



UNIVERSIDADE DA CORUÑA

Facultad de Informática

Departamento de Tecnologías de la Información y las Comunicaciones

Proyecto Fin de Carrera

Superficies multitáctiles interactivas: desarrollo de prototipo, software de control y paradigmas de aplicación a la visualización 3D

Autor	Alberto Jaspe Villanueva
Director	Julián Dorado de la Calle
Tutor	Juan Ramón Rabuñal Dopico

A Coruña, Junio de 2011

Por y para mis padres

Quiero agradecer, antes de nada, a **Julián Dorado** y **Juan Ramon Rabuñal**, director y tutor de este proyecto respectivamente, por brindarme la oportunidad de finalizar por fin mi carrera universitaria, en el momento que más lo necesitaba.

A la gente pasada y presente de **VideaLAB** y **RNASA**, de la que he aprendido mucho estos años, y que han forjado mi carácter como ingeniero y como persona.

A **mi familia**, porque los quiero.

A **mis amigos**, por estar siempre ahí.

Agradezco particularmente y de forma muy especial,

A **mi padre**, que me enseñó a sumar en binario cuando apenas sabía hacerlo en decimal.

A **Carlos** y **Antonio**, porque son mis grandes amigos. A las duras y a las maduras.

A **Taibo**, porque he aprendido de él más que de nadie. Es mi maestro.

A los **Fernández Sotelo**, que tantas veces me han apoyado, alojado y dado de comer.

A **María**, mi confidente y consejera.

A **Rubén**, porque es un crack, y de eso algo se pega.

A **Jalley**, porque su forma de ver la vida me inspira.

A **Gus**, por pegarme el último empujón que necesitaba.

A **Javi**, por invitarme a una copa en cualquier lugar del mundo.

A **Johnnie** y **Salo**, porque son buena gente.

A **Raúl**, **Pau**, **Brais**, **Isa**, **Liliana** y **Fátima**, porque da gusto tener amigos inteligentes.

A **Quique**, por creer en mí, y además demostrarlo.

Y a **Tao** y mi reciente viaje a Japón, que repentinamente ha llenado mi vida de horizontes.

A todos vosotros, gracias.

*“Let the future tell the truth and evaluate each one
according to his work and accomplishments.”*

Nikola Tesla (1856 – 1943)

RESUMEN

El gran auge de las superficies multitáctiles interactivas motiva el desarrollo de este trabajo, que pretende estudiar de abajo a arriba todos los niveles de abstracción, herramientas software y hardware necesarias para conseguir un sistema completamente funcional aplicado a la visualización 3D.

En este proyecto se aborda tanto el diseño como la implementación de un sistema software para el uso de mesas multitáctiles de la forma más eficiente y genérica posible, desde tres capas de abstracción: adquisición de datos mediante técnicas de visión artificial, gestión de dichos datos utilizando el protocolo TUIO u otros protocolos, y su integración en aplicaciones arbitrarias. Además se mostrará el desarrollo de un prototipo hardware funcional y se crearán dos paradigmas nuevos de movimiento para la visualización tridimensional, que quedarán integrados con el motor 3D de terrenos y arquitectura digital SANTI.

PALABRAS CLAVE

Interfaces naturales, superficies interactivas multitáctiles, TUIO, visualización 3D, visión artificial, SANTI

ÍNDICE

1	Introducción	1
1.1	Objetivos del proyecto	2
1.1.1	Desarrollo de software	3
1.1.2	Desarrollo de prototipo	5
1.1.3	Aplicación a la visualización 3D	6
1.2	Organización del proyecto	7
2	Fundamentos básicos.....	9
2.1	Interfaces naturales	9
2.2	Superficies Multitáctiles Interactivas	10
2.2.1	Estructura.....	11
2.2.2	Sistema de coordenadas	11
2.2.3	Toques.....	12
2.2.4	gestos.....	13
2.3	Visión Artificial.....	14
2.4	Comunicaciones: el protocolo TUIO	15
2.5	Paradigmas de movimiento en visualización 3D	16
2.5.1	Visualización 3D.....	16
2.5.2	El SANTI.....	17
2.5.3	Cámaras 3D y movimiento.....	18
3	Estado del Arte.....	20
3.1	Superficies interactivas multitáctiles	21
3.1.1	Microsoft Surface.....	21
3.1.2	Diamond Touch	22
3.1.3	Reactable.....	23
3.1.4	Lemur Input Device	24
3.1.5	Dispositivos móviles: Smartphones y Tablets	25
3.1.6	Dispositivos caseros	26

3.2	Software.....	26
3.2.1	Trackers	27
3.2.2	Gestos.....	28
3.2.3	Frameworks	28
3.3	Visualización 3D y Multitouch.....	29
4	Visión global, metodología y planificación	30
4.1	Metodología	30
4.2	Análisis General.....	32
4.2.1	Captura de requisitos	32
4.2.2	Diagrama de paquetes.....	33
4.2.3	Software y herramientas.....	36
4.2.4	Planificación	38
5	Capa de control.....	40
5.1	Introducción	40
5.2	Librería vlvisiona.....	41
5.2.1	Análisis	41
5.2.2	Diseño	46
5.3	Ejemplo de Aplicación: reconocedor de manos	51
5.3.1	Efectividad de vlvisiona.....	52
5.3.2	Planteamiento	53
5.3.3	El Procesador: Algoritmo de búsqueda de brazos.....	54
5.3.4	El Postproceso: estabilización de los cursores (Filtrado)	59
5.3.5	Calibración	61
5.4	Community Core Vision (CCV)	62
6	Capa de gestión	65
6.1	Introducción	65
6.2	Librería vlmt.....	66
6.2.1	Análisis	66
6.2.2	Diseño	69
6.3	dispositivos soportados	74
6.3.1	TUIO.....	74

6.3.2	Windows 7	79
7	Capa de aplicación - paradigmas de movimiento 3D	86
7.1	Diseño de Estrategias de Vuelo en SANTI	87
7.2	PRotaZoom	91
7.2.1	Punto de control.....	94
7.2.2	Un toque: arrastre	95
7.2.3	Dos toques: PRotaZoom.....	98
7.2.4	Inercia	101
7.2.5	Extensiones de tres toques.....	102
7.3	Touch-Walk	103
7.3.1	Punto de vista y movimiento	105
7.3.2	colisiones	106
7.3.3	Teletransportadores	108
8	Desarrollo del prototipo hardware	110
8.1	Fundamentos	110
8.1.1	Fuentes de luz infrarroja	110
8.1.2	Cámaras	113
8.1.3	Reflexión Interna Total Frustrada (FTIR)	114
8.2	Diseño y construcción	116
8.2.1	Concepto.....	116
8.2.2	Campos visuales: proyección y captura	117
8.2.3	Luz Infrarroja	120
8.2.4	Superficie	121
8.3	Materiales y costes	125
8.4	Integración y Resultado.....	126
9	Casos de estudio.....	128
9.1	La Fuente Encantada	128
9.2	Ayuntamiento de La Coruña: presentaciones PIM y FEIL 2010	131
9.3	Visualización 3d del nuevo edificio CITIC (UDC)	133
9.4	Visualización 3D Interactiva de la Catedral de Santiago de Compostela	134
10	Conclusiones y Líneas futuras	136

10.1	Conclusiones	136
10.2	Líneas futuras.....	138
10.2.1	Mejoras en la Capa de Control	139
10.2.2	Mejoras en la Capa de Gestión	140
10.2.3	Paradigmas de Movimiento 3D	140
Apéndice I.	Ficheros de Configuración.....	142
i.	Librería vlvisiona.....	142
ii.	Librería vlmt.....	144
iii.	Estrategias de vuelo.....	145
Bibliografía		149

1 INTRODUCCIÓN

Una de las grandes cuestiones pendientes en la historia de la computación es determinar cuál es la mejor manera que tenemos los humanos de interactuar con los sistemas informáticos. No en vano este campo tiene un nombre propio, denominado *CHI, Computer Human Interaction*, o **Interacción Hombre-Máquina** (ACM, 1996). Desde luego, la respuesta no es única, y en las pasadas décadas se han ideado decenas de dispositivos que tratan de conseguir una interacción cómoda, fluida y funcional. En ocasiones estos dispositivos se han orientado a un uso genérico del sistema que tratan de manejar, como los conocidos teclados y ratones, aunque para aplicaciones específicas, como simuladores, centros de control o telemando, diseño asistido (CAD) etc. se han creado interfaces muy específicos como joysticks, paneles de control o tabletas gráficas.

Los denominados **dispositivos táctiles** han tenido un gran auge en los últimos años, y desde luego toda previsión apunta hacia un agudo crecimiento de su uso. Esto es probablemente debido a varios factores, aunque fundamentalmente a las capacidades de integración con pantallas, que resulta en unos interfaces muy sencillos de usar (de tipo “*toque aquí*”). Además esta integración resuelve las necesidades de entrada – salida de todo tipo máquinas (como móviles, cajeros, puntos de información, etc.) de una forma tremendamente compacta y vistosa.

El concepto de superficie multitáctil interactiva o mesa interactiva, aúna los conceptos mencionados en un paradigma de interacción basado en la metáfora del escritorio vir-

tual. Esto es, una mesa de trabajo llena de contenidos que bien pueden ser documentos, contenidos multimedia (como fotografías o vídeos) o cualquier tipo de aplicación, en la que el usuario interactúa con sus manos de la manera más natural posible.

Como se verá en el siguiente capítulo, existen hoy en día varios prototipos de laboratorio de superficies interactivas, e incluso algunas de ellas comerciales. El problema reside en la falta de integración entre los dispositivos existentes en el mercado, que solucionan sólo partes aisladas del problema, o de aquellos de tipo “casero” de los cuales abunda información en la red, pero de forma difusa e incompleta. Apenas existen soluciones ad-hoc para lograr tener una solución integral, una mesa interactiva funcional y adaptable a las necesidades de cualquier aplicación, con la fiabilidad y el alto rendimiento que éstas suelen precisar en un entorno real de explotación.

El presente proyecto pretende abarcar estas cuestiones desde un punto de vista de ingeniería global, que abarque el análisis del problema, el desarrollo de software y la construcción de un prototipo funcional que pueda ser llevado situaciones reales de producción. A mayores, se hará una pequeña incursión en algunos nuevos paradigmas de movimiento en espacios tridimensionales, con formas de interacción derivados de las superficies multitáctiles.

1.1 OBJETIVOS DEL PROYECTO

Este proyecto analizará los métodos más viables para construir o integrar los componentes necesarios en las mesas multitáctiles, desarrollar las librerías de software necesarias para su control de una forma flexible y adaptable a cualquier tecnología, y apro-

ximar unas metáforas de interacción para su uso en aplicaciones arbitrarias orientadas a la visualización 3D.

En cada una de dichas fases, se buscan unos objetivos concretos, que se pasan a describir en las subsecciones siguientes.

1.1.1 DESARROLLO DE SOFTWARE

En el siguiente diagrama podemos apreciar que el software necesario para llevar a cabo este proyecto se ha dividido en tres capas de abstracción.

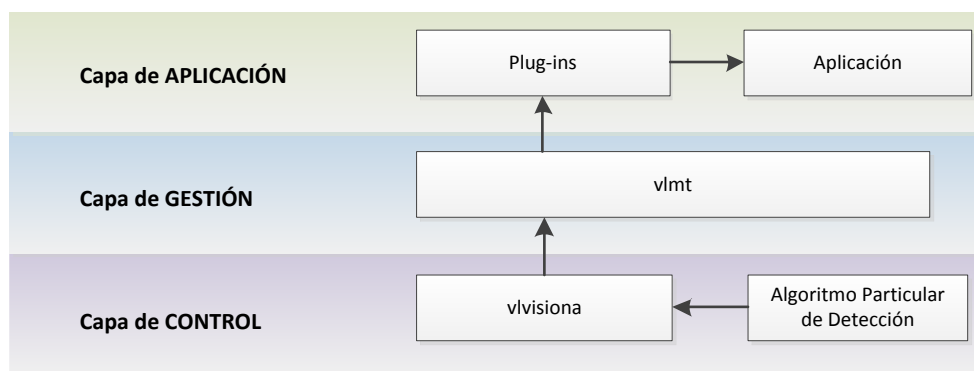


Ilustración 1. Capas de Abstracción.

La capa de más bajo nivel, denominada **capa de control**, será la encargada de suministrar, mediante técnicas de visión artificial (u otras), los datos recogidos directamente del hardware. Para ello, se provee de una librería genérica de algoritmos de visión artificial, **visiona**, que a su vez es un servidor del protocolo TUIO.

La capa de nivel intermedio, denominada **capa de gestión**, será un proxy que permita gestionar cualquier tipo de dispositivo multitouch como si se tratara de una caja negra,

homogeneizando los datos crudos que provengan de superficies interactivas arbitrarias. En esta capa se provee la librería de control ***vImt***, así como la implementación de dos tipos genéricos de dispositivos, los basados en protocolo TUIO, como clientes de dicho protocolo, y los basados en el soporte multitouch de Windows 7.

El último nivel, la ***capa de aplicación***, demuestra un uso práctico de la gestión de cursores sobre una superficie interactiva mediante la aplicación a un software de navegación 3D sobre un modelo digital de terreno y modelos arquitectónicos.

Todas estas capas se explicarán con detalle en los próximos capítulos, pero en conjunto podemos decir que cualquiera de ellas presenta unos requisitos especiales a la hora de diseñar y desarrollar el software. Haremos especial énfasis en los siguientes:

- **Flexibilidad:** El software debe ser diseñado con el mayor grado de flexibilidad posible para permitir añadir nuevas tecnologías de análisis de imagen o dispositivos multitouch en el futuro.
- **Eficiencia:** como veremos, quizá el factor más importante en un software con este fin sea la latencia, esto es, la diferencia de tiempo entre que se produce físicamente el toque hasta que la aplicación final lo interpreta. Se hará especial hincapié en este aspecto, buscando un exhaustivo análisis de las posibilidades *multithread* que ofrecen las distintas plataformas soportadas.
- **Simplicidad:** se tratará de mantener el software lo más simple posible, dada la repercusión que este factor tiene tanto en el rendimiento como en su futuro mantenimiento. Esto se traducirá en un interfaz basado en el menor número de

funciones posible, y conseguirá una rápida integración en cualquier software externo.

- **Modularidad:** se buscará trabajar de forma modular, en este caso separando claramente las librerías de las distintas capas.
- **Multiplataforma:** se impone como requisito que sea funcional en varias plataformas como Windows y Linux.
- **Robustez:** dado que el objeto principal del sistema es funcionar en entornos no técnicos, como exposiciones, presentaciones, etc. el software debe tener la mayor tolerancia a fallos posible.

1.1.2 DESARROLLO DE PROTOTIPO

Se incluye dentro de este proyecto la construcción y puesta en marcha de un prototipo de superficie multitouch, utilizando la técnica de FTIR y una retroproyección, utilizando el software de control mencionado anteriormente.

En esta parte concreta se marcan como objetivos las siguientes características:

- **Fiabilidad:** debe ser perfectamente funcional, y de hecho, será base para un producto comercial que se integrará en entornos reales de explotación.
- **Costes:** las tecnologías utilizadas en la construcción son de bajo coste, y así tratarán de mantenerse a lo largo de todo el desarrollo.

1.1.3 APLICACIÓN A LA VISUALIZACIÓN 3D

En la tercera de las capas mencionadas anteriormente, la “capa de aplicación”, se incluye un uso del sistema a las tecnologías visualización 3D. Para ello, en conjunción con el software de navegación sobre terrenos interactivos **SANTI** del laboratorio VideLAB de la Universidad de La Coruña, se generarán dos *plugins* de movimiento con nuevos paradigmas de exploración tridimensional, que denominamos ***Touch-Walk*** y ***PRotaZoom***.

En esta última etapa se buscarán dos características en particular, añadidas a las ya mencionadas sobre la creación del software:

- **Investigación:** estos nuevos paradigmas de movimiento serán base de investigación a las posibilidades de estos nuevos dispositivos, dentro del campo de los Interfaces Naturales, aplicadas a la visualización para ingeniería y arquitectura.
- **Usabilidad:** los modos de vuelo serán parte de un producto comercial, que se utilizará en entornos reales, por usuarios no entrenados. Esto implica un trabajo importante en cuanto a la usabilidad, y la gestión de movimientos no deseados.

Este trabajo culminará con la adaptación de las tecnologías analizadas y el software desarrollado a un producto comercial, que será explicado en los últimos capítulos de la memoria.

1.2 ORGANIZACIÓN DEL PROYECTO

Este proyecto plantea distintas capas de abstracción, conectadas por un protocolo común, así como la elaboración de un prototipo hardware. Como cada una de estas capas tiene sus propios fundamentos tecnológicos y etapas metodológicas, se ha optado por una división en capítulos donde inicialmente se exponen algunos argumentos para el desarrollo de este trabajo (cap. 1-3), después se profundiza el núcleo del mismo, la parte software (cap. 4-7), se comenta la construcción del prototipo (cap. 8) y finalmente se concluye exponiendo algunos casos reales de uso y discutiendo líneas futuras (cap.9).

De esta forma, se ha llegado a la siguiente distribución por capítulos:

1. **Introducción.** Se introduce el proyecto, comentando las motivaciones, objetivos y estructura del mismo.
2. **Fundamentos básicos.** Se exponen en este capítulo algunos conceptos básicos del área del dominio que se trabajará en los subsiguientes.
3. **Estado del arte.** Se analizan las soluciones, técnicas y productos existentes en cada uno de los niveles de abstracción.
4. **Visión global del sistema y metodología.** Este capítulo empasta los subsiguientes, dando una visión general y comentando la metodología global que se seguirá.
5. **Capa de control.** Detalla la capa de más bajo nivel, de adquisición de datos, según la metodología indicada en el capítulo tercero.

