responde a generar los permisos mínimos y necesarios para la conexión de la aplicación a sus sistemas de información.

Es importante precisar que los datos a utilizar por la aplicación son un grupo reducido comparado con la totalidad de datos disponibles. Los datos a utilizar corresponden específicamente a los elementos que están incluidos en la maqueta virtual. Esto significa una cantidad reducida y determinada de equipos en una sola de sus centrales termoeléctricas.

Como fue señalado anteriormente, la interfaz que permite hacer este filtro u ordenar la información a gusto es *PI AF*, parte de *PI System*, que permite generar jerarquías o árboles de información estableciendo una nomenclatura de datos a voluntad del usuario.

En este caso se desea buscar características técnicas, documentación y datos en tiempo real de equipamiento en la central termoeléctrica.

El primer paso es crear una jerarquía que permita tener un nombre único para cada elemento, este nombre único es el código *KKS* que cada equipo posee, así la estructura básica de jerarquía es la siguiente:

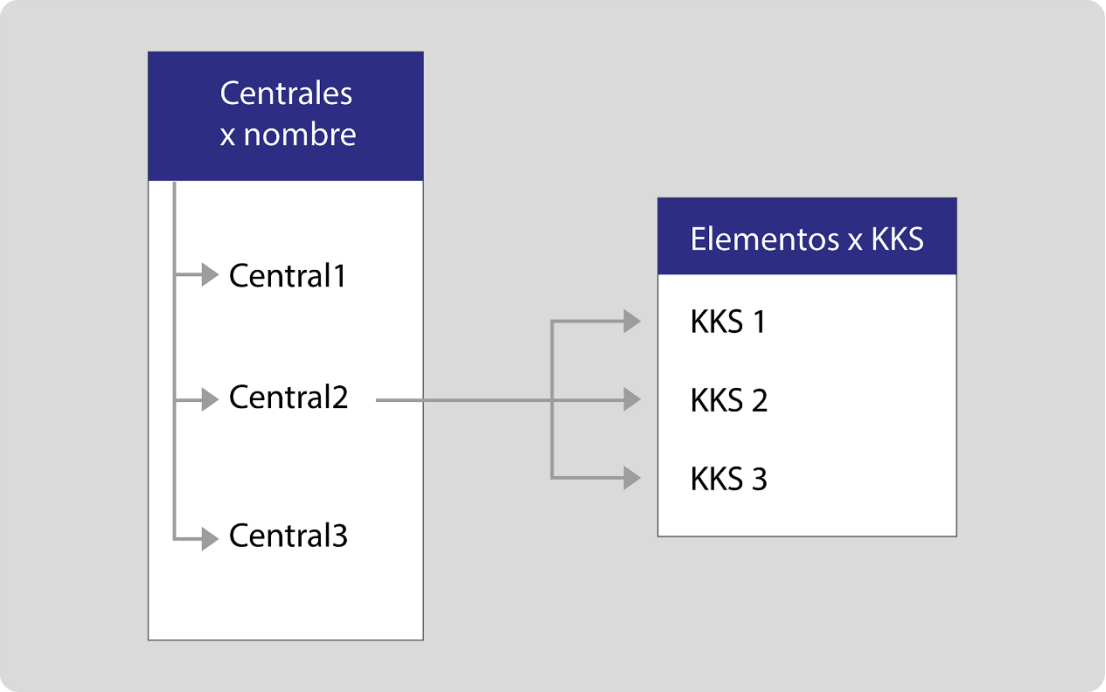

Ilustración 3.36 Esquema básico de jerarquía y búsqueda.

Con el identificador único *KKS* es fácil entonces encontrar un elemento dentro de la jerarquía. El paso siguiente es tener distintos tipos de atributo para características técnicas, datos en tiempo real y documentos.

Así la estructura de un elemento queda de la siguiente manera:

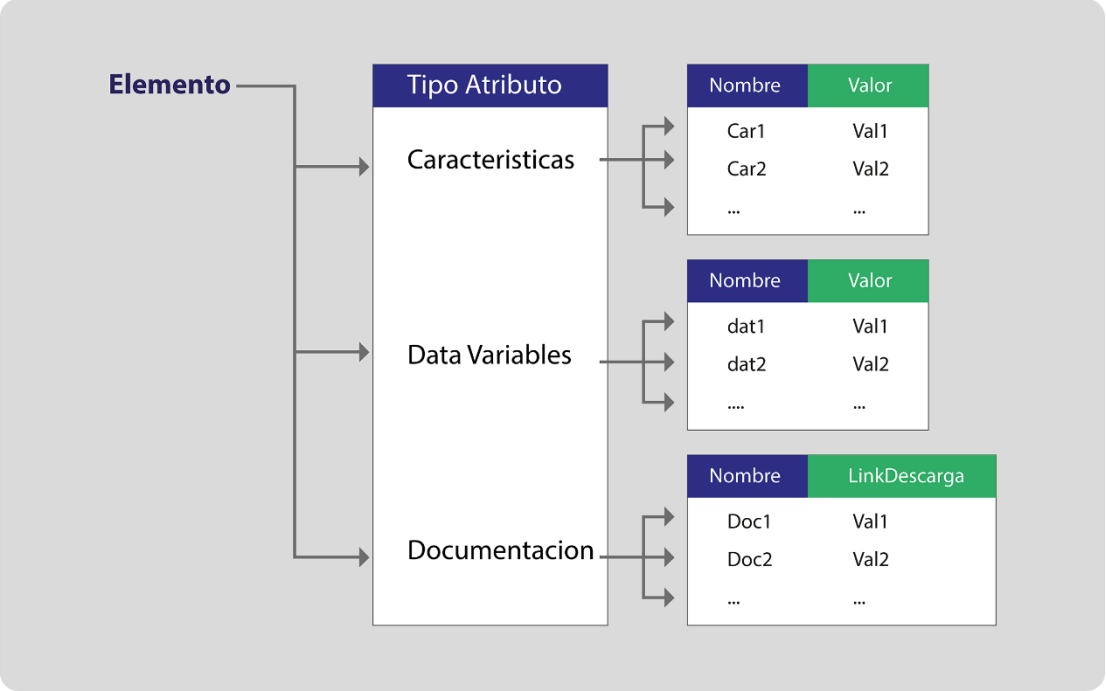

Ilustración 3.37 Jerarquía por tipo de atributo.

Finalmente con la estructura definida, las consultas tendrán tres campos de información: la central termoeléctrica, que en este momento es solo una, pero pensando a futuro incorporar nuevas, el código *KKS* del elemento que se encuestará y el tipo de datos que se busca, característica técnica, datos variables, o documentos.

3.6.3 Conexión y descarga de información

Pi System incorporó el año 2015 una *API* para conexión web, la que con los permisos adecuados permite navegar a través de consultas web los árboles de información generados a través de *PI AF*. Esta herramienta se ajusta perfectamente con los requerimientos funcionales, por lo que se solicitó al grupo del *CMD* de ENDESA instalar este complemento con los permisos necesarios para consultar la estructura que se construyó en conjunto con ellos.

Esta *API* llamada *PI WEB API*, es bastante simple de utilizar. Con los permisos adecuados de conexión, es posible realizar consultas agregando campos directamente a la *URL*.

El sistema funciona a través de *WebID*. Un *WebID* es un identificador único que utiliza *PIWebAPI* para identificar directamente un elemento dentro de la jerarquía de *PI AF*.

De esta manera para poder obtener los atributos del elemento buscado, se hace necesario separar la búsqueda en dos consultas web: La primera para obtener el *WebID* del elemento buscado y una segunda para buscar los atributos éste.

La estructura de búsqueda se obtuvo de la documentación de *PIWebAPI* [9]. Se necesita entonces el encabezado que corresponde a la dirección IP del servidor *PIWebApi*, y el nombre único del elemento a buscar. (En la estructura de jerarquía el nombre ya es único por corresponder al código *KKS* del elemento).

La ilustración 3.38:"Estructura de consulta web para búsqueda de *WebID* de elemento", muestra cómo queda definida la función de búsqueda de *WebID* a partir de un código *KKS*.

```
public string piWebApiServer ="https://10.115.135.34";
public string WebID_Central="E02-0M_RMxik-Co4KUiHw9qwwMvvoegu5RGCTQBQVsAACAQ0hMRU5EQ0";
string KKSSeachFormat="{0}/piwebapi/elements/{1}/elements?nameFilter={2}&searchFullHierarchy=true";
string KKSwebID;

void BuscarWebIDElemento(string _kks){
    string elementUrl = string.Format (KKSSeachFormat, piWebApiServer, WebID_Central, _kks);
    System.Net.WebClient client;
    client = new System.Net.WebClient();
    client.DownloadStringCompleted+= new DownloadStringCompletedEventHandler(asyncWebRequest_DownloadStringCompleted);
    AsyncWebRequest(elementUrl,client);
}
```

Ilustración 3.38 Estructura de consulta web para búsqueda de *WebID* de elemento.

El comando *SearchFullHierarchy* permite buscar por toda la rama, así si no hay ningún eslabón que se repita llegará inmediatamente al resultado. Es aconsejable acotar la búsqueda por temas de optimización y velocidad por lo que se agregó el *WebID* de la central, que permite acotar la búsqueda sólo a la rama perteneciente a la central en cuestión.

Por otro lado es importante mencionar que la búsqueda web se hace de manera asincrónica, para asegurar que no se peguen los procesos. Esto es particularmente relevante pues se debe esperar que la consulta web retorne el *WebID* del elemento, para realizar una segunda búsqueda preguntando por sus atributos. Una vez Obtenido el *WebID* del elemento, se genera inmediatamente otra consulta preguntando por sus atributos.

```
public string piWebApiServer ="https://10.115.135.34";
string KKSwebID;
string AttributeSearchFormat="{0}/piwebapi/elements/{1}/attributes?categoryName={2}";

void BuscarAtributosElemento(string _kks,string _tipoAtributo){
    string  attributesUrl = string.Format(AttributeSearchFormat, piWebApiServer, KKSwebID,_tipoAtributo);
    System.Net.WebClient client;
    client = new System.Net.WebClient();
    client.DownloadStringCompleted+= new DownloadStringCompletedEventHandler(asyncWebRequest_DownloadStringCompleted);
    AsyncWebRequest(attributesUrl,client,_tipoAtributo);
}
```

Ilustración 3.39 Estructura de consulta web para búsqueda de atributos de elemento.

La estructura es similar a la anterior. Esta vez se utiliza el *WebID* del elemento obtenido anteriormente y se agrega un filtro para que retorne sólo los atributos correspondientes a un tipo de categoría, de esta manera nuevamente se ahorra tiempo de consulta y procesamiento, tanto de la aplicación como del servidor.

3.6.4 Manejo y despliegue de información

Con el fin de ordenar la información recibida de las consultas web y de poder utilizar la aplicación sin la necesidad de conexión constante a internet, es que se hace necesario un sistema para almacenar de manera local toda la información que se va descargando a medida que se utiliza la aplicación.

Con este fin se utiliza la base de datos local *SQLite* (**apartado 2.3.2**), que permite ir guardando los datos que se descargan desde las consultas web.

La base de datos se construye en tres tablas: Características Generales, Datos Variables y Documentos. Cada una de ellas tiene los parámetros *KKS* del elemento, Nombre del atributo y Valor del atributo.

3.6.4.1 Características generales

Al seleccionar características generales de un elemento, se hará una búsqueda en la base de datos local por *KKS*, si no hay coincidencias quiere decir que no hay datos guardados localmente y automáticamente tratará de conectarse al servidor. Al no haber datos en la nube o no haber conexión, se desplegará un mensaje.

En caso de que haya datos en la base de datos local estos se desplegarán en forma de lista mostrando el nombre del atributo y su valor. Además existe un botón que permite forzar la búsqueda web.

3.6.4.2 Datos Variables

Funciona igual que la opción de características generales, con la diferencia que estos datos varían en el tiempo, por lo que es importante actualizarlos constantemente. Por lo mismo, acompañado de los datos se despliega la fecha de la última actualización, así el usuario puede ver la fecha y hora exacta de esta última.

3.6.4.3 Documentos

Al seleccionar Documentos se debe desplegar una lista con todos los documentos relacionados al elemento. Cada objeto de esta lista debe ser un botón interactivo, que permita abrir el documento seleccionado.

Para ahorrar recursos, se debe comprobar si el archivo ya ha sido descargado antes o es necesario descargarlo, para esto existe una carpeta asignada en el dispositivo móvil donde se almacenan todos los documentos.

Luego, una vez descargado el archivo o habiendo coincidencia de búsqueda, se utiliza la función de Unity3D *Application.OpenURL()* que permite abrir automáticamente el archivo en el programa externo por defecto para cada tipo de extensión. Esto es importante ya que de esta manera se evita tener que programar lectores de archivo dentro de la aplicación.

3.7 Sistema de búsqueda por teclado y códigos QR

La aplicación consta de tres maneras de buscar elementos. La primera es navegando la maqueta virtual y seleccionando el elemento dentro de ésta, proceso que fue descrito anteriormente.

Otra opción de búsqueda es la búsqueda por teclado, en la que el usuario debe ingresar el código *KKS* o nombre del elemento en cuestión, desplegándose una lista de posibles resultados. Al seleccionar uno, se hace una navegación de la cámara hacia el elemento en la maqueta virtual desplegando automáticamente su menú de elemento.

Finalmente la última opción está pensada para el usuario que está recorriendo la planta termoeléctrica y quiere revisar la información de un equipo específico. En este caso se utiliza un código *QR* pegado al equipo físico, que al enfocarlo con la cámara del dispositivo se posiciona en el elemento virtual desplegando su información.

3.7.1 Búsqueda por teclado

Como se vio anteriormente al iniciar la aplicación se hace un barrido por la jerarquía de elementos y se rescatan todos los elementos en una lista que se guarda en el script *ControlObjetos*.

Como cada elemento guarda su código *KKS* y Nombre, hacer la búsqueda de elementos por teclado se reduce a buscar coincidencias de Nombre o *KKS* respecto a los elementos en la lista guardada.

Una vez comparado el nombre ingresado con los elementos de la lista y se devuelve una lista con todas las coincidencias.

```
//Busca por nombre y retorna la cantidad de elementos encontrados
public int busquedaNombre(string _name){
    PanelTopInput.text = _name;
    indice = -1;
    ElementosEncontrados.Clear();
    for(int i=0;i<Elementos.Count;i++){
        string s1 = _name.ToUpper();
        string s2 = Elementos[i].NombreElemento.ToUpper();
        string s3 = Elementos[i].KKS.ToUpper();
        if(s2.Contains(s1) || s3.Contains(s1)) ElementosEncontrados.Add(Elementos[i]);
    }
    return ElementosEncontrados.Count;
}
```

Ilustración 3.40 Búsqueda por nombre o KKS.

Los resultados pueden darse por búsqueda completa o parcial de código *KKS* o nombre. Con la lista de coincidencias se despliega una lista en pantalla con los elementos posibles.

Al seleccionar un elemento de esta lista, se realiza un viaje de la cámara hacia el lugar del elemento seleccionado dentro de la maqueta virtual.

En caso que no exista coincidencia un mensaje se desplegará en pantalla para que el usuario pueda corregir y volver a realizar la búsqueda.

3.7.2 Búsqueda por lectura de código QR

La idea es que enfocando con la cámara del dispositivo móvil un código *QR* adherido al equipo físico, la aplicación pueda automáticamente identificar y desplegar en pantalla el equipo enfocado; dando acceso inmediato al menú del elemento y a su ubicación en la maqueta virtual.

Para leer códigos *QR* se utilizara el complemento *ZXing*, complemento gratuito que permite decodificar varios tipos de matrices de puntos entre ellos *QR*. (**Apartado 2.3.3**)

Zxing está bastante optimizada para utilizar pocos recursos, pero al tratar de decodificar la imagen, cada *frame* hace que la aplicación se ralentice considerablemente. Por esto se decide ejecutar esta decodificación en un hilo aparte de manera asincrónica.

```
void Start () {
    //maximo numero de hilos
    ThreadPool.SetMaxThreads(5, 5);
}
void Update () {
    ThreadPool.QueueUserWorkItem(new WaitCallback(DecodeQR));
}
```

Ilustración 3.41 Uso de *ThreadPool* y limitar número de hilos en paralelo.

Puesto que se creó un hilo en cada *frame*, es importante limitar el número de hilos que estarán corriendo simultáneamente, ya que se podría generar una cantidad de hilos que la aplicación no pudiese manejar. En este caso se establece un máximo de 5 hilos simultáneos.

Finalmente al encontrar y decodificar un código, este se compara con los *KKS* de los equipos en la aplicación y de haber coincidencia, la cámara navega automáticamente hasta la posición del elemento en la maqueta virtual y se despliega el menú de información del elemento.

3.8 Pruebas de rendimiento

Concluida la etapa de programación base, es decir, aquella que se remite exclusivamente a la funcionalidad de la aplicación y a los procesos de interacción con el usuario, es necesario realizar una fase de pruebas para sopesar cualquier inconveniente o error que presente la aplicación. Para ello se compiló y exportó la aplicación para hacer pruebas funcionales y de rendimiento en PC, en la cual la aplicación se comportó como se esperaba.

El paso siguiente fue hacer el mismo proceso para dispositivos móviles, para lo cual fue instalada la aplicación en un *Samsung Galaxy Tab 3* [10].

Al momento de ejecutar la aplicación, ésta mostro dificultades:

- Lentitud en la navegación de los menús.