6. **Capa de gestión.** Detalla la capa de nivel intermedio, donde se manejan e interpretan los datos adquiridos en la capa anterior, según la metodología indicada en el capítulo tercero.
7. **Capa de aplicación y Paradigmas de movimiento 3D.** Detalla la capa de nivel superior, donde se utilizan las anteriores capas para crear unos nuevos paradigmas de movimiento en la visualización 3D, según la metodología indicada en el capítulo tercero.
8. **Desarrollo del prototipo hardware.** Este capítulo describe los fundamentos y pasos en la construcción de un prototipo funcional hardware.
9. **Casos de estudio.** Se plantean casos reales de explotación del sistema descrito en entornos productivos diversos, y se describe la configuración de capas de cada uno de ellos.
10. **Conclusiones y líneas futuras.** Se concluye planteando aquí algunos casos de estudio reales del sistema completo desarrollado, los resultados del sistema y las posibles mejoras.

2 FUNDAMENTOS BÁSICOS

Conseguir crear un sistema completo para hacer funcionar cualquier superficie multi-touch en aplicaciones arbitrarias, y particularmente aplicarlo a la navegación 3D requiere abordar multitud de campos de índole muy diferente. Este carácter multidisciplinar de este proyecto motiva la existencia de este capítulo, donde se introducen de manera muy somera conceptos básicos de todos estos ámbitos de computación.

2.1 INTERFACES NATURALES

En computación, el término Interfaz Natural (*Natural User Interface* o **NUI**) es una forma común usada por diseñadores y desarrolladores de interfaces de usuario para referirse a aquellos que son o se vuelven “invisibles” a los usuarios. La palabra “natural” se utiliza como contrapunto a los habituales dispositivos artificiales que las personas deben aprender a utilizar para interactuar con computadoras.

La potencia de los NUI recae en la rápida conversión de usuarios novatos en expertos. Mientras que los interfaces habituales tienen necesariamente un proceso de entrenamiento más o menos largo, éstos están diseñados para tratar de ofrecer desde un primer momento experiencias instantáneas y exitosas.

Los NUI habitualmente se componen habitualmente de un parte software y una parte hardware, que deben concordar para conseguir el efecto deseado.

2.2 SUPERFICIES MULTITÁCTILES INTERACTIVAS

Definamos el término de Superficie Multitáctil Interactiva aunando significados individuales. Como **superficie**, ésta debe ser tal que el o los usuarios puedan usar partes de su cuerpo, normalmente las manos, para manipularla (no necesariamente tocándola). Además su forma no está sujeta a ningún canon (podría no ser rectangular) ni tampoco plana (podría ser esférica, por ejemplo). El término **multitáctil** se refiere a que debe tener la habilidad de reconocer la presencia de dos o más puntos de contacto. Por último, como **interactiva** debe mostrar imágenes en su interior que reaccionen instantáneamente (es decir, en tiempo real) a las acciones del usuario.



Ilustración 2. Superficies multitouch.

Como se explica en el capítulo dedicado al estado del arte, hoy en día existen una buena variedad de ejemplos, siendo de uso común en dispositivos móviles como teléfonos o *tablets*. Todos ellos se pueden generalizar en una estructura básica, que se comenta a continuación.

2.2.1 ESTRUCTURA

Podemos dividir cualquiera de estas superficies en dos sistemas fundamentales: el encargado de mostrar imagen, ya sea convencional, holográfica o tridimensional, y el dedicado a reconocer los puntos de contacto, que en adelante denominaremos toques. El presente trabajo se centra en estudiar las capacidades del segundo de ellos, si bien en el prototipo hardware se implementa además una solución para el primero.

2.2.2 SISTEMA DE COORDENADAS

Cualquier sistema multitáctil se basa en un marco de referencia, con unos ejes de coordenadas que normalmente serán dos, por ser superficie. Éstas pueden ser coordenadas cartesianas, en los casos planos y/o rectangulares, o polares en aquellos casos curvos.

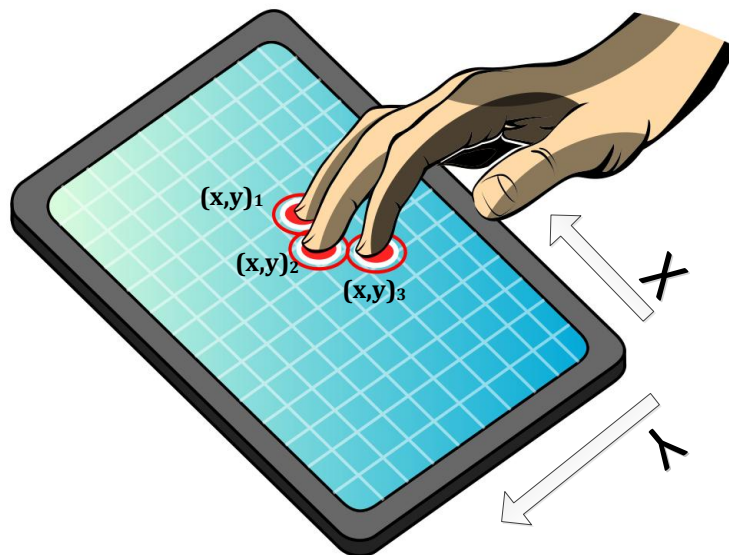


Ilustración 3. Sistema habitual de coordenadas cartesiano.

Un sistema típico sería el mostrado en la Ilustración 3, donde se aprecia que sobre unos ejes cartesianos con origen en una de las esquinas de la superficie, y coordenadas puntuales de los diferentes toques.

2.2.3 TOQUES

Cualquier punto de interacción con la superficie se define como toque. Un toque puede ser “discretizado” como un punto (como el ejemplo anterior), aunque en algunas ocasiones es interesante conocer la forma exacta del toque, representada como blob. Un **blob** es una definición geométrica aproximativa de una forma arbitraria, que normalmente se representa como la elipse que mejor la aproxima, usando los ejes principales y un ángulo de rotación. Esto nos permite además considerar el “tamaño” o área del toque, que en muchos casos, dependiendo del montaje hardware usado, representa la presión del toque.

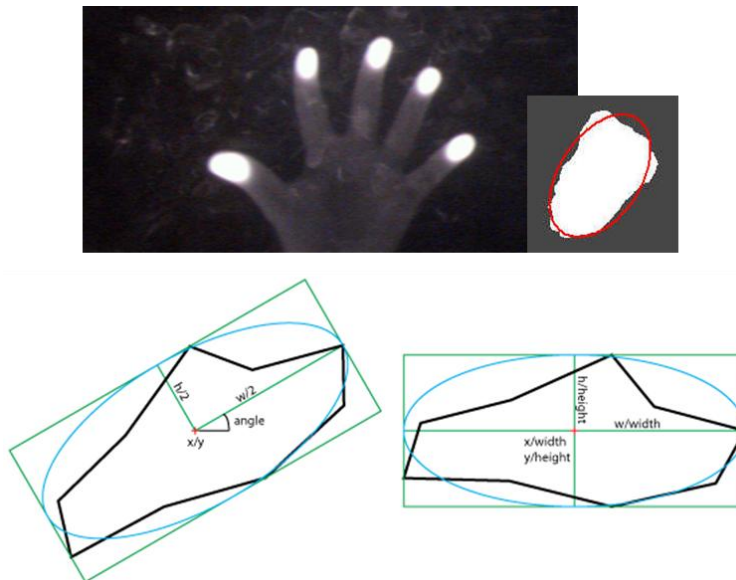


Ilustración 4. Toques como blobs, y representación elipsoidal.

2.2.4 GESTOS

Denominamos en este ámbito “gesto” a una interpretación matemática de la interrelación espacial de uno o varios toques, ya bien sea de modo estático o teniendo en cuenta su variación en el tiempo. Ejemplos de esto serían posicionar tres toques como un triángulo equilátero, o alejar dos toques entre sí a lo largo de la línea recta que los une.

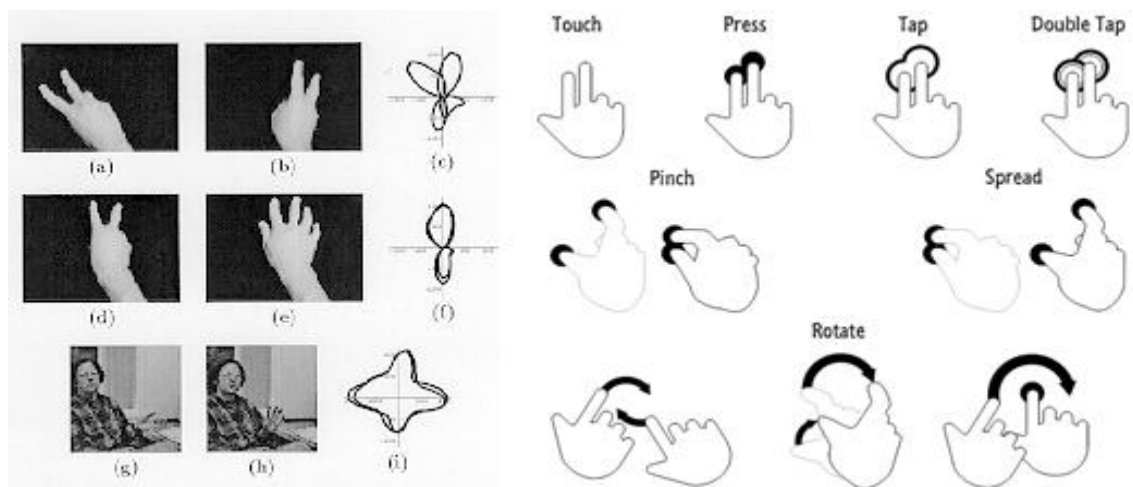


Ilustración 5. Ejemplos de reconocimiento de gestos.

Puede pensarse sobre el reconocimiento de gestos como una forma que tienen los ordenadores de comprender algorítmicamente el lenguaje humano. Esta técnica potencia enormemente estos dispositivos y justifica su pertenencia al campo anteriormente descrito de los interfaces naturales (Wu & Huang, 1999).

En el capítulo dedicado a la capa de aplicación se verá cómo se han capturado los gestos para los paradigmas de movimiento desarrollados.

2.3 VISIÓN ARTIFICIAL

La Visión por Computador o Visión Artificial (*Computer Vision*) es un subcampo de la inteligencia artificial, cuyo propósito es programar un computador para que “entienda” una escena o las características de la imagen (Davies, 2005).

En el campo de superficies multitáctiles, se utilizan, entre otros, diversos medios ópticos para lograr la interacción con el usuario. Normalmente esto es un proceso en tiempo real, que va adquiriendo información de un dispositivo óptico (como una cámara) y procesándolo con técnicas de Visión Artificial. Las más utilizadas son:

- **Segmentación**, normalmente para eliminar las zonas que no interesen y evitar ruido.
- **Reconocimiento de objetos**, para localizar aquellas partes (como los dedos) y su ubicación respecto a la superficie.
- **Tracking**, que significa seguir los objetos detectados a lo largo de una secuencia de imágenes, es decir, en el tiempo.
- **Estimación de posturas**, para el reconocimiento de gestos.
- **Análisis de patrones**, para reconocimiento de objetos marcados.

Como se vio en la sección anterior, el seguimiento de las estructuras denominadas blobs es, probablemente, la técnica más apreciada para el desarrollo de software que maneje dispositivos multitáctiles. Cabe destacar los trabajos de T. Lindeberg en este campo (Bretzner & Lindeberg, 1998) (Lindeberg, 1993), que centra sus estudios en la detección de características y la transformación de espacios, conceptos de suma relevancia en el desarrollo de este proyecto.

2.4 COMUNICACIONES: EL PROTOCOLO TUIO

TUIO (Kaltenbrunner, Bovermann, Bencina, & Costanza, 2005) es un *framework* abierto que define un protocolo y API para superficies multitáctiles. Permite la transmisión de una descripción abstracta de superficies interactivas, incluyendo eventos de toque y estados de objetos tangibles. Este protocolo está diseñado para trabajar con la estructura clásica **cliente – servidor**, es decir, codifica los datos de control de una aplicación servidora (un *tracker*, como una aplicación de visión artificial) y los envía a cualquier cliente capaz de decodificarlos. Existe un gran número (en aumento) de aplicaciones de tracking y de clientes que lo soportan, así como varias librerías de control para distintos entornos y lenguajes de programación.

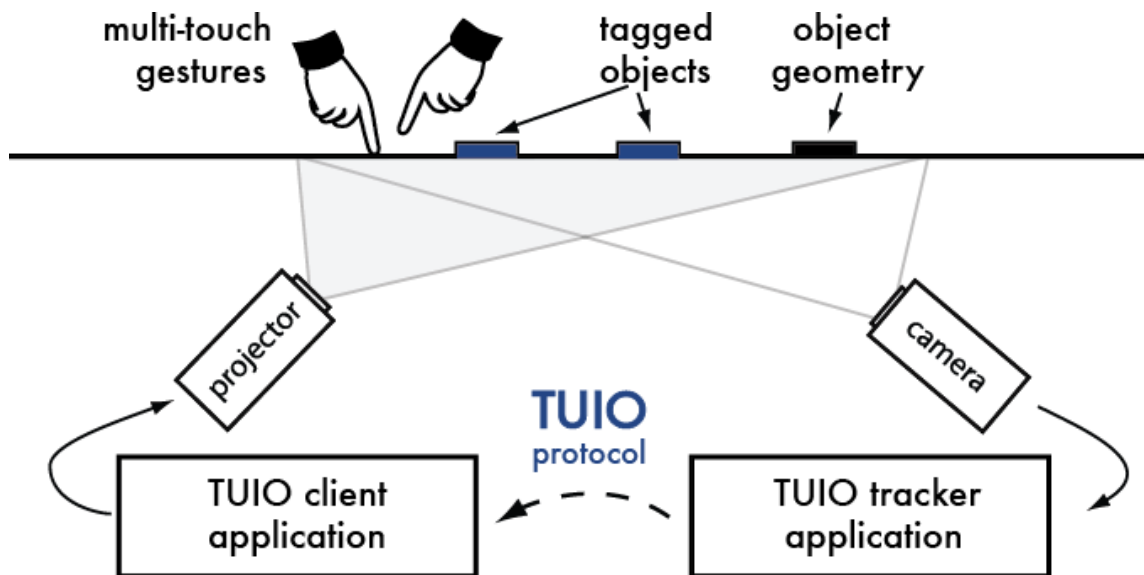


Ilustración 6. Estructura cliente - servidor del protocolo TUIO.

TUIO nace como una adaptación de otro protocolo, llamado ***Open Sound Control*** (OSC) (Wright, Freed, & Momeni, 2003), un estándar emergente para entornos creativos de control de instrumentos musicales. La definición de este protocolo era fácilmente adaptable al control de superficies interactivas, y en el año 2005 se consolida en la especificación 1.0, y coge gran auge entre otros motivos, gracias a su uso por parte del proyecto **Reactable** (ver capítulo de estado del arte). Poco después aparecerá la especificación 1.1, con algunas correcciones y añadidos que se verán más adelante, y recientemente (finales del 2010) aparece la especificación 2.0.

2.5 PARADIGMAS DE MOVIMIENTO EN VISUALIZACIÓN 3D

2.5.1 VISUALIZACIÓN 3D

Hoy en día a nadie le es ajeno este concepto. Películas de animación, videojuegos, visualizadores del globo terráqueo, y un sinnúmero de aplicaciones han irrumpido en nuestra vida cotidiana hace años y en estos tiempos forman parte habitual de nuestro repertorio de posibilidades.

Las aplicaciones 3D interactivas necesitan un motor gráfico software (normalmente asociado a un hardware, hoy por hoy las tarjetas de vídeo de consumo) que gestione todo tipo de tareas, según el objetivo para el que fue creado. Desde la carga de modelos geométricos, efectos especiales, niveles de detalle, texturas, físicas, interacción, y un largo etcétera. Por ejemplo, un juego de coches debe tener un buen sistema de simulación física, y la interacción debe estar preparada para responder a las más leves entradas del usuario, sea desde un teclado o desde un volante. Sin embargo, un software de

CAD (*Computer Aided Design*, o Diseño Asistido por Ordenador) debería estar centrado en la precisión y la formalidad de la nomenclatura.

2.5.2 EL SANTI

SANTI responde al acrónimo **Sistema Avanzado de Navegación sobre Terrenos Interactivo**, y ha sido uno de los principales proyectos del laboratorio VideALAB, desde el año 1999. Se trata de un visualizador 3D interactivo de territorio, que utiliza técnicas avanzadas del campo de los gráficos por computador para cargar y *renderizar* en tiempo real cualquier extensión de terreno, mediante el uso de niveles de detalle (*Levels of Detail* o **LODs**) (Seoane, Taibo, Hernández, & Jaspe, 2007).

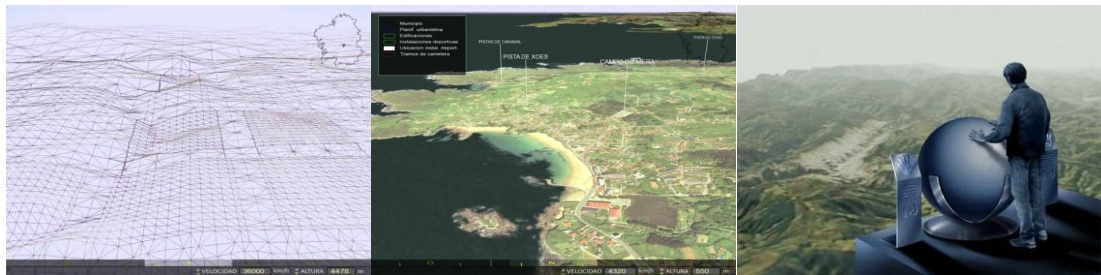


Ilustración 7. Imágenes de SANTI y distintos dispositivos de uso.

En los últimos años se han ampliado enormemente las capacidades de SANTI, y entre otras mejoras (como integración de datos GIS, Bases de Datos, texturas dinámicas actualizables, WMS, etc.) actualmente permite integrar modelos arquitectónicos complejos en sus propias ubicaciones de terreno.

Éste ha sido el sistema escogido en este proyecto para realizar la implementación de los nuevos paradigmas de movimiento, por ser el autor también desarrollador de este motor y tener a su disposición su código fuente. Hasta el momento, SANTI permitía el uso de distintos dispositivos para su manejo, desde el clásico teclado y ratón hasta joysticks, sistemas de *tracking*, *wiimote*, etc. Con la desarrollo de este trabajo, incorpora todo un sistema funcional multitouch e integra dos nuevas formas de moverse por el espacio, los paradigmas de vuelo.

Existe una buena colección de artículos, libros, proyectos y tesis que hacen referencia a este motor, además de una gran variedad de proyectos comerciales en los que se ha utilizado. El lector podrá encontrar más información e incluso algunos de esos trabajos en la página web <http://videalab.udc.es/santi>.

2.5.3 CÁMARAS 3D Y MOVIMIENTO

El motor SANTI permite implementar módulos de movimiento, con distintos dispositivos y paradigmas. Profundizaremos en este aspecto en el capítulo 7. En cualquier caso, la forma de definir una cámara en este motor es bastante similar a la de cualquier otro, y se basa fundamentalmente en la definición de dos vectores de tres elementos:

- **Posición**, vector (x,y,z) , con las coordenadas del punto en el espacio de referencia que utilicemos.
- **Dirección**, vector (h,p,r) . Hay múltiples maneras de definir la dirección, como ángulos relativos, quaternions, etc. En este caso se utilizan los denominados ángulos de navegación, o **ángulos de Euler**. Éstos definen una orientación como **Heading** o **Yaw**(cabeceo), **Pitch** (alabeo) y **Roll** (guiñada). Si bien es cierto que

matemáticamente no son demasiado eficientes a la hora de operar con ellos, son muy intuitivos para el usuario o el desarrollador, que pueden imitarlos con simples movimientos de cabeza. En la Ilustración 8 se puede ver su funcionamiento, aplicando el símil de un avión.

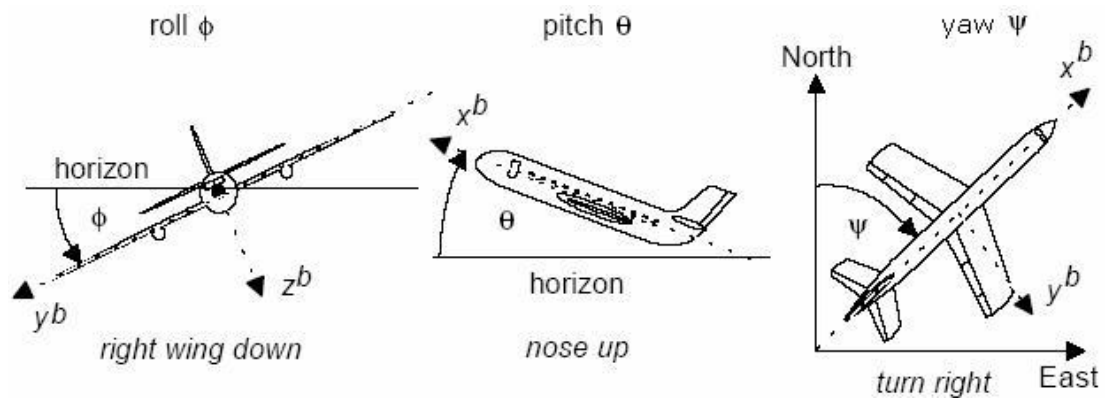


Ilustración 8. Ángulos de Euler.

La actualización de ambos vectores a lo largo del tiempo, según la entrada del usuario es lo que denominamos de forma muy genérica paradigma de movimiento, y es el *leitmotiv* último de este proyecto.

3 ESTADO DEL ARTE

El leitmotiv de este proyecto es generar un conjunto de librerías software para controlar cualquier dispositivo de tipo multitáctil y poderlo adaptar a aplicaciones arbitrarias, particularmente a visualizadores 3D.

Los sistemas comerciales multitáctiles normalmente son paquetes cerrados, con características demasiado enfocadas a objetivos concretos. Esto obliga a utilizar el software propietario de la firma, o a adaptar el nuestro a sus estándares de trabajo (programar en un lenguaje determinado, o contra un SDK determinado, a veces con licencias abusivas).

Por otra parte, empiezan a existir algunas librerías software, desarrolladas normalmente por la comunidad de software libre, pero que solamente solucionan una pequeña parte del problema.

El presente capítulo analizará primero los sistemas completos más importantes hasta el momento, y seguidamente las librerías software existentes para llevar a cabo tareas específicas, que podrían ser usadas en el proyecto. Por último, se expondrán algunos ejemplos de paradigmas de movimiento 3D en estos dispositivos.

3.1 SUPERFICIES INTERACTIVAS MULTITÁCTILES

La información acerca de estos dispositivos se encuentra actualmente muy dispersa. Bill Buxton, un prestigioso investigador del equipo de *Microsoft Research*, mantiene un listado al día de más de 30 sistemas multitouch que ha ido probando desde 1972. Para el lector interesado en esta cuestión, es una web de obligada lectura (Buxton, 2011).

3.1.1 MICROSOFT SURFACE

La **Microsoft Surface** es un producto de la empresa **Microsoft**, que permite al usuario manejar contenidos digitales con movimientos de las manos u objetos. Fue anunciada a mediados del 2007 y lanzada a finales de este mismo año, dirigido a hoteles, tiendas, restaurantes y lugares de entretenimiento.

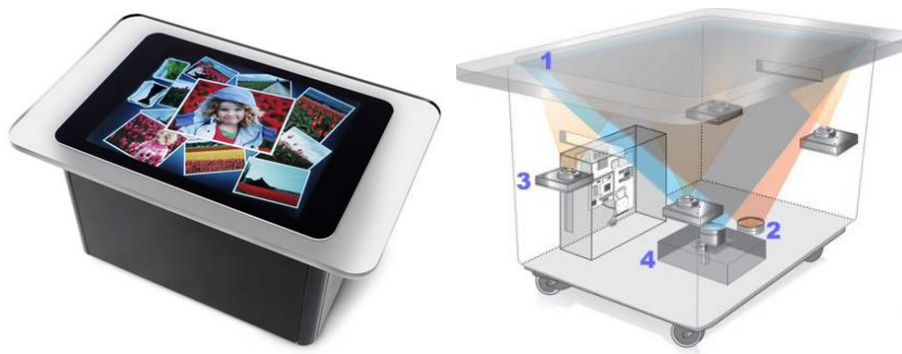


Ilustración 9. Microsoft Surface.

Este dispositivo utiliza exactamente la misma solución tecnológica que el prototipo desarrollado en este proyecto, es decir, el fenómeno óptico denominado FTIR, que se describe en detalle en el capítulo 8.

Su funcionamiento interno forma parte de un paquete software cerrado. Los desarrolladores tienen la posibilidad de crear aplicaciones para ella mediante Windows Presentation Foundation (aplicación de quasi-alto nivel tipo *WYSIWYG*) o XNA (SDK para el desarrollo de aplicaciones gráficas que utiliza DirectX por debajo).

Microsoft mantiene actualmente un equipo de investigación llamado ***Tabletop Research Display*** dedicado únicamente a trabajar sobre tecnología multitouch en sus sistemas operativos (Microsoft).

3.1.2 *DIAMOND TOUCH*

La mesa interactiva ***Diamond Touch*** es un producto de la empresa ***Circle Twelve Inc.***, muy similar y anterior a la Microsoft Surface, presentada en el *ACM Symposium on User Interface Software and Technology* en 2001. A pesar de correr también con el sistema Windows, está quizá más orientada a aplicaciones de trabajo colaborativo. En los últimos años le han dado un gran empuje con el desarrollo de *plugins* para sistemas CAD y GIS.



Ilustración 10. Diamond Touch.

En este caso, la solución para conseguir el multitouch no se basa en métodos ópticos, optimizando el espacio requerido a únicamente la pantalla de tipo capacitiva.

Con la mesa se provee a los desarrolladores de un SDK mucho más abierto, que permite construir aplicaciones siguiendo estándares como C, C++, Java, C# y Flash.

3.1.3 REACTABLE

La mesa Reactable merece una mención especial, por haber nacido de un proyecto de investigación y en una universidad española, igual que este. Se trata de un instrumento musical electrónico colaborativo, manejado con un interfaz tangible basada en una mesa multitáctil, e inspirada en los sintetizadores de los años setenta. Fue desarrollada por el **Grupo de Tecnología Musical** de la **Universidad Pompeu Fabra** de Barcelona (Universitat Pompeu Fabra, 2010), y ganó en fama cuando la músico islandesa **Björk** lo utilizó en su gira *Volta*, en el 2007.



Ilustración 11. ReactTable.

El software utilizado se denomina **ReacTIVision** es un referente en la comunidad por ser de código abierto (excepto la parte de audio) y está basado en los mismos princi-

pios e incluso en algunas de las librerías usadas en este proyecto. Cabe destacar los grandes aportes de este grupo a la comunidad, tanto en participación activa en foros de discusión especializados, como en artículos de investigación a nivel internacional.

Actualmente acaban de presentar una versión para dispositivos móviles basada en el sistema operativo *Android* De *Google*.

3.1.4 LEMUR INPUT DEVICE

Este dispositivo, desarrollado por la compañía francesa **JazzMutant**, es un controlador MIDI musical que maneja todo tipo de aparataje electrónico, como pueden ser sintetizadores, mesas de control, luces, o sistemas de domótica.



Ilustración 12. Lemur Input Device.

Es interesante mencionarlo por dos motivos. Por una parte, es “el abuelo” del protocolo en el que se basa este proyecto, TUIO, al popularizar *Open Sound Control*. Por otra parte, es uno de los primeros dispositivos multitáctiles portables, concepto que a día de hoy está ampliamente extendido con los dispositivos móviles.

3.1.5 DISPOSITIVOS MÓVILES: SMARTPHONES Y TABLETS

Hablar hoy por hoy de dispositivos móviles ya no es hablar del futuro, si no del presente. En la misma semana en la que se escriben estas líneas, Google ha desvelado que recibe ya más visitas desde dispositivos móviles que desde ordenadores con operativos tipo Unix.

El uso de Smartphones y Tablets está tan extendido en el mundo occidental y asiático que éstos empiezan a ser sustitutivos de los PCs. Si bien es cierto que, tal y como se definió el concepto de multitouch (3 o más toques simultáneos) no entrarían dentro de esta denominación, la realidad es que la mayoría de ellos admite dos toques. El mercado se ha llenado de aplicaciones que aprovechan esta capacidad, y gracias a esto, el usuario medio cada vez está más entrenado para utilizar dispositivos multitáctiles sin necesidad de indicación alguna.



Ilustración 13. Smartphones y Tablets, considerados multitouch.

Sería injusto negar que el gran artífice de esto ha sido la empresa **Apple**. La introducción de las tecnologías de toque en toda su gama de dispositivos (*iPhone*, *iPad*, *iPod*, etc.) es, fue el empujón que necesitaban para popularizarse. Desgraciadamente, la em-

presa de *Steve Jobs*, mantiene decenas de patentes cerradas que incluyen técnicas, hardware, e incluso gestos, dificultando enormemente cualquier desarrollo comercial que se quiera hacer de estos productos.

3.1.6 DISPOSITIVOS CASEROS

No se podría cerrar esta sección sin hacer alusión alguna a **Jefferson Y. Han**, investigador de la Universidad de Nueva York, y uno de los desarrolladores más importantes y pioneros en el campo del multitouch (Han J. , 2010).

Lo creativo de sus proyectos, formal de sus artículos de investigación, e inspirador de sus conferencias, han alentado el trabajo de muchos a la hora de desarrollar sus propios dispositivos, incluido el autor de este proyecto.

Existen decenas de proyectos denominados **DIY** (*Do It Yourself*) para construir superficies interactivas de todo tipo. La comunidad provee de un amplio abanico de webs, foros, listas de correo, librerías software opensource, etc. Nombrar una de ellas obligaría tener que descartar muchas otras, así que se insta al lector a realizar una simple búsqueda para descubrirlo.

3.2 SOFTWARE

Existen algunas librerías y paquetes de desarrollo que pueden cubrir algunos de los aspectos abordados en este proyecto. A continuación se describen las más relevantes, que podemos dividir en **trackers**, aquellos que adquieren y envían la información cap-

tada de los dispositivos físicos; de **gestos**; **frameworks**, donde los desarrolladores pueden crear aplicaciones completas y **miscelánea**, para las demás.

La estructura modular por capas de este proyecto permite que cualquiera de las siguientes librerías y aplicaciones que se van a mencionar puedan sustituir o apoyar los módulos desarrollados.

3.2.1 TRACKERS

En este caso, los *trackers* sustituirían a *vlvisiona* en la capa de control.

- **Community Core Vision:** desarrollada *por NUI Group*. Es una aplicación de Visión Artificial para *trackear* blobs de mesas interactivas. En siguientes capítulos se hablará más a fondo, ya que este proyecto la usa.
- **Touchlib:** quizá de las más conocidas, es una librería en C++ que utiliza OpenCV por debajo para el mismo propósito que la anterior. Su ventaja es que es modificable, si se requiere.
- **reacTIVision:** como ya se ha comentado, software del proyecto ReacTable. más orientada al reconocimiento de etiquetas de patrones, aunque con un soporte básico de toques.
- **xTouch:** aplicación para tracking de blobs, para el operativo Mac OSX.
- **OpenNI2TUIO:** tracker de esqueleto basado en la librería OpenNI, que ha alcanzado recientemente mucha popularidad por su soporte de Microsoft Kinect.
- **TUIOdroid:** software de tracking para dispositivos móviles con Android.

3.2.2 GESTOS

La detección de gestos es un problema lo suficientemente complejo para que tenga su propia sección. En el caso de este proyecto, podría situarse entre la capa de gestión y de aplicación, dependiendo del diseño que se planifique.

- **MIRIA:** librería de detección de gestos para utilizar en el framework de Microsoft Silverlight.
- **tuioZones:** gestión de gestos para el framework Processing.
- **Grafiti:** en este caso, programado en C#
- **SGDriver:** es un driver para Windows que detecta gestos y los transforma en eventos de este sistema operativo.
- **TUIOKinect:** además de tracker, detecta gestos de las manos, capturadas con el dispositivo Kinect.

3.2.3 FRAMEWORKS

En esta subsección se hace referencia a entornos de trabajo que permiten generar aplicaciones enteras. En muchos casos serán entornos conocidos, diseñados quizá con otros fines, pero que convenientemente adaptados trabajan muy bien con tecnologías multitouch.

- **Processing:** es un entorno para la creación de aplicaciones multimedia, normalmente con carácter artístico, y que no requiere grandes conocimientos de programación, puesto que maneja un sencillo lenguaje propio.
- **Silverlight:** entorno de desarrollo multimedia de Microsoft, con soporte web.

- **Qt:** archiconocido entorno de programación de interfaces. Adaptado al multi-touch mediante qtui.
- **Multitouch Cocoa framework:** utilizado en Mac OSX.
- **Adobe Flash:** probablemente el entorno más utilizado en la actualidad, está adaptado al menos de una decena de formas a diversos protocolos multitouch.

3.3 VISUALIZACIÓN 3D Y MULTITOUCH

Es muy habitual que los desarrolladores hagan interfaces muy sencillos para conectar las capacidades multitáctiles de un dispositivo con las aplicaciones de visualización tridimensional. Por ejemplo, una vez desarrollado un controlador que funcione como el propio ratón del sistema, éste se utiliza con los controles propios de la aplicación, siendo esta conexión completamente antinatural e ineficiente.

En este proyecto se busca una **integración nativa** de las ventajas que los dispositivos multitouch ofrecen con los visualizadores. A pesar de que existen numerosos ejemplos de esta integración, no existe demasiada información publicada acerca del tema. Probablemente esto se deba en parte a la lucha que las grandes compañías mantienen sobre las patentes, en parte por el carácter comercial de software y hardware.

A día de hoy, el autor de este trabajo no ha sido capaz de encontrar publicaciones científicas al respecto, sino solamente unos cuantos vídeos o bien de desarrolladores amateur, o bien de software comerciales.

4 VISIÓN GLOBAL, METODOLOGÍA Y PLANIFICACIÓN

4.1 METODOLOGÍA

Para la elaboración de este proyecto se ha optado por una **metodología clásica o en cascada**, aunque aplicada de una forma diferente, debido a la división en capas.

La ciclo de vida clásico es un proceso secuencial en el cual las etapas de análisis, diseño, implementación y prueba se siguen siguiendo un riguroso orden (Rerych, 1970). Es probablemente el más conocido y utilizado antes de la introducción de las nuevas metodologías.

El carácter unipersonal y secuencial de este proyecto hace que este método de trabajo se adapte adecuadamente a los objetivos, si bien debe ser adaptado para la estructura por capas comentada anteriormente, como se puede ver en el siguiente diagrama (Ilustración 14).