- Navegación de la maqueta virtual casi nula (demasiado lenta).
- Problemas con la detección de los gestos *touch*.

En orden a resolver los problemas presentados, se decidió ir acotando las funcionalidades de la aplicación para identificar la fuente del problema, a través de variantes de la aplicación.

3.8.1 Variables de la aplicación

- En un primer intento se quiso descartar que fuera un problema con los gestos táctiles, por lo que se montó una aplicación sin *TouchScript* y con un recorrido pregrabado de la planta, pero las dificultades persistieron.
- Luego fue montada una aplicación que sólo permitía la navegación de la planta, eliminando todo código de programación excepto un recorrido pregrabado. Esta aplicación fue implementada para conocer si el origen del problema era la programación o la renderización de la planta.

Luego de esta implementación se pudo concluir que el problema tenía que ver con el renderizado de la maqueta virtual y no con problemas de lógica de programación.

3.8.2 Detección del problema específico

Una vez identificada como causa de los problemas de rendimiento el renderizado, es necesario realizar otro tipo de pruebas para aislar el problema específico.

El siguiente paso, entonces, fue cambiar el Tablet por uno de última generación. Se escogió un *Samsung Galaxy Tab Active* [11]. Con este nuevo Tablet efectivamente la aplicación mejoró su comportamiento, pero aún lejos de una velocidad óptima de funcionamiento.

Tabla 3.1 Características técnicas Samsung Galaxy Tab Active.

Procesador	*Qualcomm Snapdragon 400 QuadCore 1.2GHz*
Resolución	*WXVGA 1200x800 Pixeles*

Procesador Grafico	*Adreno 305*
RAM	*1.5GB*
Memoria	*16GB*

Como se puede ver en la tabla anterior, si bien las características técnicas de un Tablet de última generación como el *Tab Active* son muy buenas, están lejos de compararse con la capacidad de un computador. A continuación se hace la comparación de un PC estándar de gama media con el Tablet anteriormente descrito:

Tabla 3.2 Especificaciones técnicas PC vs Tablet Samsung Tab Active.

	Tablet Samsung Tab Active	***PC Samsung NP300V4A***
Procesador	*Qualcomm Snapdragon 400 QuadCore 1.2[GHz]*	*Intel Core i5 2450M (Dual-core / 2500 MHz - 3100 MHz)*
Resolución	*1200x800 [Pixeles]*	*1366x768 [Pixeles]*
RAM	*1.5[GB]*	*4[GB]*
Memoria	*16[GB]*	*500[GB]*
Procesador Gráfico	*Adreno 305*	*Intel HD Graphics 3000 (Integrada)*
GPU Clock	*400 [Mhz]*	*850[Mhz]*
Pixel Fill Rate	*800[MPixeles/s]*	*1.7[GPixeles/s]*

Como se puede observar un computador en términos de procesador es 2,5 veces más rápido que un Tablet.

Además, existe una gran diferencia en términos de procesamiento gráfico, donde no sólo se duplica la velocidad de procesamiento, sino que también se duplica la velocidad de rellenado de pixeles.

Por lo que la única solución es optimizar lo más posible la maqueta virtual y su proceso de renderizado.

3.9 Optimización de la aplicación TR3

Como se menciona en el capítulo anterior, para optimizar la aplicación TR3 es necesario optimizar el proceso de renderizado de la maqueta virtual, para lo que se hace necesaria una comprensión acabada de este tema en particular.

3.9.1 Proceso de renderizado

El proceso de renderizado en términos simples se entiende como el cálculo matemático que hace la tarjeta de video o *GPU* para dibujar los pixeles en pantalla. Pero el proceso es un poco más complejo que eso.

Para la computadora los objetos no son más que un grupo de vértices y datos de colores o texturas.

Es trabajo entonces del *CPU* y el *GPU* transformar estos arreglos de datos y convertirlos en imágenes en pantalla.

Es el *CPU* quien hace la petición al *GPU* que dibuje algo en pantalla, para lo que le debe pasar la información de los *meshes* que tiene que dibujar (*draw calls*) y los parámetros con los que tiene que dibujarlos (*renderState*). Estos incluyen: *vertex* y *pixel shader*, textura, material, iluminación, transparencia, otros.

Sin profundizar en como procesa los datos el *GPU* (*Pipeline*) y sin considerar operaciones altamente costosas (como el cálculo de la iluminación y sombreado), comparativamente la velocidad del *GPU* es varias veces mayor a la velocidad de entrega de peticiones por parte del *CPU*. De esta manera, si las peticiones son muchas y los cálculos del GPU simples, este último queda ocioso mucho tiempo. Para evitar este cuello de botella, a continuación se describen una serie de posibles mejoras que permiten minimizar los tiempos de renderizado.

3.9.2 Mejoras clave del proceso de renderizado

En el **apartado 3.3.3** se habló de la generación de *Lightmaps* para evitar todo el cálculo de iluminación y sombreado, reduciendo enormemente el tiempo de renderizado y por ende mejorando considerablemente la velocidad de la aplicación.

Si bien generar *Lightmaps* es la práctica más común al crear aplicaciones, por el gran impacto que esto tiene en cuánto optimización y velocidad de la aplicación, hay otras intervenciones que se pueden hacer para mejorar aún más estos tiempos de respuesta, las que se enumeran y describen a continuación.

3.9.2.1 Occlusion culling

Normalmente todos los objetos en la escena son enviados a renderizar, estén dentro del área de vista de la cámara o no. Lo mismo pasa con los objetos que están siendo ocluidos por otros objetos que están en frente en relación a la cámara.

Para evitar estos renderizados adicionales existe *Occlusion culling*, que permite desactivar el renderizado de los objetos que no están siendo mostrados por la cámara. Es similar a trabajar por *layers* (**apartado 2.1.2**) pero es una oclusión un poco más dinámica, que permite desactivar o descartar el renderizado de objetos cuando estos no están en línea de vista con la cámara o cuando están siendo tapados u ocluidos por otros objetos.

Unity3D viene con la opción de agregar esta característica, para ello se deben marcar en Unity3D como estáticos los elementos que se quieran incluir y seleccionar un área de oclusión.

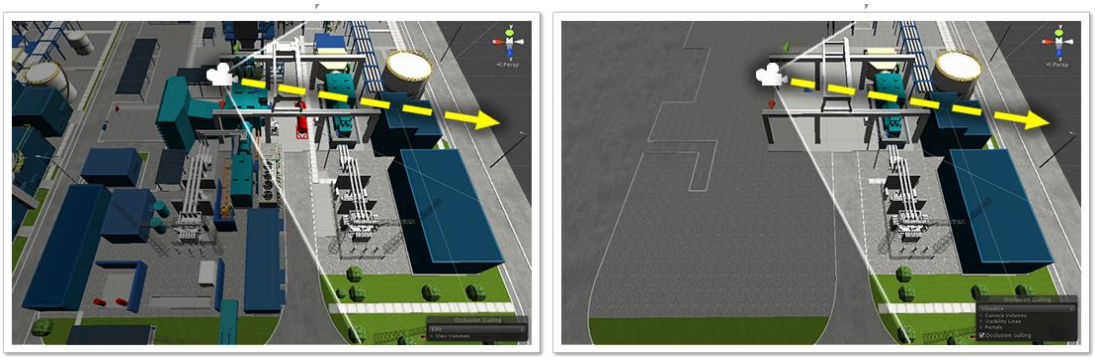

Ilustración 3.42 Comparación escena sin y con oclusión.

Teniendo en cuenta que *Occlusion culling* descarta los *meshes* o geometrías completas, si una geometría es muy grande, aunque sólo se vea una porción pequeña de ella, la geometría completa será renderizada.

3.9.2.2 Disminución de peticiones DrawCalls y RenderStates

Como se mencionó anteriormente, las peticiones del *CPU* al *GPU* de renderizar por lo general son más lentas que el mismo renderizado del *GPU* (suponiendo procesos de renderizado simples), por lo que disminuir estas peticiones es esencial.

El *CPU* tiene que enviar la información de lo que quiere que se dibuje, las geometrías o *meshes* (*DrawCalls*) y los parámetros con que ellos deben ser dibujados (*RenderStates*), y repetir esta acción para cada geometría a dibujar y por cada material distinto que exista.