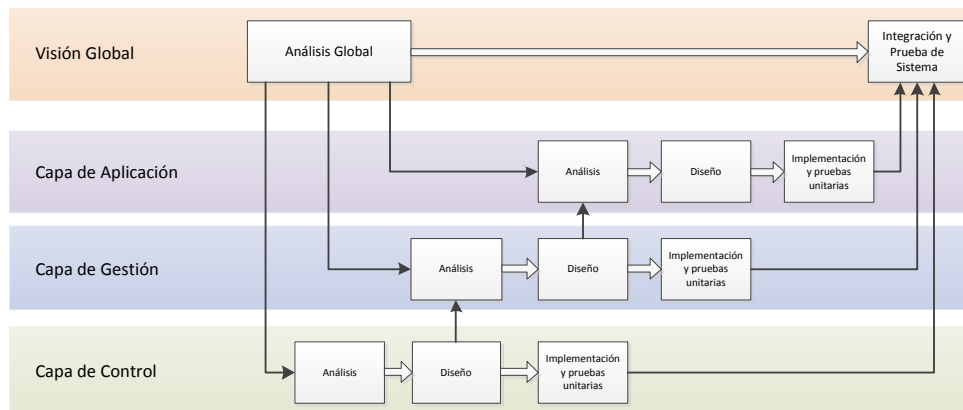


Ilustración 14. Metodología del proyecto: adaptación por capas del ciclo de vida clásico.

Además de un análisis global inicial, algunas de las etapas de desarrollo de las capas son dependientes de aquellas inmediatamente anteriores, y todas ellas convergen en una última etapa de integración y prueba de sistema.

Esta **estructura *down-top*** permite utilizar parte de las fases de diseño de las capas inferiores como entrada al análisis de las capas superiores, de forma que el sistema completo se estructure de forma coherente desde un primer momento.

Para mayor simplicidad en la lectura de esta memoria, se ha decidido dividir por capítulos cada una de estas capas, siendo este capítulo el que describe la visión global, y los siguientes cada una de las capas, con todo su proceso de análisis específico, diseño e implementación.

Es importante remarcar que la mayor parte de los diagramas utilizados en esta memoria siguen utilizando el **Lenguaje de Modelado Unificado, o UML**, en su versión 2.0. Fundamentalmente se utilizarán diagramas de paquete, diagramas de clase y diagramas de

secuencia. Por las características específicas del proyecto, en caso de hacer excepciones en alguna norma, serán debidamente remarcadas con comentarios estándar.

4.2 ANÁLISIS GENERAL

4.2.1 CAPTURA DE REQUISITOS

Todo análisis de un sistema software a desarrollar conlleva un análisis previo de requisitos. Es bien sabido que las típicas aplicaciones de gestión utilizan a menudo las **herramientas CASE**, y los denominados **Casos de Uso** con este fin. Sin embargo, el carácter de bajo nivel e investigación de este proyecto hace que estas herramientas sean de poca utilidad, pues los requisitos más importantes son funcionales y no de comportamiento (Bittner, Kurt, & Spence, 2003). Prueba de ello, es que a nivel de comportamiento el sistema generado sólo dispondrá de un rol, el del usuario, cuya única pretensión consiste en manejar una superficie interactiva con sus manos, y moverse de forma natural en un entorno tridimensional. Sin embargo, se podría llegar a considerar útil en la capa de gestión, donde el actor principal es la aplicación que quiera utilizar el sistema. En este caso si se han utilizado para descubrir las funcionalidades de cara al exterior que debe tener la librería vlmt desarrollada.

Cada uno de los subsistemas generados en cada capa tendrá unos requisitos muy específicos, y normalmente funcionales. En cualquier caso, los objetivos en el desarrollo software del proyecto comentados en la introducción, pueden traducirse en requisitos técnicos:

- **Flexibilidad:** uso exhaustivo de los denominados **Patrones de Diseño** (Gamma, Helm, Jhonson, & Vlissides, 1995).
- **Eficiencia:** el software debe trabajar a bajo nivel, con código lo más cercano al procesador, sin máquinas virtuales ni intérpretes.
- **Simplicidad:** parámetros de funcionamiento configurables con archivos tipo XML que permitan edición rápida e instalaciones fijas.
- **Modularidad:** las librerías generadas son de uso genérico, que permita expandirse en caso de querer incorporar distintas técnicas, protocolos, etc. Una estructura de módulos consistente ahorrará tiempos de *refactoring* en el futuro y permitirá insertar con poco esfuerzo nuevos algoritmos.
- **Multiplataforma:** las librerías de soporte y lenguajes utilizados deben, al menos, soportar las plataformas Windows y Linux.

4.2.2 DIAGRAMA DE PAQUETES

En esta sección veremos la estructura general del software desarrollado en este proyecto. Como se comentó anteriormente, para conseguir los objetivos citados en el capítulo de introducción se han de abordar diferentes problemáticas, que se han dividido en capas de abstracción. La siguiente imagen (Ilustración 15) muestra la correspondencia de estas capas con los paquetes software que, o bien serán desarrollados (marcados en color rojo), o bien intervienen en el proceso de alguna manera (marcados en color amarillo). Además existe una capa vertical, llamada de Soporte, con los paquetes de terceros requeridos para el funcionamiento del sistema.

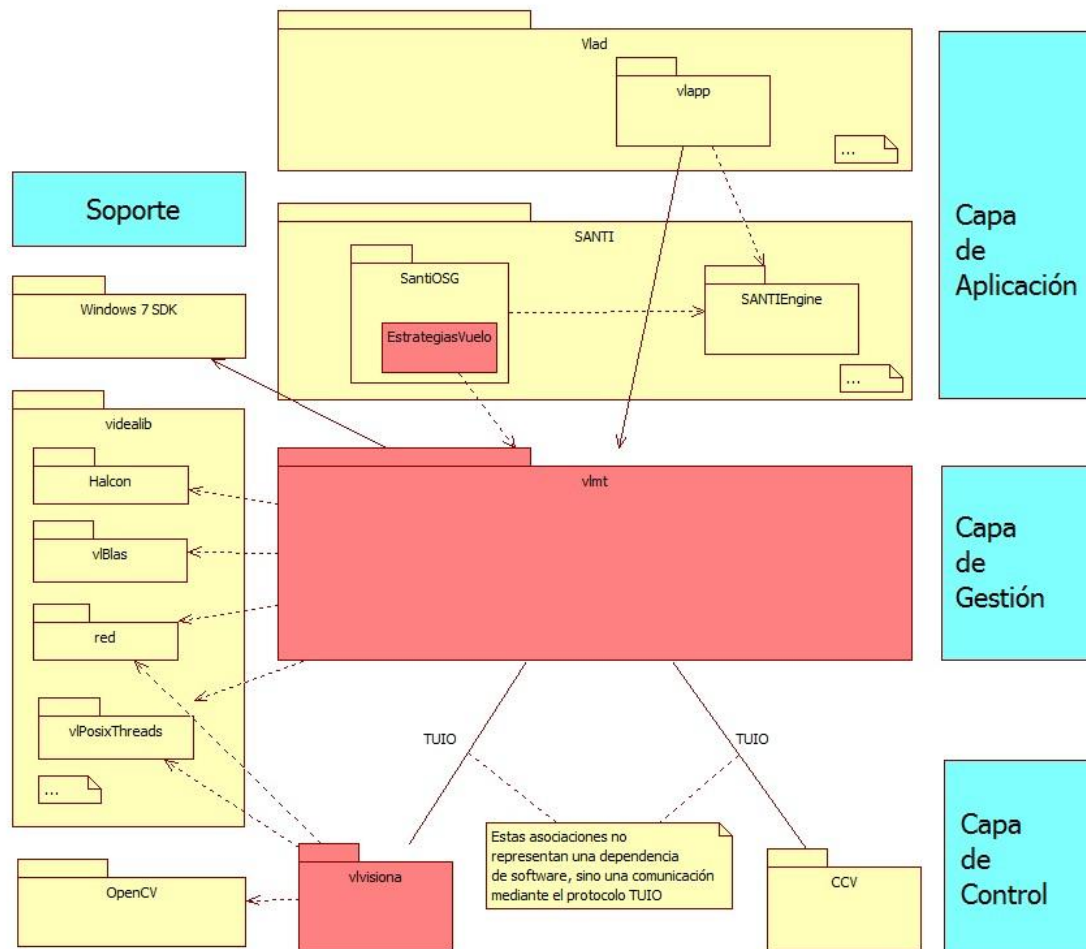


Ilustración 15. Diagrama de paquetes y capas de abstracción. En rojo marcadas las elaboradas en este proyecto.

A continuación se describe cada uno de dichos paquetes, que será explicado en detalle en su capítulo correspondiente.

En la **Capa de Control** existen los dos siguientes paquetes:

- **vvisiona:** librería de Visión Artificial aplicada al ámbito de las superficies multitouch.

- **CCV:** *Community Core Vision* (NUI Group) software desarrollado por el *NUI Group*, análogo a *vlvisiona* pero en un ámbito mucho más específico. Se nombra porque se utiliza sustituyendo a *vlvisiona* en varios de los proyectos desarrollados a posteriori.

La **Capa de Gestión** está ocupada por un único paquete, **vlmt** (de *VideaLab MultiTouch*) que será el encargado de traducir los datos “crudos” provenientes de la Capa de Control, y generar un interfaz único para las aplicaciones. Contiene varios subpaquetes, que serán detallados en su capítulo correspondiente.

La **Capa de Aplicación** abordaría todos aquellos software que quisieran hacer uso del sistema, y sobre los cuales habría que desarrollar módulos (o *plugins*) específicos para adaptar su funcionamiento. Para el caso de este proyecto, se tendrán en cuenta dos enmarcados dentro del *framework* de desarrollo del laboratorio VideaLAB:

- **vlad:** esta librería (proveniente de *VideaLab Application Deployment*) se encarga de hacer el trabajo de “launcher” de cualquier aplicación del laboratorio. Gestiona a bajo nivel la creación de las ventanas, contextos gráficos e interacción con dispositivos de entrada, en cualquiera de los sistemas operativos soportados. En el caso que nos ocupa, debe adaptarse un “ratón virtual” multitouch para poder seguir interactuando libremente con aplicaciones arbitrarias.
- **SANTI:** responde al acrónimo Sistema Avanzado de Navegación sobre Terrenos Interactivo, y se define como un motor gráfico 3D para la visualización del territorio y de modelos arquitectónicos (Seoane, Taibo, Hernández, & Jaspe, 2007). En la última fase de este proyecto, donde se crean dos nuevos paradigmas de

movimiento para la visualización 3D, se adaptarán las técnicas multitouch a los modos de vuelo de este navegador.

En la capa vertical de **Soporte**, dispone de un conjunto de librerías que se han utilizado en el desarrollo, y que se describen en la siguiente sección.

4.2.3 SOFTWARE Y HERRAMIENTAS

Todo el proyecto se plantea utilizando el paradigma de la orientación a objetos, tal y como es común en el entorno de desarrollo VidealAB. El lenguaje utilizado, por compatibilidad y potencia es C++, utilizando de entorno IDE Microsoft Visual Studio 2008.

Como se ha comentado anteriormente, el laboratorio VidealAB dispone de una serie de librerías genéricas, estándar y multiplataforma, para el desarrollo de software, enmarcadas dentro del módulo llamado videalib. Ésta contiene varios sub-módulos; a continuación se describen aquellos utilizados en este proyecto:

- **vlcomun:** incluye las clases más genéricas, como los punteros inteligentes, timers, threads, interfaces para trabajar con ficheros de disco, clases para el uso de redes tipo IP (para paquetería tanto TCP como UDP), etc.
- **vlBlas:** librería matemática. Contiene clases de vectores, matrices, operandos entre ellos, etc.
- **vlXML:** para lectura/escritura, y “parsing” de con ficheros de configuración tipo XML. Utiliza por debajo la librería **Xerces**.
- **vlHalcon:** librería de gestión de dispositivos de entrada para aplicaciones 3D, centrada en el movimiento.

Además de videalib, se han utilizado algunas librerías de soporte desarrolladas por terceros, normalmente de tipo *Open Source*. Las más relevantes son:

- **Posix Threads:** librería de gestión de hilos de proceso (Gallmeister, 1995). Utilizada en la capa de Control y de Gestión.
- **OpenCV:** librería de Visión Artificial desarrollada por Intel y la comunidad de software libre (Intel; Open Source Community), muy popular una buena orientación a objetos desde su versión 2.0, que soporta cientos de operaciones sobre imágenes. Utilizada en la capa de Control.
- **OpenSceneGraph:** es una librería gráfica, que utiliza grafos dirigidos para representar escenas 3D, utilizando a bajo nivel es estándar *OpenGL*. Utilizada en la capa de aplicación, y en general para cualquiera de las pequeñas aplicaciones de soporte o de test que requiriese gráficos.
- **SDK de Windows 7:** el *SDK* estándar de Microsoft para su sistema operativo. La capa de gestión lo utilizará para dar soporte a la capa propia de abstracción que esta plataforma trae por defecto. Utilizada en la capa de gestión.

Como se ha comentado anteriormente, para la aplicación y prueba de los paradigmas de movimiento en espacios tridimensionales que se aborda en este proyecto, se utiliza el Sistema Avanzado de Navegación sobre Terrenos Interactivo (**SANTI**). Este será descrito con más detalle en el capítulo dedicado a la capa de aplicación.

A pesar de que todas las capas han sido implementadas bajo la plataforma Windows, se ha tenido en cuenta en todo momento la compatibilidad de desarrollo con Linux, utilizando código estándar y librerías de terceros soportadas en ambos sistemas operativos.

4.2.4 PLANIFICACIÓN

El presente proyecto se plantea para realizar en 9 meses. A continuación se presenta el correspondiente **diagrama de Gantt** (Ilustración 16), para dar una visión general de los tiempos.

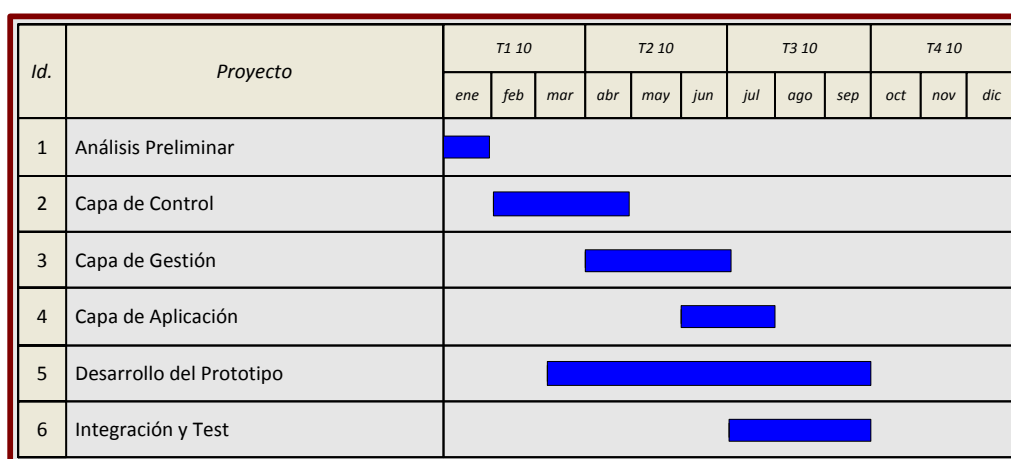


Ilustración 16. Diagrama de Gantt general.

Las áreas solapadas entre los desarrollos de las capas, significan esfuerzo dividido en ambos procesos, debido a que el equipo de trabajo es unipersonal, y representan la interacción entre las etapas de diseño y de análisis entre distintas capas, como se comentó en la sección anterior dedicada a la metodología.

Inicialmente se plantea un mes de análisis previo, y sucesivamente la secuencia de capas. En paralelo, y como soporte a las pruebas se irá desarrollando el proceso durante prácticamente todo el ciclo, y finalmente se utilizarán los dos últimos meses como fase de integración y prueba.

El cálculo de coste por personal en términos brutos sería de unas 1080 horas a 18 € por hora de ingeniero técnico especializado, lo que da un coste aproximado de unos 22680€ brutos.

5 CAPA DE CONTROL

5.1 INTRODUCCIÓN

En este capítulo se profundiza en la capa más baja dentro de los niveles de abstracción comentados anteriormente. Se puede pensar en la **Capa de Control** como un controlador que maneja dispositivos multitáctiles basados en técnicas de visión artificial.

Es muy común que este tipo de dispositivos sean muy dependientes de la aplicación concreta a desarrollar, y es por ello que se ha propuesto el desarrollo de una librería completamente genérica de visión artificial que pueda resolver cualquier problema concreto orientado a este fin. Se verá una aplicación concreta de esta librería a un caso muy particular.

En paralelo se estudia un software de terceros, el *Community Core Vision* o CCV, que cumpliría exactamente el mismo cometido, y que resuelve perfectamente un problema muy concreto como es la detección de toques sobre superficies acrílicas inyectadas con haces de luz infrarroja, principio en el que se basa el prototipo hardware construido. Ambas alternativas son intercambiables, ya que trabajan con el protocolo estándar TUIO.

5.2 LIBRERÍA VLVISIONA

A continuación se describe el proceso de desarrollo de la librería **vlvisiona** (*VideaLab VISION Artificial*), pensada como *framework* genérico para el uso de algoritmos de visión artificial para dispositivos multitouch.

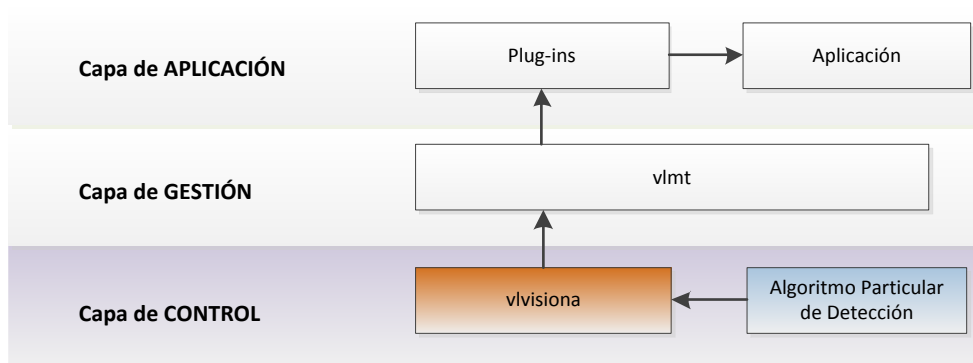


Ilustración 17. Capa de Control.

Es importante aclarar que es un marco de trabajo genérico y como tal, no implementa algoritmo alguno de por sí. En la siguiente sección se verá una implementación concreta para un reconocedor de manos sobre mesas retroproyectadas.

5.2.1 ANÁLISIS

De una forma muy abstracta, cualquier librería dedicada al procesamiento de datos por técnicas de visión artificial en tiempo real sigue unas pautas de funcionamiento básicas, que consisten en ir adquiriendo imágenes de una captura de vídeo, aplicando ciertos algoritmos sobre esos *frames* y obteniendo unos datos de salida, en función de lo que se pretenda conseguir.

En los sistemas en tiempo real, el tiempo de respuesta es un factor clave, así que la optimización de los procesos, aprovechando al máximo las posibilidades de paralelización de las máquinas actuales, será un factor clave en el desarrollo de esta librería.

En la siguiente imagen (Ilustración 18) podemos observar una estructura genérica de cómo sería el funcionamiento de una librería de estas características.

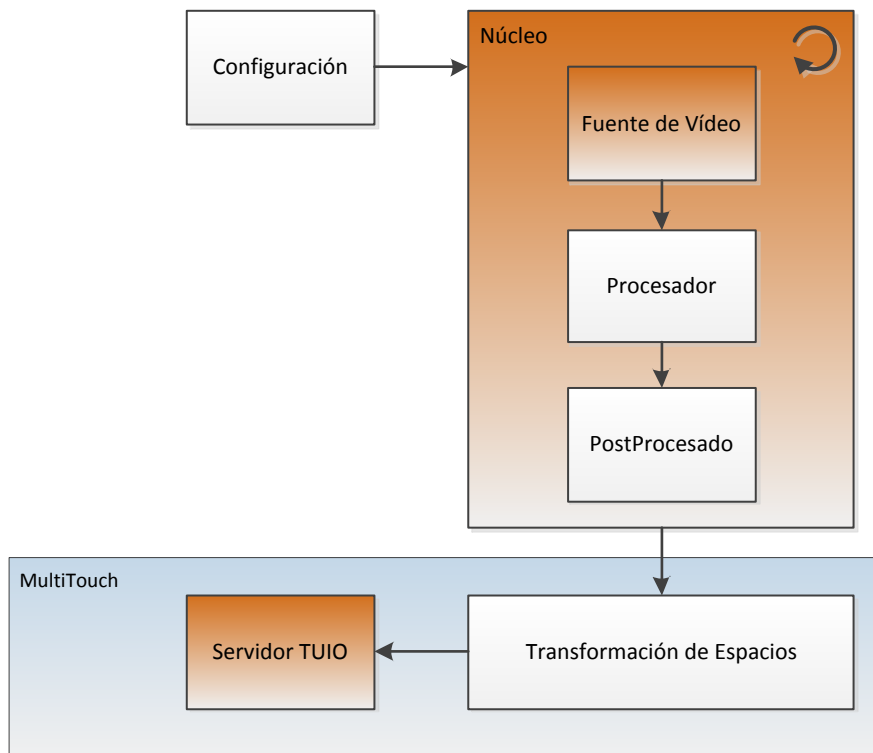


Ilustración 18. Estructura genérica para una librería de Visión Artificial en tiempo real. Los procesos pintados de color naranja son asíncronos.

Partiendo de una configuración paramétrica del algoritmo, se debe entrar en un bucle, que denominamos **Núcleo**, donde distintos módulos ejercen su tarea:

- **Fuente de Vídeo:** ya sea leyendo de un fichero de disco, o de una cámara, es el encargado de proporcionar un flujo continuo bajo demanda de imágenes para procesar.
- **Procesador:** es el que tiene el grueso de algoritmo de visión artificial. Recibe imágenes que analiza y procesa, encontrando datos particulares.
- **PostProcesado:** en algunos casos, es interesante tener otro proceso inmediatamente posterior, para realizar múltiples tareas como la sincronización, transformación de datos, etc.

El resto del proceso es específico para su aplicabilidad a las tareas de multitouch. Por una parte está la **Transformación de Espacios**, requisito indispensable que se explica a continuación. Finalmente, el **Servidor TUIO** será el encargado de traducir y enviar la información de toques a los clientes.

5.2.1.1 Latencia

En la imagen anterior cabe destacar la naturaleza asíncrona de muchos de estos procesos, aquellos pintados en color naranja. En los sistemas en tiempo real se hace imprescindible reducir la latencia lo máximo posible, de forma que las reacciones de la superficie a las acciones del usuario sean lo más inmediatas posibles.

Para ello es fundamental crear un mecanismo eficiente de paralelización mediante *threads* y semáforos situados de forma eficiente. En el apartado de diseño se mostrará de qué forma se ha resuelto esta cuestión.

5.2.1.2 Transformación de espacios

En cualquier aplicación gráfica manejada con un dispositivo externo, debe existir una correspondencia entre el sistema de coordenadas asociado al dispositivo y aquel asociado a la propia aplicación. Por ejemplo, si una superficie interactiva pinta líneas al pasar la mano sobre ella, y esta mano está captada por una cámara, a nivel de software debe haber una transformación entre la posición de la mano dentro del frame captado por la cámara y la situación del cursor dentro de la superficie interactiva. Este proceso se denomina transformación de espacios, y debe estar precedido por un proceso de calibración.

La calibración consiste únicamente en tomar una lista de pares de coordenadas, que refleje la correspondencia entre ambos espacios. Esto normalmente se hace pidiendo al usuario algunas muestras estratégicamente situadas en la aplicación. Una vez confeccionada esta lista, se busca la solución algebraica más precisa al Sistema de Ecuaciones Lineales que origina.

Teniendo la siguiente **matriz de transformación**, que pasa puntos bidimensionales del espacio superficie al espacio aplicación,

$$(x \ y \ 1) \begin{pmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{pmatrix} = (x' \ y' \ 1)$$

El sistema de ecuaciones que debemos resolver queda definido por la matriz del sistema ampliada, que para ser compatible determinado, debe estar compuesto de 3 pares de coordenadas. Este resultado parece correcto, dado que, precisamente con 3 puntos podemos definir un plano. La matriz ampliada del sistema queda entonces como

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & x'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & x'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & x'_3 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & y'_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & y'_2 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & y'_3 \end{pmatrix}$$

Debido a que la calibración no es exacta, pues los pares de coordenadas están sacados por el usuario, normalmente mediante una aplicación, la solución de este S.E.L. debe aproximarse con un método numérico iterativo. Normalmente se utilizará el algoritmo SVD, *Single Value Decomposition* o Descomposición en Valores Singulares.

5.2.1.3 Requisitos

Esta librería tiene una serie de requisitos propios, provenientes en su mayor parte de las necesidades del futuro desarrollador de algoritmos, de la generalidad de las posibles instalaciones interactivas y de su facilidad de uso.

- **Latencia:** como se vio en la sección anterior, es fundamental que su diseño e implementación atienda a unos requisitos de eficiencia de forma que se minimice la latencia.
- **Modularidad:** los algoritmos, deben ser fácilmente implementables como *plugins* para cubrir todos los aspectos del Núcleo, es decir, el Proceso y el Postproceso.
- **Multiplataforma:** en función de la aplicación concreta que se quiera desarrollar, y siguiendo las directivas habituales de trabajo del laboratorio VidealAB, debe poder funcionar igualmente en Linux y en Windows.
- **Fuentes de Vídeo:** precisamente por ser multiplataforma, deben poder implementarse distintas fuentes de vídeo en función de las necesidades. Por ejemplo,

para desarrollo siempre es más útil tener un vídeo en disco pregrabado que se pueda utilizar como banco de pruebas. En entornos de explotación, debe poder funcionar contra cualquier cámara o sistema de vídeo en tiempo real.

- **Red:** al no ser esta librería dependiente necesariamente de la estructura cliente-servidor del protocolo TUIO, el Núcleo debe poder ejecutarse en una máquina distinta a aquella que reciba los datos. Si bien es cierto que este modo no es el común, se ha visto necesario en algunas aplicaciones donde los algoritmos de visión artificial tienen una gran carga de CPU, al igual que la aplicación gráfica. De esta forma se podrán separar ambas tareas en dos máquinas distintas que trabajen en red.

5.2.2 DISEÑO

5.2.2.1 Modelado Estático

En la siguiente imagen (Ilustración 19) se presenta el diagrama de clases simplificado para la librería *vlvisiona*. A continuación se ofrece una explicación textual de las clases más relevantes que lo componen.

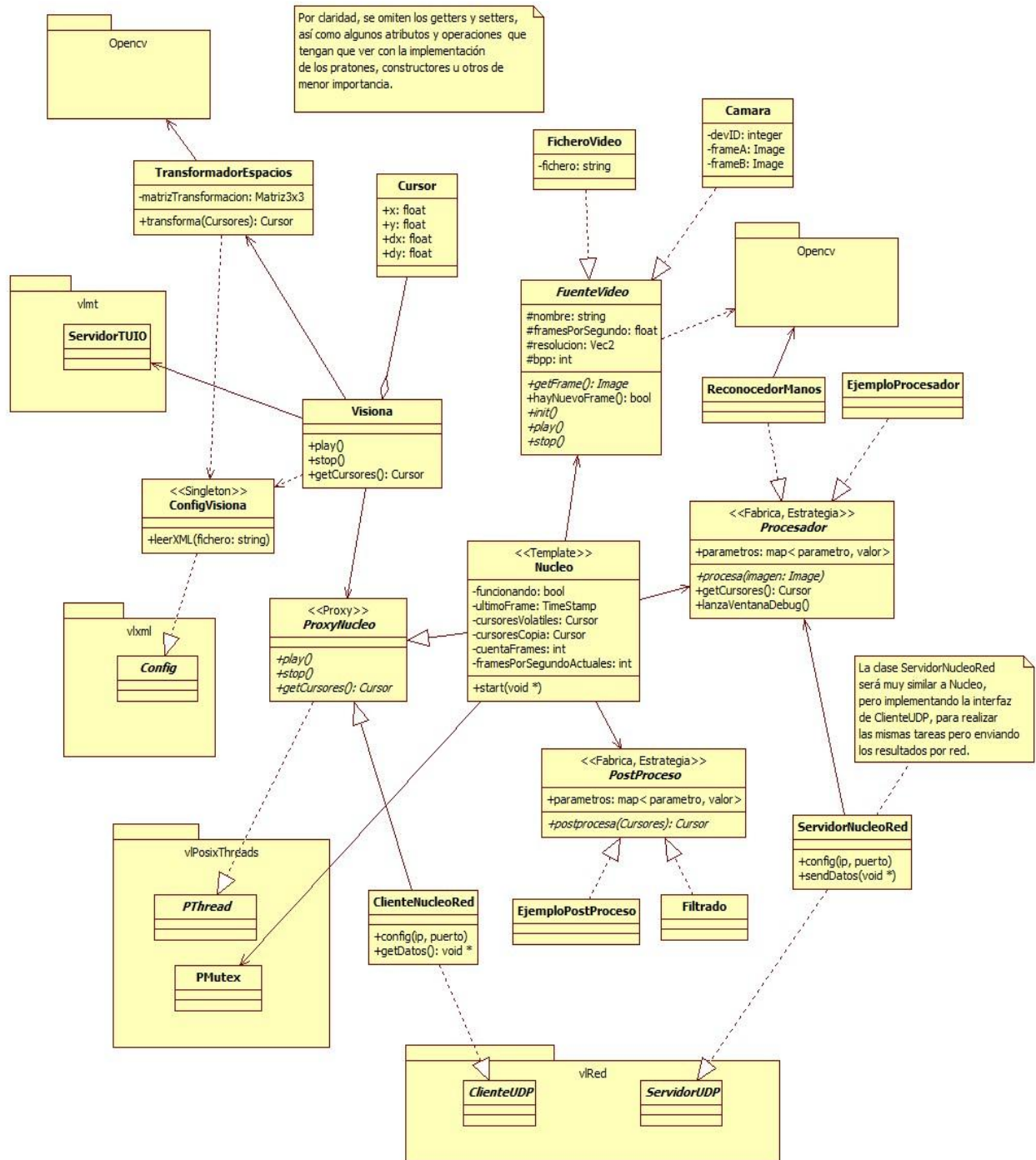


Ilustración 19. Diagrama de clases muy simplificado de vlvisiona.

- **Visiona:** es la clase interfaz para toda la librería. De alguna forma se podría decir que sigue el patrón *fachada*. Hace además de empaste entre todos los componentes que son los que realizan los procesos.
- **Cursor:** es la unidad básica de información, que se agrupa en listas denominadas TListaCursores. Cada cursor lleva su *timestamp*, coordenadas y velocidad.
- **ProxyNucleo:** clase para implementar el patrón *proxy*, de forma que se pueda escoger si utilizar un núcleo real (clase Nucleo) o uno de red (clase ClienteNucleoRed). Es virtual pura.
- **Núcleo:** es la clase más importante de la librería. Sigue el patrón *template*, ya que en función de su configuración, crea instancias concretas de Procesador y Postproceso diferentes, que configuran algoritmos particulares. Además es hija de PThread, clase de las librerías propias videalib, que al llamar a su método *start* lanza un nuevo *thread* de ejecución donde correrá el bucle principal.
- **Procesador:** es una clase virtual pura que implementa una versión híbrida entre el patrón *fábrica* y el patrón *estrategia*, muy utilizado en VidealAB para crear módulos extensibles (en el capítulo 7 se comenta este híbrido). Básicamente los procesadores (estrategias particulares) se registran al inicio de la ejecución y esta clase es encargada de instanciarlas según su configuración. El procesador representa el algoritmo de visión artificial que se usará para encontrar los cursores. Gestiona además una lista de parámetros con los que configurar el algoritmo.
- **EjemploProcesador y ReconocedorManos:** son dos implementaciones de Procesador. La primera es una implementación *dummy*, simplemente para proce-

sos de testeo. La segunda es un algoritmo que se explica en la siguiente sección de este capítulo.

- **Postproceso:** como se vio en el análisis, en ocasiones es interesante tener un procesado siguiente al principal, para hacer algunas operaciones. Esta clase es muy similar a Proceso.
- **EjemploPostproceso y Filtrado:** dos implementaciones concretas de Postproceso. De nuevo, la primera de ellas es una versión *dummy* para procesos de testeo. La segunda es una implementación particular, de la que se hablará en la siguiente sección de este mismo capítulo.
- **ClienteNucleoRed y ServidorNucleoRed:** implementan la estructura comentada en el análisis cliente-servidor para poder alojar el núcleo en una máquina diferente, si esto fuera necesario por motivos de eficiencia. Utilizan los servicios de red de la librería interna *vlred*, mediante paquetería UDP.
- **FuenteVideo:** esta clase es la encargada de proveer regularmente la secuencia de imágenes al núcleo, para que este lo procese. Es virtual pura y debe ser implementada por fuentes de vídeo concretas.
- **FicheroVideo y Camara:** son implementaciones concretas de FuenteVideo, la primera de ellas se configura con un fichero de vídeo (como un *avi*) y mediante el API de OpenCV lo reproduce.; muy útil para desarrollo y *testing*. La segunda permite recibir datos de una cámara conectada al PC, también utilizando OpenCV, que se encarga de la gestión del driver en aquella plataforma que se esté ejecutando.
- **ConfigVisiona:** esta clase representa la configuración de la librería para un uso concreto. Todas las configuraciones del laboratorio VidealAB funcionan here-

dando de la clase base `vl::Config`, que tiene métodos de *parsing* XML para leer y escribir ficheros de configuración.

5.2.2.2 Modelado dinámico

A continuación se presenta el diagrama de secuencia que indica cómo se comporta la librería (de una forma muy simplificada) con la cuestión de la concurrencia, teniendo en cuenta que hay 3 *threads* en ejecución. Cada uno de ellos se marca en color rojo, y corresponden a:

- *Thread* principal de visiona
- *Thread* de la Fuente de Vídeo, para la captura de *frames*
- *Thread* del núcleo que recogiendo y precesando dichos *frames*

En las estructuras de datos intermedias, existe una doble cache' de cursores, sincronizada mediante *mutex* que evitan las escrituras concurrentes en memoria.

Todas las operaciones que se hacen son atómicas, pero sólo a nivel lógico. A nivel de CPU, en algunos casos escribir un tipo de dato mayor que la palabra de la máquina con la que se trabaje (un estructura de datos, o un entero tipo *long*) deja de ser una operación atómica y puede inducir a problemas de concurrencia en acceso a memoria.