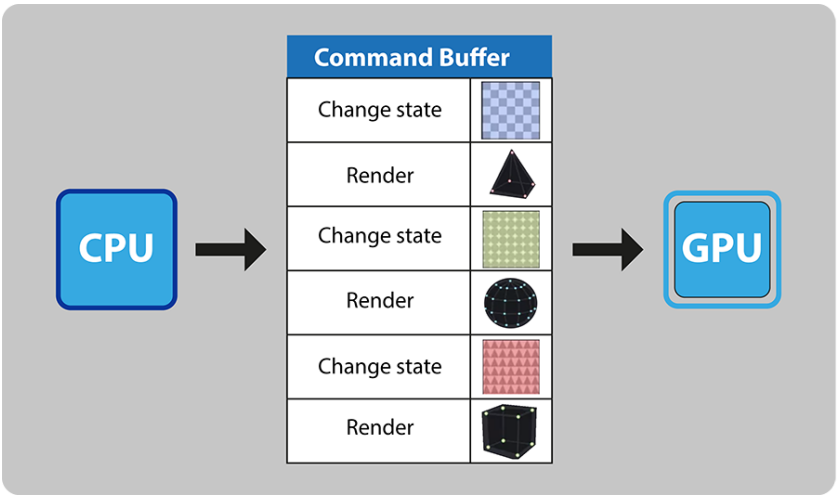

Ilustración 3.43 Peticiones por cada *mesh* a renderizar.

Por lo que un punto importante es revisar el no tener materiales ni texturas duplicadas y también tratar de agrupar muchas texturas en un sólo material. A este proceso se le llama generar atlas de texturas y mapas UV.

De esta manera se puede agrupar a todos los *meshes* que se van a dibujar con los mismos parámetros y así reducir la cantidad de peticiones de *RenderStates* considerablemente.

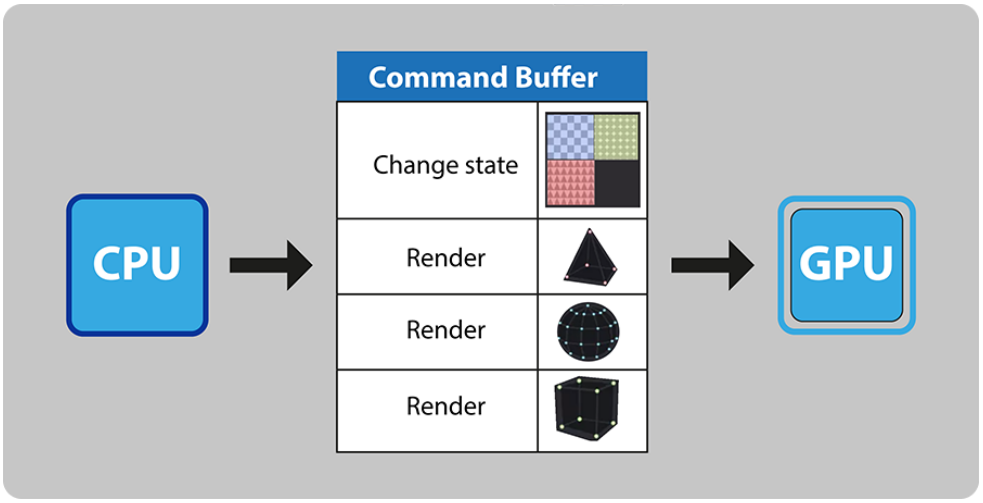

Ilustración 3.44 Petición de un *RenderState* para varios *meshes* a renderizar.

Por otro lado para disminuir la cantidad de peticiones *DrawCalls* se debe disminuir la cantidad de *meshes* o geometrías en la escena.

La opción tomada aquí fue agrupar todas las geometrías que compartían el mismo material en una sola gran geometría. De esta manera se pueden disminuir los *DrawCalls* de cientos a unos cuantos.

Para tomar esta decisión es muy importante tener claro qué los objetos no se van a volver a mover, ya que una vez que los objetos son agrupados en una sola geometría, no podrán volver a ser manipulados individualmente de nuevo.

Es por eso que solo los elementos no interactivos y estáticos deben pasar por este proceso.

Si bien el ejecutar esta decisión de unir las geometrías reduce considerablemente los *drawcalls*, deja inutilizado el proceso de *Occlusion culling*. Esto debido a que *occlusion culling* descarta las geometrías que están totalmente fuera de pantalla, pero al unirlas siempre quedará algún trozo de geometría en pantalla, por lo que *Occlusion culling* no podrá descartarlas.

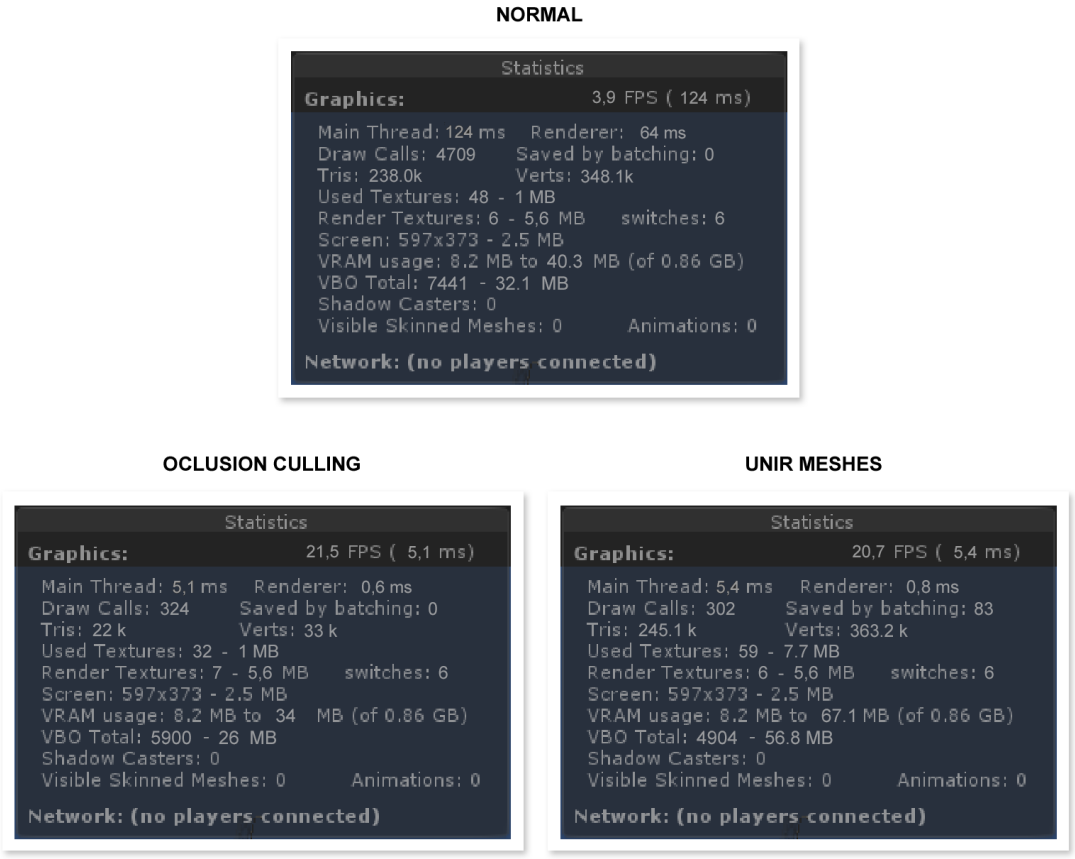

Ilustración 3.45 Comparativa mejoras: Rendimiento normal, Rendimiento con *Occlusion culling* y Rendimiento uniendo *meshes.*

Como se puede ver en la figura anterior, *Occlusion culling* puede llegar a ser tan efectivo como la reducción de *drawcalls,* esto dependiendo de cuantas geometrías se encuentren visibles en pantalla. (**Apartado 3.9.2.1**)

3.9.2.3 Reducción de texturas y cantidad de vértices

Finalmente otra mejora posible es disminuir el tamaño de las texturas para que ocupen menos memoria en *VRAM* y disminuir la cantidad de vértices de las geometrías.

Si bien ambas opciones ayudan y aportan a la disminución de tiempo de renderizado, no se convierten en un factor relevante mientras no se solucione el cuello de botella

que generan las peticiones de renderizado. Además de tener repercusión directa en la calidad de las geometrías y texturas en pantalla.

Sin embargo, siempre tiene que existir un equilibrio entre *DrawCalls* y cantidad de vértices en un escenario. Como se mencionó anteriormente el *GPU* puede trabajar sin problema en cientos de miles de vértices, pero la cantidad se convierte en un problema si se piensa en millones de ellos.

3.9.3 Conclusión de la mejora

Como se puede ver el conocer en detalle el proceso de renderizado es un factor clave a la hora de optimizar recursos.

Esto es aún más importante cuando se planea generar aplicaciones para dispositivos móviles, ya que estos últimos presentan capacidades mucho más limitadas en términos de renderización y *GPU* que los computadores de hoy en día.

Utilizar luces estáticas quemadas y mapas de oclusión son elementos esenciales para reducir el procesamiento de renderizado.

Se puede ir más allá aún, entendiendo bien el proceso de renderizado se pueden aprovechar al máximo los recursos disponibles. Es así como reestructurando la maqueta virtual se pudo reducir los *Draw Calls* de 4709 a 302. Lo que significó una mejora del 500%, de 3,9[fps] a 20,7[fps] sin pérdida de calidad ni reducción de polígonos.