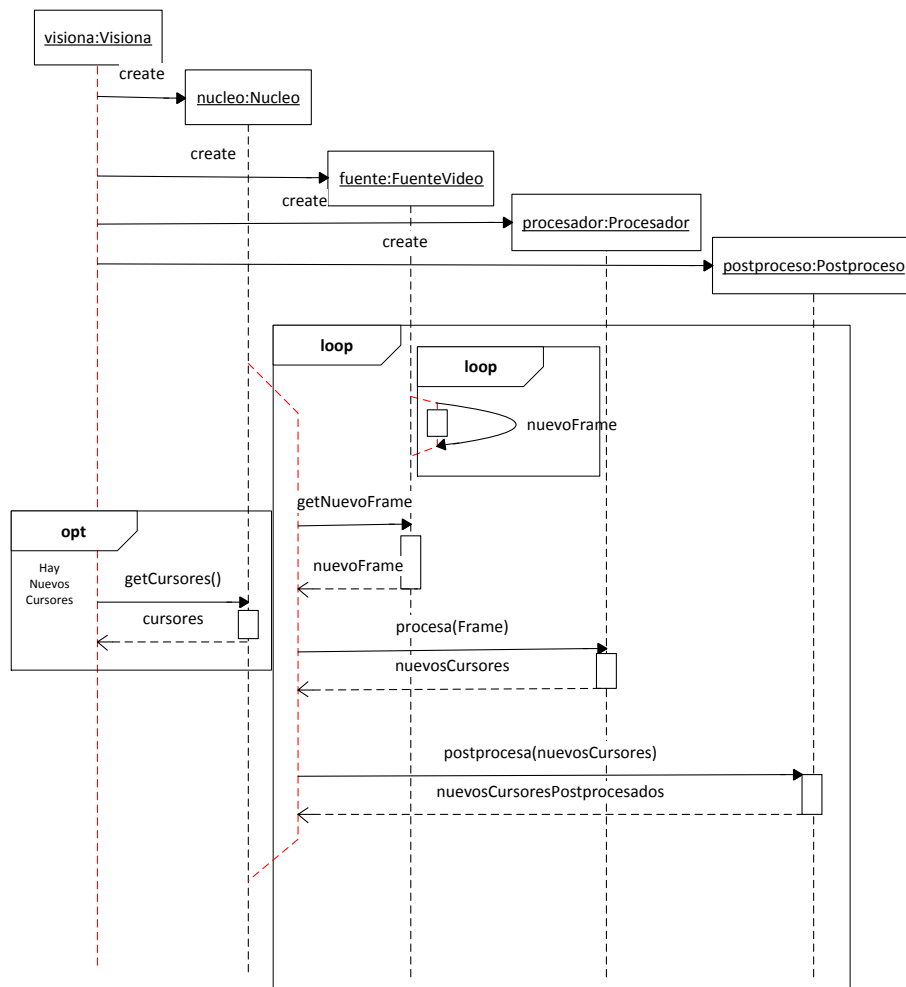


Ilustración 20. Diagrama de secuencia resumido del funcionamiento de vlvisiona.

5.3 EJEMPLO DE APLICACIÓN: RECONOCEDOR DE MANOS

Uno de los usos que se han desarrollado en este proyecto es la instalación interactiva llamada “Fuente Encantada”. En el capítulo de Casos de Estudio, se profundizará un

poco más en esta aplicación, pero describiremos aquí el desarrollo efectuado sobre *vlvisiona* para resolver la parte de la interactividad.

A grandes rasgos, la instalación consiste en una mesa con forma irregular, retroproyectada desde su base, donde un número ilimitado de usuarios pueden interactuar al mismo tiempo con una serie de escenas, moviendo las manos sobre la mesa, pero sin necesidad siquiera de tocarla, produciendo así un “efecto mágico”.



Ilustración 21. Objetivo de "La Fuente Encantada": varios usuarios interactuando "sin toque" en una mesa con forma irregular.

5.3.1 EFECTIVIDAD DE VLVISIONA

Para lograr este objetivo, se implementó un algoritmo de visión artificial sobre la librería *vlvisiona*. Dada su estructura generalista que se acaba de describir, este proceso se reduce a crear unas clases hijas de *Procesador* y, si fuera necesario, de *Postproceso*, disponiendo además de todas las herramientas que *OpenCV* pone a nuestra disposición.

Este ejemplo demuestra perfectamente la potencia de la librería, y de la estructura por capas del sistema presentado, ya que normalmente este desarrollo sería bastante costoso, y obligaría a entrar en particularidades como la gestión del stream de vídeo, la paralelización o incluso el sistema de red, a mayores del algoritmo en sí.

La reducción de costes a la hora de crear instalaciones interactivas utilizando *vlvision* es enorme. Prácticamente cualquier interacción que los creativos diseñen para una instalación, es inmediatamente implementada únicamente pensando en un correcto algoritmo de visión artificial e implementándolo.

5.3.2 PLANTEAMIENTO

Para resolver el problema que plantea esta instalación, se pensó en las condiciones ambientales de la sala donde estaría expuesta. En cualquier aplicación donde se utilicen técnicas de Visión Artificial, resulta de una ayuda inestimable tener un cierto control de las condiciones de iluminación. Después de algunas pruebas, se constató que una cámara convenientemente calibrada y colgada del techo apuntando *zenitalmente* hacia la mesa, podía conseguir un gran contraste entre la imagen *retroproyectada* y los brazos de los usuarios. Además de esta forma se consiguen diferenciar la *polilínea* irregular que define los límites de la mesa. En la siguiente imagen (Ilustración 22) puede verse un *frame* de ejemplo de la captura de la cámara.



Ilustración 22. Ejemplo de *frame* capturado por la cámara, en posición *zenital* hacia la mesa.

5.3.3 EL PROCESADOR: ALGORITMO DE BÚSQUEDA DE BRAZOS

El algoritmo final a la que se llegó para resolver este problema no sigue ninguna técnica común de Visión Artificial, y es el resultado de una labor de investigación. Se basa es en algunas directivas que parecen cumplirse siempre:

- El contraluz de los brazos y la mesa, con la cámara correctamente configurada, genera un contraste lo suficientemente alto como para poder segmentar ambas partes, mediante un umbral.
- La mano está obviamente precedida de un brazo, que necesariamente cruza la *polilínea* que define el borde irregular de la mesa.

- El refresco de la cámara (30 frames por segundo) es lo suficientemente alto como para que el cruz del brazo con el borde en un *frame* esté solapado con el anterior, a pesar de que el usuario mueva muy rápido la mano.
- El brazo de un usuario, a no ser que esté utilizando la instalación de forma indebida (como abalanzándose con todo su cuerpo sobre la mesa), no puede llegar con la mano al borde contrario de la mesa.

Estas premisas han demostrado ser perfectamente válidas para un uso normal de la mesa, aun teniendo en cuenta que su carácter “mágico” hace que el entrenamiento del usuario para utilizarla deba sea nulo.

A grandes rasgos, el algoritmo “umbraliza” la imagen proveniente de la cámara, es decir, aplica un filtro a paso alto y bajo, convirtiéndola en binaria, e inmediatamente después la suaviza con un filtro gaussiano. Con esta imagen, y siguiendo la *polilínea* que define los bordes de la mesa, busca intersecciones de brazos con ella, buscando píxeles blancos (comienzo de la intersección), y siguiéndolos hasta encontrar uno negro (fin de la intersección). Con esto consigue elaborar una lista de brazos metidos en la mesa. Después, en función de sus posiciones relativas con el *frame* anterior identifica qué brazos son “antiguos” (ya estaban antes, y ya tienen un ID único) y cuales son nuevos, asignándoles nuevos IDs. Hecho esto, por cada brazo empieza a seguir su trayectoria hacia dentro de la mesa, mediante un proceso similar al del mínimo gradiente. Esto es, traza secuencialmente líneas de control, partiendo de una primera, paralela al borde de la mesa. A partir de la diferencia de píxeles que le quedan a su izquierda y derecha, aplica pequeñas variaciones al ángulo de la derrota. De esta forma, puede seguir cualquier posición en la que el usuario tenga el brazo, como un codo doblado. Cuando llega

al final del brazo, da una pequeña marcha atrás para buscar el centro de la mano, comparando con parámetros biométricos preestablecidos. Sólo resta actualizar las coordenadas del cursor que corresponde a esa mano, y restarlas con las anteriores para calcular su vector de velocidad.

El único problema encontrado a este algoritmo estriba en la gestión del cruce de manos de dos usuarios, ya que el proceso que sigue el brazo hasta la mano puede continuar hasta llegar al borde contrario de la mesa. En ese caso, este proceso vuelve atrás hasta el punto intermedio donde sitúa ambos cursores.

Este algoritmo se representa gráficamente y por pasos en la siguiente imagen (Ilustración 23).

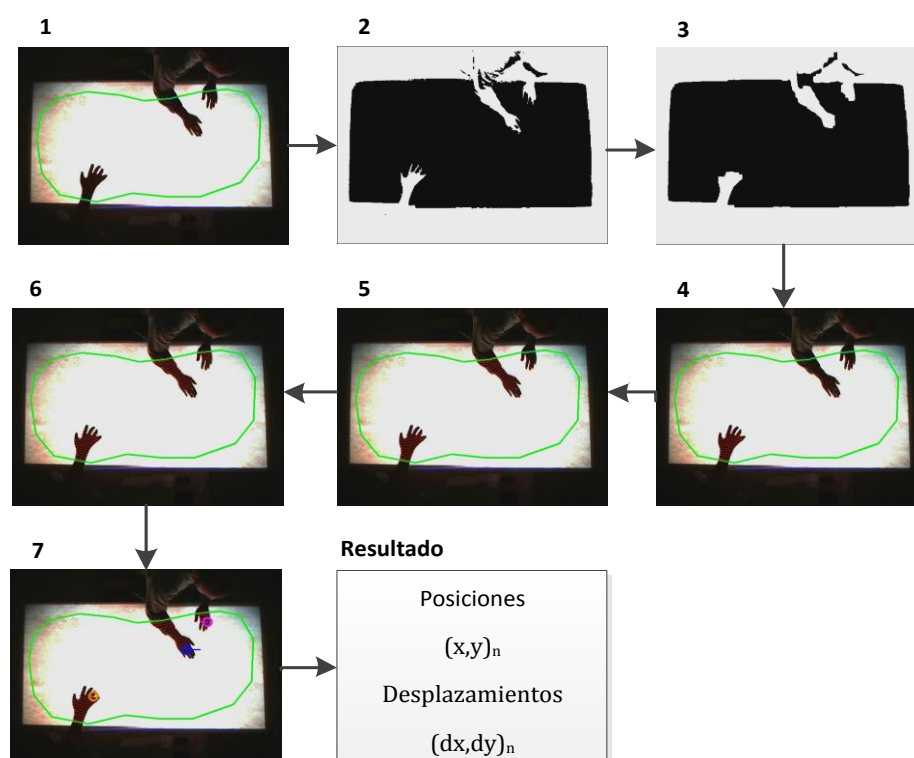


Ilustración 23. Pasos del algoritmo reconocedor de manos.

A continuación se describe el algoritmo en pseudocódigo.

```
Por cada nuevoFrame
    frame = umbraliza(frame)
    frameBinario = filtroGausiano(frame)
    listaBrazos = buscarBrazos(polilinea, frameBinario)
Por cada brazo de listaBrazos
    coordsMano = seguirBrazo(brazo)
    si esNuevo(brazo)
        añadirCursor(coordsMano)
    sino
        id = identificarCursor(brazo)
        actualizarCursor(id, coordsMano)
    Fin si
Fin Por
Fin Por
```

Donde la función buscarBrazos:

```
Funcion buscarBrazos(imagenBinaria) : listaBrazos
    Por cada pixel de la polilinea
        si pixel == blanco
            inicioBrazo = coordsPixel
            avanza pixeles hasta pixel == negro
            finBrazo = coordsPixel
            brazo = (inicioBrazo, finBrazo)
            añadeBrazo(listaBrazos, brazo)
        Fin si
    Fin Por
    devuelve listaBrazos
```

Y la función sigueBrazo:

```
Funcion sigueBrazo(brazo) : coordMano
    coordActual = (brazo.comienzo + brazo.fin) / 2
    anguloAvance = perpendicular(brazo)
    Mientras anchoSondeo > umbral
        Haz
            nPixelesIzq ++
        Mientras siguientePixelIzq(coordActual) == blanco
            Haz
                nPixelesDcha ++
        Mientras siguientePixelDcha(coordActual) == blanco
            anchoSondeo = nPixelesIzq + n PixelesDcha
        si nPixelesIzq > nPixelesDcha
            anguloAvance += incrementoAngulo
        si nPixelesIzq < nPixelesDcha
            anguloAvance -= incrementoAngulo
        coordActual = saltaCoord(anguloAvance, incrementoDistancia)

    si fueraLimites(coordActual)
        devuelve puntoIntermedio
    Fin Mientras
    devuelve coordActual
```

En la siguiente imagen () se puede apreciar con más detalle el *frame* de depuración, producto de la aplicación del algoritmo .

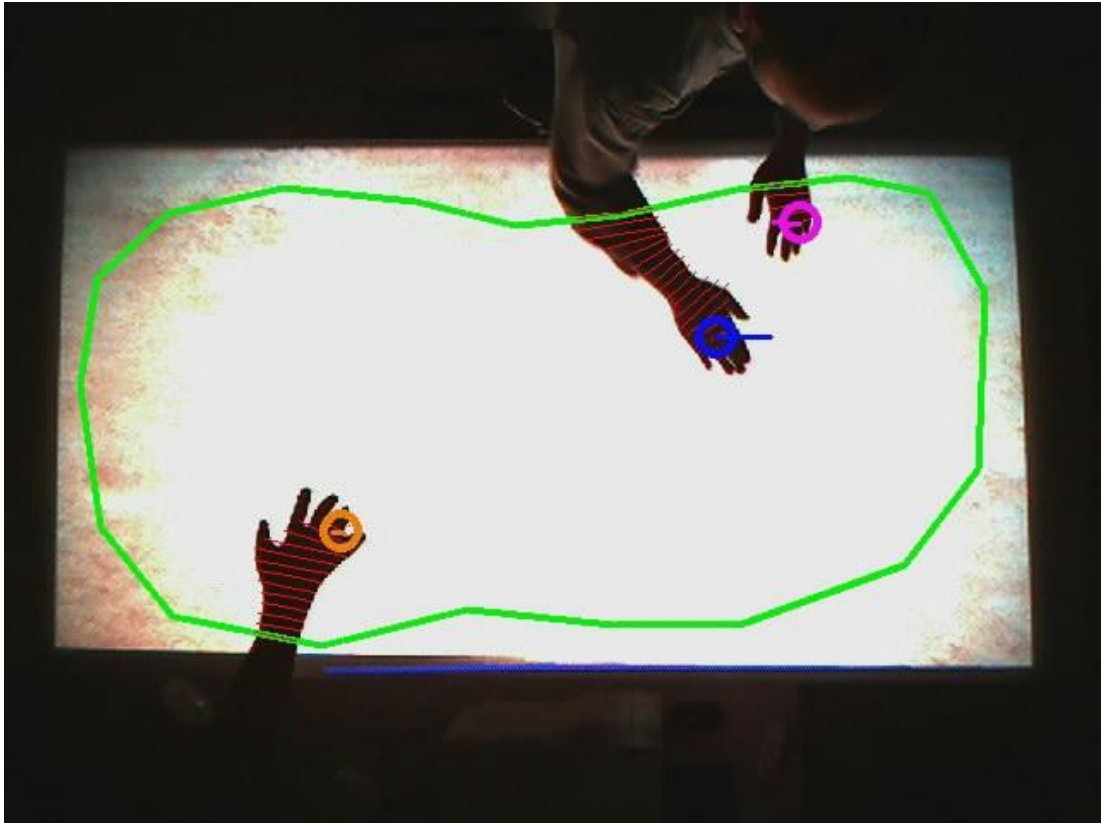


Ilustración 24. *Frame* de depuración de salida del algoritmo reconocedor de manos.

5.3.4 EL POSTPROCESO: ESTABILIZACIÓN DE LOS CURSORES (FILTRADO)

En la sección anterior, vimos que el módulo de PostProceso era opcional. En este caso se utilizó para filtrar las coordenadas de salida de las manos, puesto que el algoritmo anteriormente descrito no tiene precisión suficiente como para obtener una posición “suave” o estable en los sucesivos frames. Esto se traduce en cursores que “bailan” demasiado alrededor de su auténtica posición.

Este problema se puede resolver sencillamente mediante un filtrado de los datos de salida. Se ha escogido un filtro de tipo **F.I.R.** (Finite Impulse Response), que se basa en

obtener medias ponderadas de los valores actuales y pasados, para cada coordenada (Ramirez, Barros, & Lima, 2002). **El orden del filtro N** , representa el número de valores anteriores utilizados

$$y_n = \sum_{k=0}^{N-1} b_k x_{(n-k)}$$

Donde para la muestra n la salida y es el sumatorio de los N valores anteriores, con la ponderación b . Si simplemente se escoge la media aritmética, queda como

$$y_n = \frac{\sum_{k=0}^{N-1} x_{(n-k)}}{N}$$

Este tipo de filtrados tienen una repercusión directa sobre la latencia. Cuanto mayor sea el orden del filtro, mayor el retardo de la señal. El reto consiste en encontrar el equilibrio entre tener el mayor número de muestras (un filtrado más suave) sin sacrificar una buena latencia.

En la siguiente imagen (Ilustración 25) se puede apreciar perfectamente que el incremento del orden del filtro (N) produce una mayor suavidad en la señal, al mismo tiempo que un mayor retraso en la misma.