4 CONCLUSIONES

Actualmente para dar respuesta a posibles fallas en terreno u obtener información sobre el funcionamiento de un equipo físico en una planta industrial se debe recurrir a diversas fuentes de información, variables en tiempo real, provenientes de algún sistema de monitoreo, planos y documentos almacenados en bodega, guías de trabajo y otros recursos en papel, etc.

El objetivo de TR3 fue crear una aplicación para dispositivos móviles que permita centralizar información y documentación relevante en una maqueta virtual georreferenciada, como herramienta para la toma de decisiones de los trabajadores en terreno.

Así la aplicación permite mejorar los procesos de mantención y supervisión, aumentando el rendimiento y la productividad. Entendiendo a estos últimos como el ahorro de tiempo que generará en los trabajadores el uso de la aplicación en la ejecución de sus tareas cotidianas, sin prejuicio de mantener intactos sus estándares de calidad.

En relación con el desarrollo de TR3, optar por Unity3D como plataforma de desarrollo fue ventajoso ya que las prestaciones otorgadas por el software son de alto alcance en cuanto a programación y conectividad, además de proporcionar un ecosistema completo para el desarrollo de contenido interactivo en 3D.

Otro aspecto relevante, refiere a la compresión del proceso de renderizado. Hoy en día los desarrolladores de software han obviado las etapas de este proceso debido a que en la actualidad las computadoras y dispositivos móviles son de altas prestaciones. Sin embargo, dependiendo de la complejidad del desarrollo, se hace del todo relevante optimizar este proceso lo más posible, no sólo para resolver posibles problemas de respuesta de la aplicación sino que también para disminuir considerablemente el uso de batería en el proceso de renderizado en dispositivos móviles.

En esta memoria se muestra una breve descripción del proceso de renderizado, enfocándose en puntos clave a la hora de optimizar este proceso, considerando no sólo

los aspectos de iluminación y *Occlusion culling*, sino que detectando posibles cuellos de botella en la comunicación del *CPU* con el *GPU*. En este último caso se muestran modos de combatir esto, reduciendo *DrawCalls* y *RenderStates* y exponiendo finalmente los resultados prácticos de estas mejoras.

Finalmente es importante comprender que si bien en principio es responsabilidad del cliente disponibilizar los datos, parte importante del trabajo, del negocio y del éxito de la aplicación está relacionado con entender la estructura de información que maneja cada cliente, lo que exige trabajar en conjunto para abrir espacios de comunicación interpersonal y convenir aspectos técnicos.

Si bien esto significa muchas veces trabajo adicional en el estudio y adaptación al funcionamiento de sus plataformas, significa también dar un enorme valor agregado a las propuestas, entregando al cliente una solución adaptada a sus sistemas y necesidades.

4.1 Trabajos Futuros

El desarrollo y puesta en marcha de este proyecto considera el modelado tridimensional de la planta industrial, así como también el acceso y los protocolos de comunicación con los sistemas del cliente, con lo cual se generan nuevas oportunidades de desarrollo.

- Tres proyectos adicionales en los que se está trabajando:
    - El primero y más relevante, utilizando los protocolos de comunicación que existen con los sistemas del cliente, hace posible que TR3 pase de solo recoger y desplegar información a actuar sobre las variables de control de los elementos de la planta industrial.
    - Segundo, utilizar el sistema de georreferenciación del dispositivo móvil para ubicar al trabajador dentro de la central termoeléctrica. Esto pensado por un lado para la auto-referenciación del individuo dentro de la planta, pero principalmente para enviar esta posición a una plataforma central, permitiendo tener un sistema de gestión y monitoreo de personal en terreno.

    - Tercero, crear una plataforma de edición, conectada a la aplicación principal, que permita generar y editar los equipos de una central. Este proyecto tiene como objetivo poder masificar el producto al permitir al usuario crear, idealmente desde cero, la reconstrucción de una planta industrial.

- Mejora en la comunicación con los servidores del cliente:
    - Esta está orientada a mejorar la comunicación cambiando los sistemas "*Pull*", donde el usuario consulta al servidor por nueva información cada vez que lo requiere, por sistemas "*Push*" donde el cliente solo se suscribe al servidor con la información que quiere recibir, y es este último el encargado de enviar información cuando hay algún cambio. Esto es de principal relevancia en los temas relacionados a las alarmas y cambios de estado, ya que es posible ahorrar gran cantidad de recursos al no tener que estar el usuario consultando por cambios constantemente, sino que es el servidor, quien al experimentar un cambio, lo informa puntualmente al usuario suscrito.

5 BIBLIOGRAFIA

1. ***Unity Technologies, Unity.*** *[en línea] <https://unity3d.com/es/unity> [consulta: 15 mayo 2017]*

2. ***Valentin Simonov, TouchScript: Multitouch library for Unity.*** *[en línea] <https://github.com/TouchScript/TouchScript/wiki> [consulta: 15 mayo 2017]*

3. ***SQLite consortium, SQLite: Features of SQLite.*** *[en línea] <https://www.sqlite.org/features.html> [consulta: 15 mayo 2017]*

4. ***ZXing.Net team, ZXing.*** *[en línea] <https://zxingnet.codeplex.com/> [consulta: 15 mayo 2017]*

5. ***Autodesk, 3DS Max.*** *[en línea] <https://latinoamerica.autodesk.com/products/3ds-max/overview> [consulta: 15 mayo 2017]*

6. ***Autodesk, Maya.*** *[en línea] <https://latinoamerica.autodesk.com/products/maya/overview> [consulta: 15 mayo 2017]*

7. ***OSIsoft, PI System overview.*** *[en línea] <http://www.osisoft.com/pi-system/> [consulta: 15 mayo 2017]*

8. ***OSIsoft, PI Interfaces list.*** *[en línea] <http://www.osisoft.com/pi-system/pi-capabilities/pi-system-connections/pi-Interfaces/> [consulta: 15 mayo 2017]*

9. ***OSIsoft, PI Web API.*** *[en línea] <https://techsupport.osisoft.com/Products/Developer-Technologies/PI-Web-API/Overview> [consulta: 15 mayo 2017]*

10. ***SAMSUNG, Galaxy Tab 3.*** *[en línea] < http://www.samsung.com/es/consumer/mobile-devices/tablets/galaxy-tab/SM-T2100ZWAPHE/> [consulta: 15 mayo 2017]*

11. ***SAMSUNG, Galaxy Tab Active.*** *[en línea] <http://www.samsung.com/es/business/business-products/tablets/tablets/SM-T360NNGAPHE> [consulta: 15 mayo 2017]*

Anexo A. LIBRERIAS EXTERNAS

En esta sección se explica el proceso de instalación y funcionamiento básico de las librerías externas utilizadas en el desarrollo de la aplicación TR3.

A.1 TouchScript

A.1.1 Implementación de la librería TouchScript

Una de las ventajas de esta librería es que el proceso de instalación es bastante simple.

La librería está disponible para su descarga desde el sitio oficial de complementos de Unity3D, por lo que sólo se requiere buscar por *TouchScript* en el *AssetStore*.

Link directo: *https://www.assetstore.unity3d.com/en/#!/content/7394*.

Una vez descargado simplemente debe abrirse desde Unity3D, lo que incorporará automáticamente todos los archivos necesarios al proyecto.

A.1.2 Funcionamiento básico de la librería TouchScript

El *plugin TouchScript* se compone de dos partes o clases principales: *TouchManager* que es la encargada de detectar y manejar todos los *input touch* en la escena y *GestureManager* que se encarga de reconocer si es un *Gesture* válido o no.

A.1.2.1 TouchManager

Como se mencionó anteriormente *TouchManager* es la clase que está encargada de procesar todos los *input touch* que se generan en un *frame*.

Primero remapea todos los *input* para que calcen con las coordenadas de pantalla, luego procesa y clasifica estos *input* en *TouchesBegan*, *TouchesMoved*, *TouchesEnded* y *TouchesCancelled*, finalmente le pasa esta información a *GestureManager*.

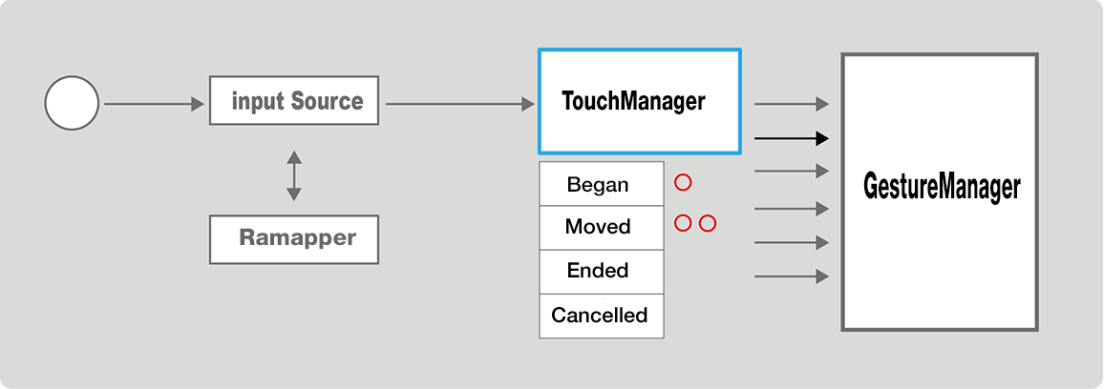

Ilustración A.1 Esquema funcionamiento *TouchManager*.

A.1.2.2 GestureManager

GestureManager toma los *touch events* de *TouchManager* y los clasifica de acuerdo a los posibles gestos que se están desarrollando.

Si hay algún objeto o script interesado, suscrito a algún gesto táctil, *GestureManager* es el encargado de pasarle la información de este gesto o del posible gesto en desarrollo.

Dependiendo de en qué tipo de gesto se esté interesado, *GestureManager* puede pasar distintos tipos de evento al o los objetos suscritos.

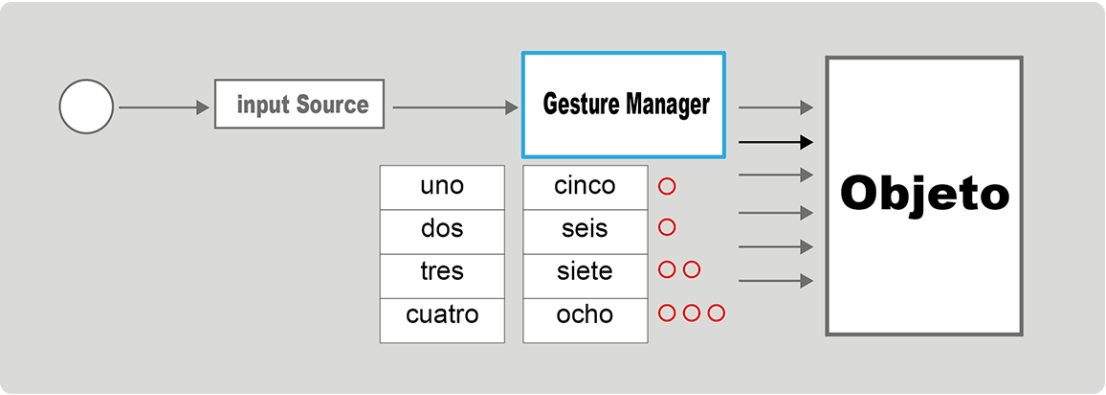

Ilustración A.2 Esquema funcionamiento *GestureManager*.

A.1.3 Tipos de Gestures

Los *Gestures* o gestos táctiles son *inputs* táctiles que obedecen o representan un movimiento "gesto" sobre una superficie táctil.

Dentro de los *Gestures* se definen dos categorías, **los gestos discretos y los continuos**:

• **Los gestos discretos** son reconocidos e inmediatamente terminados, un ejemplo de gesto discreto es el *Tap*. Este es un gesto discreto que se dispara cuando el usuario toca la pantalla.

• **Los gestos continuos** continúan enviando eventos cuando se le reconoce, un ejemplo de gesto continuo es el *Pan*. El *Pan* es un gesto continuo que consiste en desplazar el dedo por la pantalla. Este se activa cuando el movimiento comienza, enviando deltas de cambio mientras el movimiento continúe.

Un *Gesture* es una máquina de estados. Reconoce la entrada táctil y cambia de estado cuando se reconoce un patrón. Un *Gesture* puede estar en uno de los siguientes estados:

- Posible: el *Gesture* es posible pero no se ha reconocido ningún patrón aun.
- *Began* – continuo : el Gesture comenzó.
- *Changed* – continuo: el gesto se actualizo.
- *Ended*- continuo: el gesto termino.
- *Canceled*: el gesto fue cancelado por el sistema.
- *Failed*: no se pudo reconocer el patrón del gesto o se hico fallar por otro gesto.
- *Recognized* - discreto: El gesto fue reconocido.

A.1.3.1 Gestures incorporados en TouchScript

Como es señalado anteriormente, *TouchScript* trae la opción de crear *Gestures*, pero esta librería viene con la mayoría de gestos típicos ya incorporados.

A continuación se presentan los *gestures* incorporados con los eventos y mensajes que estos arrojan.

Tabla A.1 Gestos discretos.

Gesture	Eventos	Mensajes
Tap	*Tapped*	*OnTap*
Press	*Pressed*	*OnPress*
Release	*Released*	*OnRelease*
LongPress	*LongPressed*	*OnLongPress*
Flick	*Flicked*	*OnFlick*

Tabla A.2 Gestos continuos.

Gesture	Eventos	Mensajes
SimplePan/Pan	*PanStarted* *Panned* *PanCompleted*	*OnPanStart* *OnPan* *OnPanCompleted*
SimpleScale/Scale	*ScaleStarted* *Scaled* *ScaleCompleted*	*OnScaleStart* *OnScale* *OnScaleComplete*
SimpleRotate/Rotate	*RotateStarted* *Rotated* *RotateCompleted*	*OnRotateStart* *OnRotate* *OnRotateComplete*
MetaGesture	*TouchBegan* *TouchMoved* *TouchEnded* *TouchCanceled*	*OnTouchBegan* *OnTouchMoved* *OnTouchEnded* *OnTouchCanceled*

A.2 SQLite

A.2.1 Implementación de la librería SQLite

A diferencia de la anterior, la incorporación de esta librería a Unity3D no es trivial, por lo que a continuación se hace una breve descripción de como incorporar *SQLite* a un proyecto en Unity3D:

- Descargar desde el sitio web de *SQLite http://www.sqlite.org/download.html* los *SQLite-DLL* que se encuentran en la sección binarios precompilados para Windows.
- Crear una carpeta en *Assets* llamada *Plugins* y a su vez en el caso de querer exportar la aplicación a Android, crear la subcarpeta *Android.*
- Los archivos descargados, *sqlite3.dll* y *sqlite3.def* se copian a la carpeta *Assets/Plugins/Android.*
- Se necesitan 2 *DLL* adicionales, *System.Data.dll* y *Mono.Data.Sqlite.dll* que se deben copiar desde *C:\Program Files (x86)\Unity\Editor\Data\Mono\lib\mono\2.0* a *Assets/Plugins.*

Ilustración A.3 Incorporación archivos necesarios de *SQLite.*

Siguiendo los pasos anteriores, la librería de *SQLite* se ha integrado correctamente al proyecto. A continuación se explica cómo utilizar *SQLite* en Unity3D.

A.2.2 Funcionamiento básico de la librería SQLite

Una vez incorporada la librería a Unity3D sólo hace falta crear una base de datos local y luego conectarse a ella.

Para crear una base de datos de *SQLite* se recomienda utilizar un programar externo: hay un sinnúmero de programas de creación y administración de bases de datos de *SQLite*.

No es relevante que programa administrador utilizar, lo que sí es importante es crear y guardar una base de datos, la que posteriormente se debe agregar al proyecto.

Así mismo, todas las operaciones de bases de datos se pueden hacer desde Unity3D. Por comodidad muchas veces se prefiere crear las tablas, con sus atributos a través de una de estas herramientas de administración. Y dentro del proyecto sólo leer y escribir en estas tablas.

Una vez dentro del proyecto, los scripts que se creen para leer o modificar la base de datos, necesitan incorporar las siguientes dependencias: *Using Mono.Data.Sqlite* y *Using System.Data*.

Finalmente se presenta un ejemplo básico de conexión y consulta a base de datos, donde las funciones o clases básicas a utilizar son *IdbConnection* para la conexión, *IdbCommand* para el *query* y *IdataReader* para atrapar las respuestas.

```
string connection;
IDbConnection dbcon;
IDbCommand dbcmd;
IDataReader reader;

public void ConsultaSimpleDB()
{
    string connection ="URI=file://"+_pathDB;
    dbcon = new SqliteConnection(connection);
    dbcon.Open();

    string sqlQuery = "SELECT * FROM _tabla WHERE _columna==_dato";
    dbcmd = dbcon.CreateCommand();
    dbcmd.CommandText = sqlQuery;

    reader = dbcmd.ExecuteReader();
    while (reader.Read())
    {
        string _datos = reader.GetString(0);
        Debug.Log("datos encontrados="+_datos);
    }
}
```

Ilustración A.4 Ejemplo consulta básica *SQLite.*

A.3 ZXing

A.3.1 Implementación de la librería ZXing

La incorporación de esta librería no es complicada solo se debe descargar *ZXing.NET* desde la página oficial *https://zxingnet.codeplex.com/* y luego copiar *zxing.unity.dll* desde la carpeta descargada a *Assets/Plugins* dentro del proyecto.