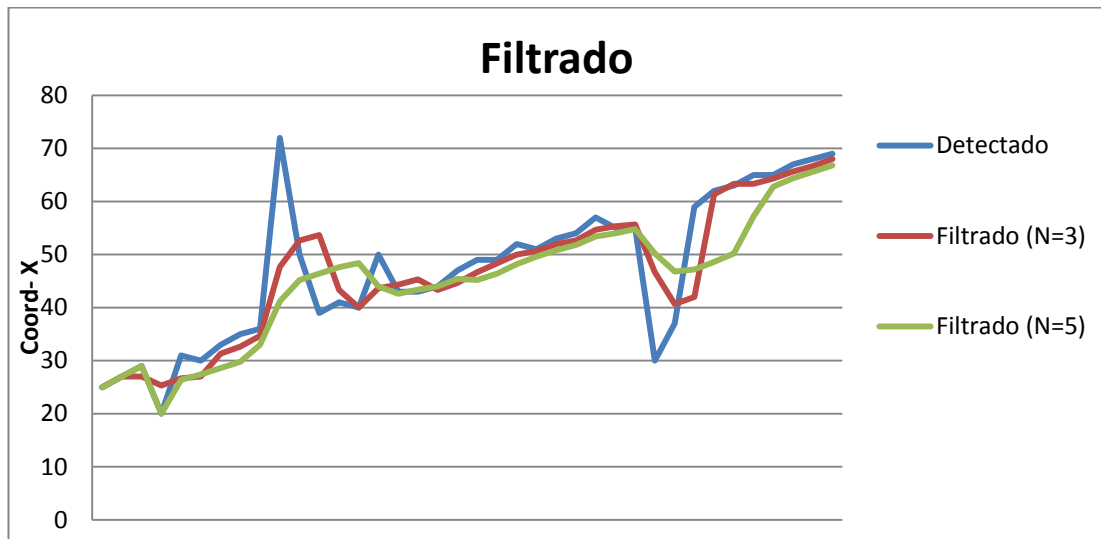


Ilustración 25. Filtrado de los Cursores con F.I.R.

5.3.5 CALIBRACIÓN

Para conseguir configurar correctamente la Matriz de Transformación de Espacios, se creó una pequeña aplicación de calibración. Las características concretas de este proyecto, que utilizaba una doble proyección con blending, requirieron una aplicación de calibración particular, aunque basada exactamente en los mismos principios explicados en la sección anterior.

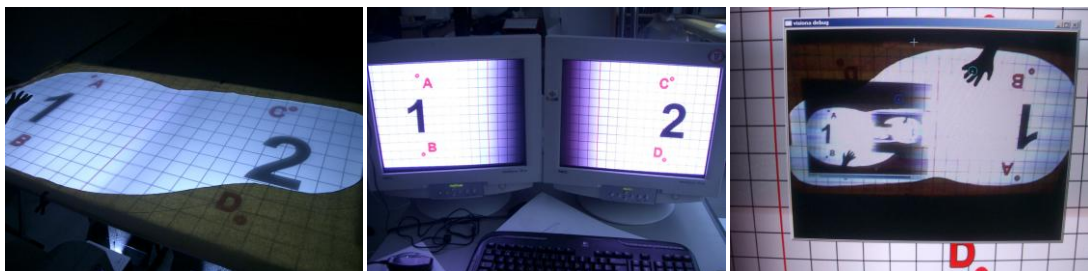
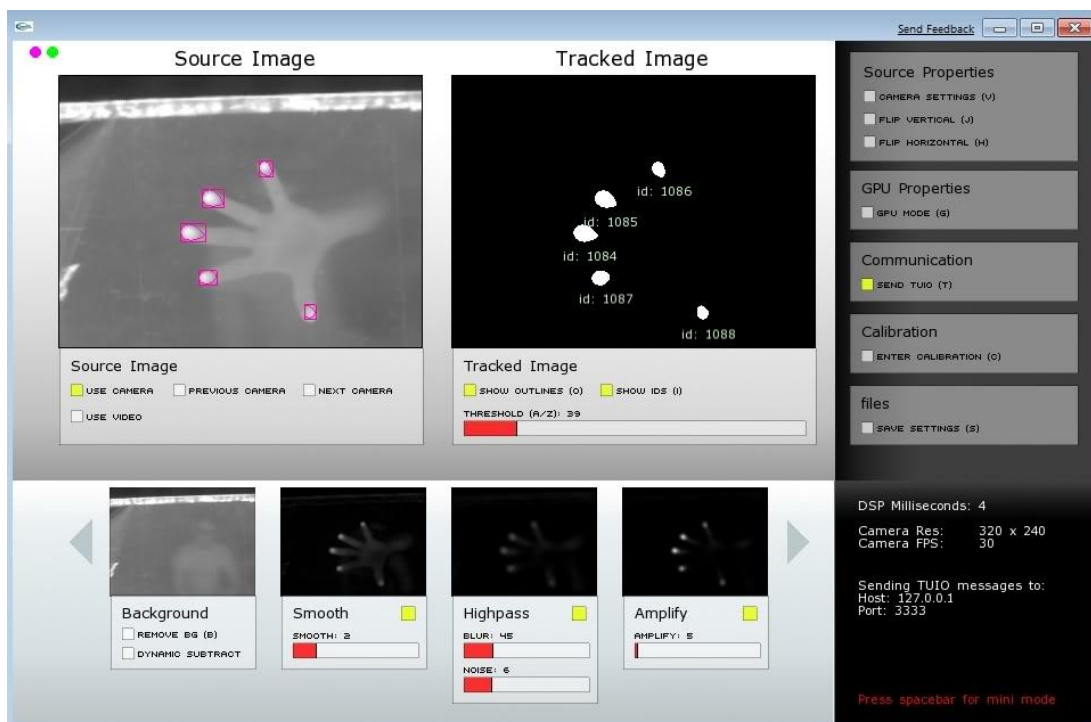


Ilustración 26. Calibración del Reconocedor de Manos.

5.4 COMMUNITY CORE VISION (CCV)

Community Core Vision, más conocida como CCV, es una aplicación libre desarrollada desde el 2006 por el **NUI Group**, grupo de investigación en Interfaces Naturales comunitario, colaborativo y libre (<http://nuigroup.com>)



Consiste en un software multiplataforma de visión artificial, que capta y hace “tracking” en tiempo real de blobs a partir de un stream de vídeo. Para ello, de un modo muy resumido, sigue los siguientes pasos:

1. Capturar imagen de la fuente de vídeo.
2. Filtrado de la imagen:
 - a. Eliminación del fondo.
 - b. Suavizado de la imagen con un filtro gaussiano.

- c. Filtro a paso alto.
 - d. Amplificación del resultado.
3. Con la imagen resultante, aplica las técnicas de búsqueda de blobs contenidas en la librería OpenCV.
 4. Transforma las coordenadas en función de una calibración previa.
 5. Envía los datos mediante alguno de los protocolos que soporta, fundamentalmente TUIO.

El lector no debe sorprenderse al ver que la estructura se asemeja enormemente a la seguida por vlvisiona, puesto que como ya se explicó, estas etapas son las fundamentales por cualquier aplicación con este objetivo.

Se podría decir, para mejor comprensión, y salvando las distancias, que **CCV podría ser una “instanciación” (o aplicación concreta) de la librería vlvisiona**. Es por este motivo que uno y otro software son perfectamente intercambiables, y si se hace mención en esta memoria de CCV, es porque en el prototipo fabricado se ha utilizado esta plataforma y no vlvisiona.

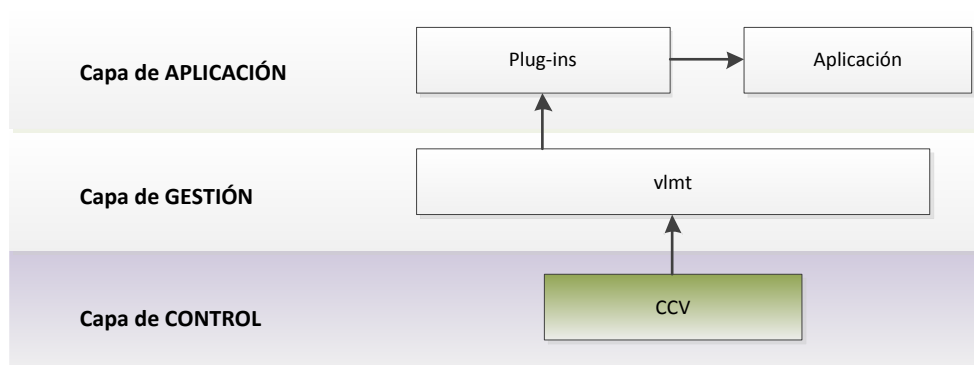


Ilustración 27. Sustitución de vlvisiona por CCV en la capa de Control, para el desarrollo del prototipo.

Como se verá, el tipo de problema de visión artificial que plantea el prototipo es ampliamente resuelto por CCV, y siendo esta una herramienta libre, carece de sentido reinventar la rueda. El uso de herramientas libres que solucionen problemas eficientemente contribuye en buena medida a lo que hace grande la comunidad open source. En cualquier caso, es de ley comentar que durante el desarrollo de este proyecto el autor ha participado activamente en esta comunidad, aportando dudas y soluciones, así como incluso algunos fragmentos de código.

6 CAPA DE GESTIÓN

6.1 INTRODUCCIÓN

Este capítulo trata la capa intermedia de software denominada **Capa de Gestión**, donde se integran dispositivos arbitrarios de multitouch y se ofrece a la capa siguiente una interfaz sencilla y única para tratar con estos datos.

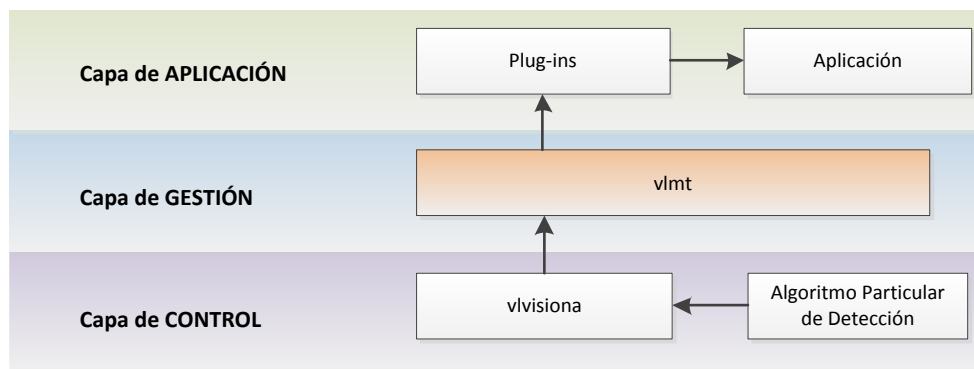


Ilustración 28. Estructura de capas. Capa de Gestión (vlmt).

Esta capa está compuesta exclusivamente por la librería vlmt. Su propósito es hacer de interfaz común para cualquier aplicación que quiera trabajar con uno de los dispositivos multitouch implementados.

6.2 LIBRERÍA VLMT

6.2.1 ANÁLISIS

Como se comentaba, el propósito de **vlmt** es gestionar distintos tipos de dispositivos multitouch, y ofrecer a las aplicaciones una **interfaz única de acceso a los datos** que éstos proporcionan.

6.2.1.1 Casos de uso

En este caso, si es interesante aplicar la técnica de los casos de uso, para descubrir de qué manera deben interactuar las aplicaciones con el interfaz único proporcionado.

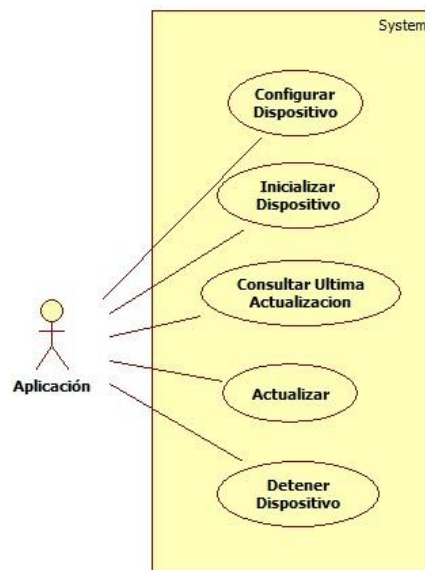


Ilustración 29. Casos de Uso de vlmt.

Los casos de uso de vlmt no podrían ser más sencillos. Una aplicación sólo necesita elegir y configurar el dispositivo que necesite usar, inicializarlo y detenerlo, y actualizar los datos que posee en caso de que éstos estén obsoletos.

No es de extrañar que sean tan pocos casos de uso, ya que, de cara a las aplicaciones, el funcionamiento de los dispositivos multitouch es muy simple. Podría decirse que es como un ratón con múltiples cursores.

6.2.1.2 Robustez y Latencia

El único problema importante a salvar es la cuestión de la fiabilidad de las transacciones de datos. Esta librería debe implementarse de forma que el desfase entre los datos obtenidos por el dispositivo hardware y aquellos entregados a la aplicación que los use, es decir, la latencia, sea mínima.

En el apartado de dedicado al diseño del sistema se verá de qué forma se ha resuelto esta cuestión.

6.2.1.3 Múltiples aplicaciones a la vez

Es posible y muy probable que varias aplicaciones (o varios procesos de una misma aplicación) quieran consultar los datos recibidos del dispositivo, con diversos fines. Por ejemplo, puede haber un proceso encargado de reconocer gestos para ejecutar acciones, y al mismo tiempo, otro encargado de saber si se pulsa un botón, tipo ratón.

La librería desarrollada debe gestionar eficientemente consultas periódicas desde distintas partes del código, y controlar qué procesos han sido ya actualizados o están pendientes de hacerlo.

6.2.1.4 Dispositivos

La librería `vlmt` debe soportar dispositivos de la forma más genérica posible. En anteriores capítulos se ha hablado del protocolo TUIO como unificador, a partir del cual una aplicación servidora envía a otra cliente un stream continuo de toques. Si todos los dispositivos multitouch hardware del mercado utilizasen dicho protocolo (que, además, es de uso libre) probablemente no se podría justificar la existencia de la capa de gestión de este proyecto.

Desgraciadamente, esto no es así. Existen otros dispositivos que utilizan protocolos cerrados, o que vienen preparados para trabajar mediante un driver directamente contra el sistema operativo. Además el uso de estas superficies multitouch normalmente es muy atractivo, debido a su facilidad de funcionamiento y su bajo coste.

El mejor de los ejemplos (y el que está implementado en este proyecto) son todas las pantallas (monitores) táctiles que se venden en la actualidad, y que ofrecen 2 o incluso 3 toques simultáneos. Estas vienen preparadas para su uso exclusivo con el sistema operativo Windows 7. Si una aplicación quiere aprovechar sus capacidades, debe programarse utilizando por debajo uno de los *SDKs* o *frameworks* que Microsoft pone a disposición de los desarrolladores.

En un entorno tan dinámico como el empresarial, tener un producto software limitado a un tipo de hardware particular no deja de ser un inconveniente. Adaptarlo a otro tipo de dispositivo supondría un nuevo esfuerzo de desarrollo, y probablemente un gran esfuerzo de *refactoring* del código ya elaborado (Fowler, 2004).

La librería *vlmt* debe soportar distintos tipos de dispositivos multitouch tanto presentes como futuros, y de esta forma evitar este problema. Al final de este capítulo se ofrecen algunos detalles de los dos tipos de dispositivos soportados hasta el momento:

- Aquellos que utilizan el **protocolo TUIO**.
- Aquellos con *driver* a **Windows 7**.

6.2.1.5 Ratón

La primera y obvia función de una superficie multitouch es utilizar los toques como ratón de la aplicación o del sistema. Aunque no es necesario para el desarrollo de este proyecto, que pretende ir mucho más allá dando soporte nativo a gestos en el visualizador 3D, lógicamente es muy útil tener implementado el soporte necesario para utilizar la multitouch como puntero. La librería *vlmt* debe soportar esta característica.

6.2.2 DISEÑO

El diseño elaborado para *vlmt* se puede explicar someramente desde la Ilustración 30. En ella vemos que un único interfaz, denominado MultiTouch, hace de fachada contra las aplicaciones externas. A su vez, el dispositivo configurado se dedica a mantener una doble caché de los cursores detectados. Mientras la implementación concreta del dispositivo que se esté utilizando actualiza un segmento de memoria con la última información adquirida, la fachada MultiTouch hace peticiones al dispositivo. Esta doble caché, que funciona de forma asíncrona en un thread independiente, es gestionada por unos *mutex* que hacen función de semáforos. Si una aplicación solicita nuevos cursores,

el dispositivo devolverá los últimos que tenga, uno por uno, independientemente de si en ese instante los está actualizando.

Por otra parte, la fachada MultiTouch es la encargada de llevar la gestión de “solicitan-tes” de cursores, guardando *timestamps* de cuándo han sido éstos actualizados y si dis-ponen de nuevos datos. Con esto se evita el *pooling* masivo que los procesos puedan hacer a la librería, y se asegura que éstos siempre trabajarán con los últimos datos dis-ponibles, pero no consumirán tiempo de CPU innecesariamente.