Los script que se agreguen para decodificar o codificar necesitan añadir el uso de *ZXing* (*using ZXing*).

A.3.2 Funcionamiento básico de la librería ZXing

Finalmente, un ejemplo básico para decodificar un código de barra dentro de una imagen es tan simple como lo que se presenta a continuación:

```
void LeerImagen()
{
    //instancia BarcodeReader
    IBarcodeReader reader = new BarcodeReader();
    Texture2D imagen;
    //trata de decodificar imagen
    var result = reader.Decode (imagen.GetPixels32 ());
    if (result != null)
    {
        Debug.Log(result.Text);
    }
}
```

Ilustración A.5 Ejemplo decodificación básica *ZXing.*

Anexo B. MANUAL DE USUARIO

Esta sección pretende explicar y ser una guía en el uso de la aplicación.

A continuación se describe el uso e interacción con las distintas interfaces que forman parte de la aplicación: Opciones de uso, Navegación y Método de utilización que tiene la aplicación.

B.1 Login

Al iniciar la aplicación la primera pantalla con la que el usuario se enfrenta es la de autenticación.

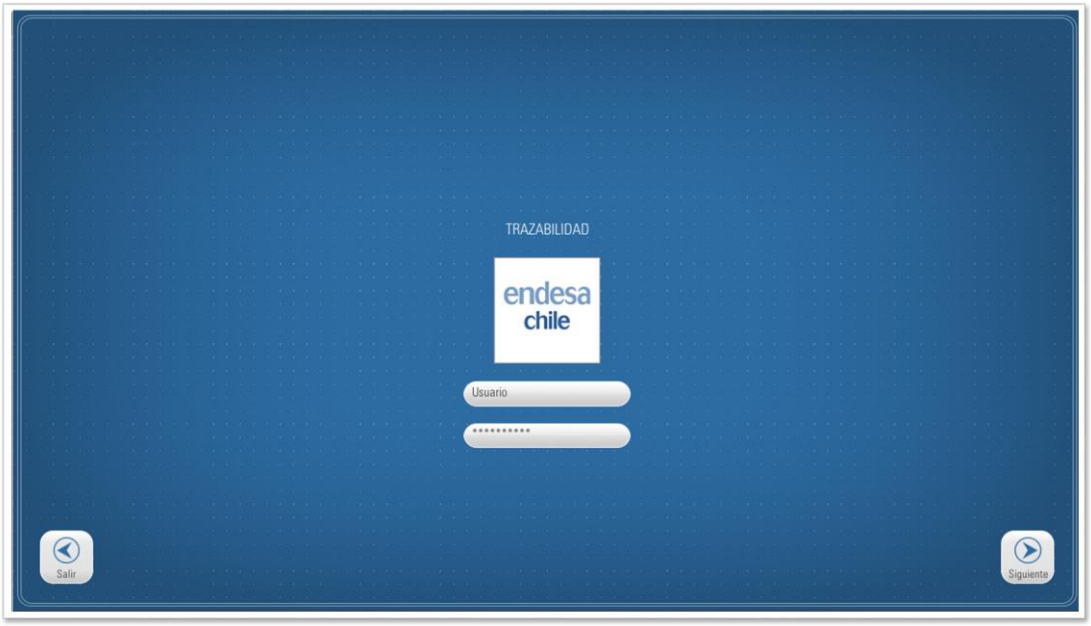

Ilustración B.1 Pantalla de *Login.*

En esta pantalla el usuario debe ingresar su nombre de usuario y contraseña, para lo que basta con tocar el área de texto para que se despliegue un teclado en pantalla.

Una vez correctamente ingresados los datos se debe presionar el botón "*Siguiente*" para proceder a la autenticación. (Ingresar a la aplicación)

Existe también la posibilidad de salir de la aplicación presionando el botón "*Salir*".

B.2 **Selección de planta (instalación, segunda pantalla)**

Aquí el usuario a través de gestos táctiles puede navegar a través de las distintas plantas productoras, viendo una fotografía y descripción general de cada una de ellas.

Ilustración B.2 Selección de instalaciones.

En la parte inferior hay una barra deslizante con imágenes pequeñas que se pueden seleccionar, la imagen seleccionada se muestra en la parte superior en tamaño grande desplegando información de la planta seleccionada.

Una vez seleccionada la planta que se desea revisar (no necesariamente en la que uno se encuentra), se debe presionar el botón "*Siguiente*", lo que lleva a la maqueta tridimensional de la planta seleccionada, y al menú principal de la aplicación.

B.3 **Barra superior**

La barra superior se compone de 3 partes, el icono en miniatura de la aplicación, una barra de búsqueda y el usuario actualmente conectado. Es un menú de acceso rápido, ya que se encuentra visible en todas las demás interfaces o pantallas.

Ilustración B.3 Barra de búsqueda.

El icono en miniatura de la aplicación permite volver de manera inmediata al menú principal de la aplicación.

La barra de búsqueda es una manera rápida de buscar un elemento por código identificador (*KKS*) o por su nombre.

Finalmente al lado derecho se encuentra la información del usuario conectado, aquí se puede cerrar sesión y conectarse con otro usuario o cambiar la planta a visitar.

B.4 **Menú Principal**

El menú principal despliega las 4 maneras de navegar a través de la aplicación. Navegar la central, buscar por código o por nombre, escaneo de códigos *QR*, y alarmas.

Ilustración B.4 Menú principal.

B.4.1 Recorrer la central

Esta opción permite recorrer la central de manera libre. Navegar y buscar elementos a medida que esta se va recorriendo.

Al seleccionar esta opción el usuario se enfrenta a una toma aérea de la central, la cual permite visualizarla en su totalidad.

Ilustración B.5 Vista de recorrido de la central.

En la navegación global existen 3 movimientos de interacción *TOUCH* (*Pan, Scale, Rotate*) que permiten respectivamente moverse, acercarse o alejarse y rotar la cámara. Todo esto dentro del levantamiento o planta tridimensional.

Una vez suficientemente cerca de algún elemento interactivo o seleccionable, se disponibilizan dos gestos *touch* adicionales (*LongPress y Press*).

Con *LongPress*, que es tocar un elemento por más de 1.5 segundos, la cámara se mueve y centra frente al elemento seleccionado, esto permite ver el elemento en detalle y navegarlo con los gestos de *Scale* y *Rotate*.

El *Press*, que es tocar un elemento, hace aparecer (despliega) un panel lateral con información del elemento en cuestión, además, el resto de la planta se vuelve transparente (o maqueta transparente) para facilitar la navegación y vista del elemento seleccionado.

B.4.2 Buscar elemento por nombre o código único

Buscar por nombre o *KKS* despliega en pantalla un campo de texto donde se puede ingresar el nombre o *KKS* de un elemento para hacer la búsqueda de éste.

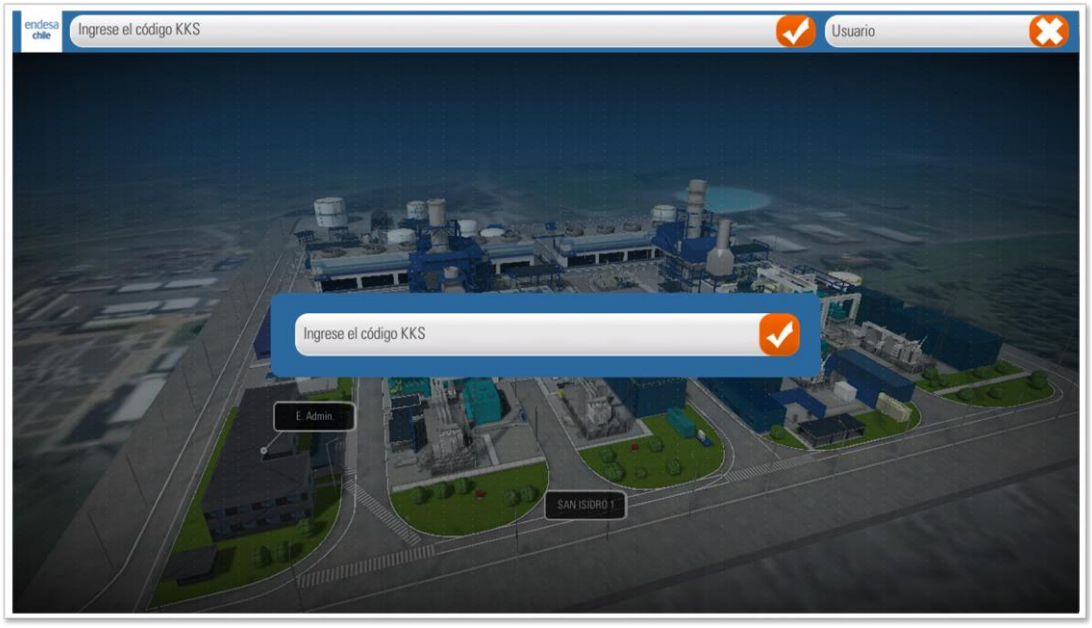

Ilustración B.6 Búsqueda por nombre o código KKS.

Al ingresar la búsqueda, si se busca por nombre existen tres alternativas: que el nombre no coincida con ninguno en la base de datos, en este caso la aplicación pide reingresar un nombre válido, que exista más de un elemento, con lo que se despliega un panel lateral con todas las coincidencias y finalmente que exista un solo elemento.

Al seleccionar un elemento o un código único válido, la pantalla cambia a la de navegación, con la diferencia que en el mapa se resalta un círculo destacando un sector de la planta (como se muestra en la siguiente figura).

Ilustración B.7 Destacar zona en el mapa.

Al presionar sobre este círculo, la cámara hace una navegación automática hasta el elemento seleccionado. Durante esta navegación se bloquean los gestos *touch*, deshabilitando la interacción con la aplicación. La idea detrás de esto, es que con la navegación de la cámara quede claro en qué sector de la planta se encuentra el elemento.

Una vez que la cámara queda centrada en su posición final frente al elemento, se reactivan todos los gestos *touch* y opciones de navegación.

B.4.3 Búsqueda de información de elemento por código QR

Esta opción está pensada para cuando el usuario que está en terreno y quiere ver información de algún elemento presente, como averiguar que elemento es, alguna de sus especificaciones técnicas o saber si está operando de forma correcta.

Para eso la pantalla muestra lo que ve la cámara del dispositivo móvil, al posicionar ésta sobre el código *QR* físico, pegado al elemento.

La aplicación inmediatamente correlaciona este código *QR* con el elemento correspondiente, desplegándolo automáticamente en pantalla. De esta manera el operador tiene una manera fácil y rápida de obtener información de un elemento en terreno, sin necesidad de conocer el nombre o código único del elemento, y sin perder tiempo ingresando estos datos.

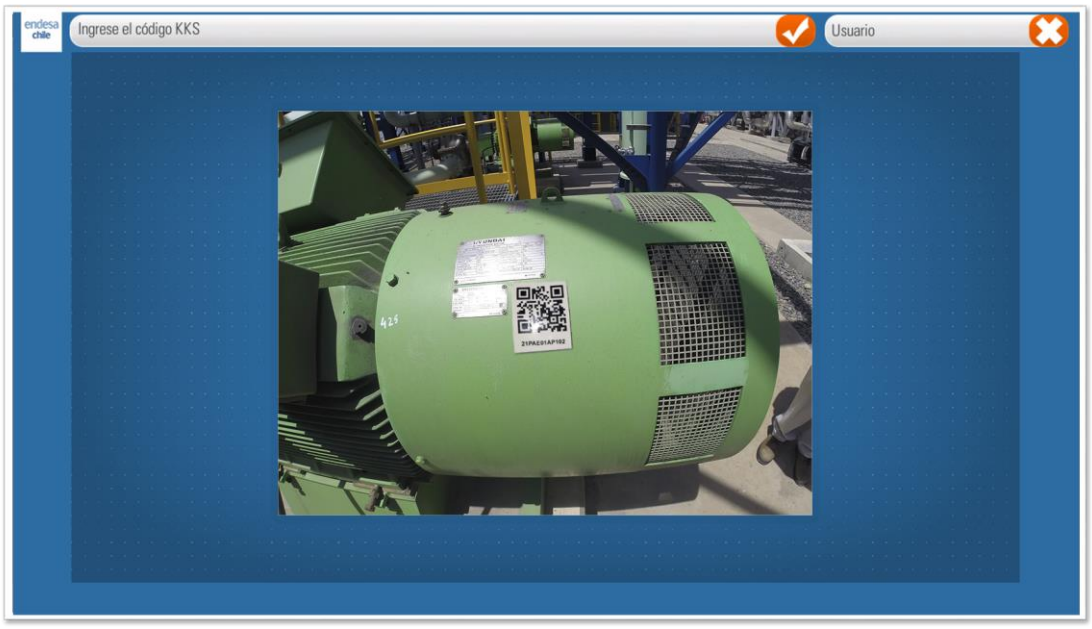

Ilustración B.8 Captura de QR a través de cámara web.

B.4.4 Alarmas

El último botón del menú principal es el de alarmas. Aquí gracias a la información en tiempo real, es posible desplegar las alarmas de elementos que estén fuera de rango normal de operación o en falla. Estas alarmas son proporcionadas por los mismos sistemas de control que utiliza la industria.

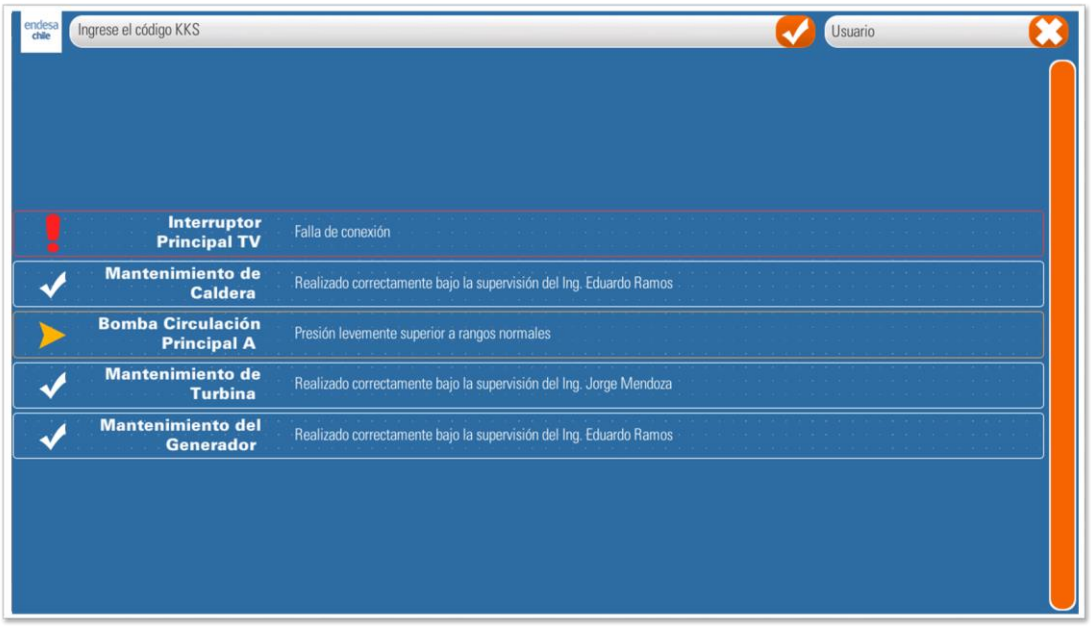

Ilustración B.9 Lista de alarmas.

La alarma se muestra como un botón que contiene el elemento en cuestión y el tipo de falla o alarma. Al presionar el botón o alarma correspondiente, equivale a ingresar el código único del elemento en el buscador, por lo que se va a la vista aérea de la central y aparece el círculo verde señalando la posición del elemento seleccionado. Al igual que al buscar un nombre por código único, se hace la navegación desde la vista aérea general hasta el elemento seleccionado para saber exactamente en qué parte de la planta se ubica.

B.5 Panel Lateral Izquierdo

El panel lateral izquierdo, se muestra al hacer *Press* sobre un elemento, este panel contiene 3 botones: información general del elemento, información en tiempo real del elemento, documentos y planos asociados al elemento.

Ilustración B.10 Despliegue de información de elemento.

B.5.1 Información general del elemento

Este botón despliega un panel lateral al lado derecho, que contiene en la parte superior el nombre del elemento, su código único y el código *QR* del mismo. Debajo de esto se despliega una lista de atributos o características técnicas del elemento en cuestión.

B.5.2 Información en tiempo real

La etiqueta "*On-Line"* es similar a "*Información*", con la diferencia que la lista despliega datos en tiempo real del elemento, datos que son obtenidos en el dispositivo a través de consultas web a una central de monitoreo. Esta central es la que recibe información de sensores alrededor de toda la planta, variables de estado *(run, stop,*

temp high, range over, entre otras) y sensores de medida (temperatura, corriente, presión, etc).

Estos datos se van actualizando periódicamente. Además en la parte inferior se despliega la fecha de última actualización y un botón para actualizar los datos.

B.5.3 Documentos y planos

Aquí se despliega una lista de planos y documentos relacionados con el elemento seleccionado, lo que permite tener información y documentación de fácil acceso en terreno.

Como los documentos y planos están en distintos formatos, entre ellos, *autocad, PDF, Docx, Excel,* etc. se optó por abrir estos archivos en programas externos. Esto principalmente por dos razones, primero para asegurar la mejor forma y capacidad de visualización de los documentos y además para reducir considerablemente los tiempos de desarrollo al tratar de generar visualizadores internos.