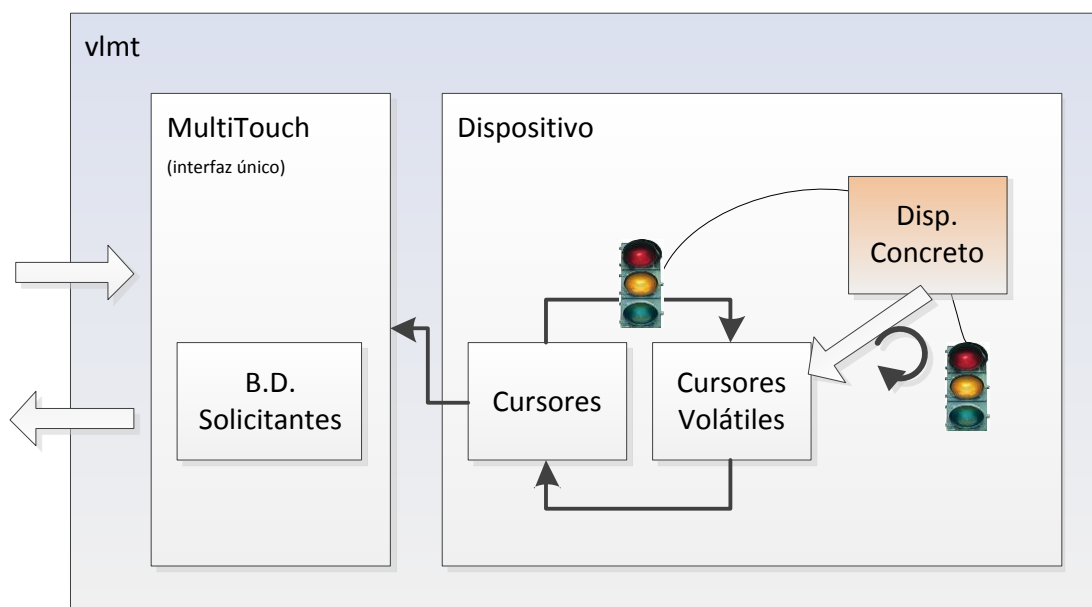


Ilustración 30. Funcionamiento abstracto de vlm. El proceso marcado en rojo es asíncrono.

6.2.2.1 Modelado estático

En la siguiente figura (Ilustración 31) se presenta el diagrama de clases de la librería vlm y a continuación una explicación textual de las más importantes, y sus patrones.

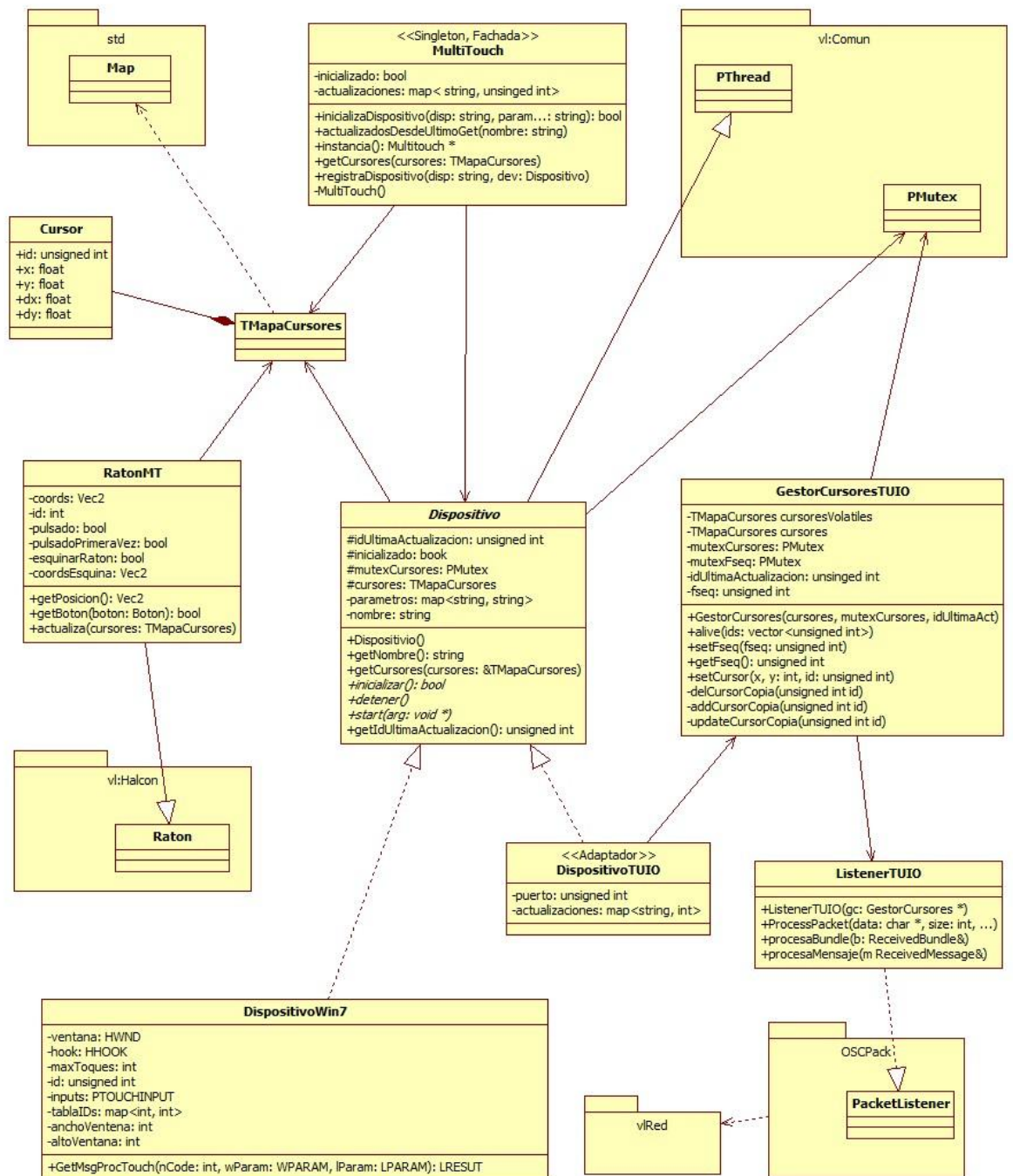


Ilustración 31. Diagrama simplificado de clases de vlmt.

- **Multitouch:** es la clase principal de la librería, y la que hará de interfaz contra las aplicaciones externas. De alguna forma es también una *fachada* de Dispositivo, ya que recoge en unas pocas funciones las operaciones que se tienen que ejecutar contra él. Además es un *singleton*, para que pueda ser accesible desde cualquier parte. Lo más importante es que mantiene una estructura tipo mapa (*actualizaciones*) para ir registrando los solicitantes de cursores y el timestamp de su última actualización. Se ha decidido esta estructura y no un patrón *observador* por cuestiones de eficiencia.
- **Cursor y TMapaCursores:** es la unidad básica de información multitouch. Representa un cursor en pantalla, su velocidad e identificador, con coordenadas normalizadas. Se crea una clase tipo mapa para agruparlas, porque normalmente te querrá acceder a ellos por ID, no de forma secuencial.
- **Dispositivo:** clase abstracta pura, que define un dispositivo multitouch. Hereda de la clase *PThread* de *videalib*, que hace referencia a *Posix Thread*. Los dispositivos concretos deberán definir las funciones *inicializar*, *detener* y *start*, en la cual se lanza un nuevo *thread* en el que están obligados a mantener actualizada la estructura *cursores*. No hace falta que lo hagan de forma *threadsafe*, porque de esto se encargará *getCursores*. Sólo se pide que bloqueen el mutex *mutexCursores* cuando escriban en él.
- **DispositivoTUIO:** una de las implementaciones de *Dispositivo*, que da soporte a aquellos que utilicen en protocolo TUIO. Por cuestiones de modularidad, es un adaptador de *GestorCursoresTUIO*, que tiene la auténtica carga de conocer el protocolo.

- **GestorCursoresTUIO**: como decíamos, es el encargado de recibir un *stream* de datos TUIO, analizarlos y sacar la información válida concerniente a los cursores.
- **ListenerTUIO**: es la contrapartida a la clase vista anteriormente de *ServidorTUIO*. Este, mediante un *PacketListener* de *OSCPack*, escucha por el puerto asignado UDP el chorro de datos y lo descompone en paquetes, que serán interpretados por el *GestorCursoresTUIO*.
- **DispositivoWin7**: el otro dispositivo implementado hasta la fecha. Esta clase (que contiene macros para ser compilada sólo en su entorno), permite acceder a los mensajes multitouch de forma nativa en el sistema operativo Windows 7, y adaptarlos como cursores de *vlmt*. Los procesos que sigue para conseguirlo se explican en la siguiente sección de este capítulo, dedicada a los dispositivos.
- **RatonMT**: esta es la clase que simula un ratón ordinario. Es un clase hija de *Halcon::Raton*, que es la representación de un ratón dentro del código interno de *videalib*. Cualquiera puede registrar esta clase como un ratón en el gestor de dispositivos de su aplicación, y desde ese momento la superficie multitáctil actuará como tal (sólo con un toque, el resto los descarta). En realidad, actúa como un solicitante de cursores más, haciendo peticiones a la fachada *MultiTouch* con el nombre “*ratonMT*”.

6.2.2.2 Modelado dinámico

Igual que en *vlvisiona*, se presenta ahora () un pequeño diagrama de secuencia muy resumido, en el que el lector habituado a la notación UML podrá divisar de forma general el funcionamiento estándar de la librería. De nuevo los *threads* pintados en rojo.

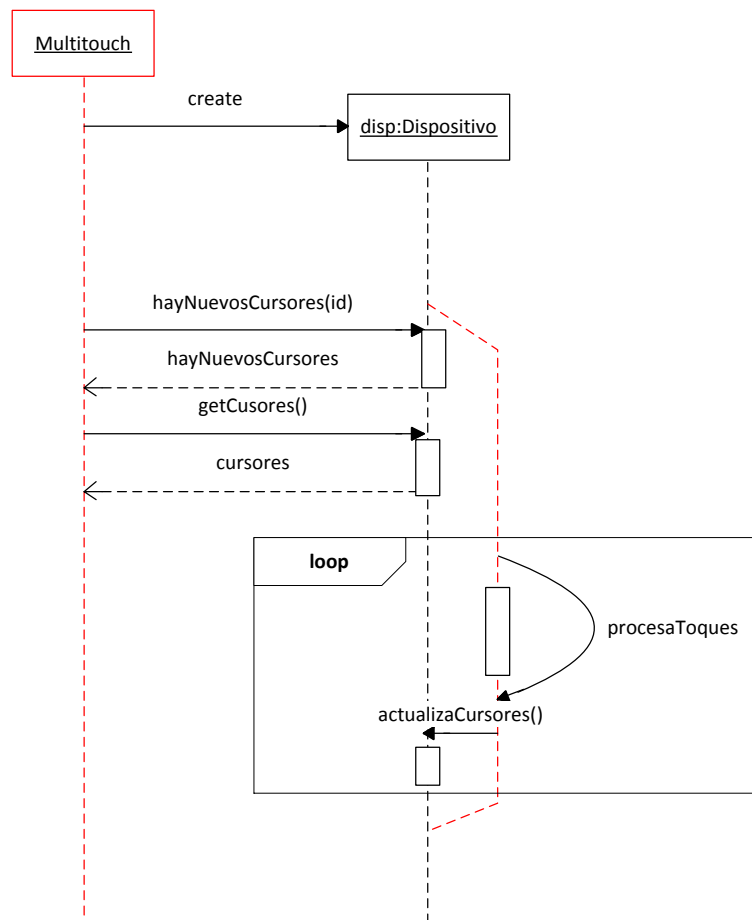


Ilustración 32. Diagrama de secuencia resumido para la librería vlmt.

6.3 DISPOSITIVOS SOPORTADOS

6.3.1 TUIO

En el capítulo de fundamentos básicos ya se introdujo el protocolo TUIO como un *framework* abierto para la comunicación entre dispositivos multitouch y aplicaciones, con estructura cliente – servidor. A continuación se detallan algunas de las características

más importantes de TUIO, así como su especificación técnica de la versión 1.1, la utilizada en este proyecto.

6.3.1.1 Características generales

Como ya se ha comentado, el protocolo TUIO se codifica utilizando el formato de *Open Sound Control*, que provee una codificación binaria eficiente para la transmisión de datos de control arbitrarios. El canal de transporte por defecto de TUIO es mediante un socket de tipo UDP, y el puerto estándar es el 3333. Al conjunto del protocolo y este método concreto de transporte es a lo que se suele denominar TUIO/UDP, y la mayor parte de las implementaciones están basadas en él debido a su simplicidad y velocidad en redes tanto de ámbito local como Internet. En cualquier caso, cualquier otro método de transporte podría ser usado, como TCP u otros no basados en las comunicaciones IP, como FLC (para aplicaciones Flash, utilizando memoria compartida). Como veremos a continuación, las conexiones UDP se adaptan perfectamente a la filosofía de envío de mensajes, donde prima la velocidad y volumen a la fiabilidad de recepción de paquetes.

6.3.1.2 Mensajes

El protocolo TUIO define básicamente dos tipos de mensajes:

- **Mensajes SET:** utilizados para comunicar información acerca del estado de un objeto, como su posición, orientación, etc.
- **Mensajes ALIVE:** indican el conjunto de objetos presentes en la superficie mediante una lista de IDs únicos.

Para evitar posibles descontroles por pérdida de paquetes (algo posible utilizando UDP), el protocolo TUIO no utiliza paquetes de tipo ADD o REMOVE. Esto quiere decir

que para conocer si el conjunto de objetos en la superficie aumenta o disminuye, debe examinarse la secuencia temporal de paquetes tipo ALIVE.

Además de los dos mensajes mencionados, existen dos tipos de más de mensajes, quizá menos relevantes pero útiles para ciertos cálculos:

- **Mensaje FSEQ:** con otro ID único, que define una secuencia de *frames* en los que se computan los objetos de la superficie.
- **Mensaje SOURCE:** que identifica la fuente de mensajes TUIO, y que permitiría *multiplexar* varias superficies.

El formato que siguen los paquetes es el definido por OSC, con cabeceras tipo */tuio/2Dcur*, que hacen alusión al *profile* utilizado. En este caso, TUIO define, entre otros, *2Dcur* y *3Dcur*, que significan cursores bidimensionales y tridimensionales. Un paquete (o *bundle*) típico sería como sigue (con variables en vez de datos):

```
/tuio/2Dcur source application@address
/tuio/2Dcur alive s_id0 ... s_idN
/tuio/2Dcur set s_id x_pos y_pos x_vel y_vel m_accel
/tuio/2Dcur fseq f_id
```

Comienza con un mensaje tipo SOURCE , que define la superficie fuente del paquete, seguido de un ALIVE, para saber qué objetos o cursores siguen estando presentes. Después vendrá un número indeterminado de mensajes tipo SET, que definen, por este orden, la coordenada x e y del cursos, los componentes x e y de su velocidad y, su aceleración. Por último, el mensaje FSEQ ofrece la información del número de *frame* en el que se encuentra la computación de los cursores sobre la superficie multitouch.

Es importante remarcar que los IDs asignados a los distintos objetos definidos por TUIO son únicos para cada sesión, es decir, que siempre se verán incrementados mientras la aplicación servidora o *tracker* no se reinicie, y además se asignan por estricto orden de aparición.

6.3.1.3 Computación de parámetros

Para una mejor comprensión de este apartado, se adjunta una tabla con los posibles parámetros de los distintos objetos que el protocolo TUIO define.

S	ID de esta sesión	Int32
I	ID de clase	Int32
x, y, z	Posición	Float32, rango 0..1
a, b, c	Ángulo	Float32, rango 0...2PI
w, h, d	Dimensión	Float32, rango 0..1
f, v	Área, Volumen	Float32, rango 0..1
X, Y, Z	Vector de velocidad	Float32
A, B, C	Vector de rotación	Float32
m	Aceleración de la moción	Float32
r	Aceleración de la rotación	Float32
P	Parámetro abierto	Tipo definido por OSC

El sistema de coordenadas de TUIO se normaliza para cada eje, variando en un rango entre 0 y 1. Para calcular las coordenadas X e Y del *profile* tipo 2Dcur, el utilizado mayoritariamente en este proyecto, simplemente se debe dividir los valores de las auténticas coordenadas entre el tamaño total de la superficie:

$$x = \frac{sensor_x}{sensor_{ancho}} \quad y = \frac{sensor_y}{sensor_{alto}}$$

Los vectores de movimiento, así como la aceleración, tienen en cuenta el incremento de tiempo (*dt*) entre los *frames* de cálculo, medido en segundos. Por ejemplo, un toque de

dedo de izquierda a derecha generará un vector de movimiento $v = (1,0)$. Por otra parte, la aceleración de la moción dependerá de la variación del vector velocidad en ese mismo incremento de tiempo.

$$x = \frac{\frac{sensor_{dx}}{sensor_{ancho}}}{dt} \quad y = \frac{\frac{sensor_{dy}}{sensor_{alto}}}{dt}$$

$$m = \frac{velocidad - ultima_velocidad}{dt}$$

Las unidades de la velocidad de rotación se definen según una rotación completa por segundo. Es decir, una rotación completa en un segundo significará una velocidad de rotación de 1. Se calcula con el incremento de ángulo normalizado a radianes y divididas entre dt . La aceleración de la rotación se calcula de forma análoga a la de la moción.

$$A = \frac{\frac{a - ultimo_a}{2\pi}}{dt}$$

$$r = \frac{A - ultimo_A}{dt}$$

6.3.1.4 Eficiencia y Fiabilidad

En la comunicación TUIO, es muy importante tener en cuenta que el parámetro que más prima es la latencia, ya que definirá cómo de inmediata es la reacción de la aplicación interactiva a nuestros toques. De una forma visual, si colocásemos unos cursores reales bajo los toques en la superficie, la latencia marcaría cómo de rápido éstos siguen a nuestros dedos cuando movamos la mano. Se podría decir que este es el motivo principal de utilizar de medio de transporte el protocolo UDP. Utilizando otras opciones, como TCP, garantizaríamos la comunicación de todas y cada una de las posiciones cal-

culadas por el *tracker*, pero aumentaríamos considerablemente la latencia. Balanceando ambos resultados, es por mucho preferible la posibilidad de pérdida de paquetes (que se traduciría en pequeñísimos saltos del cursos) a una latencia alta (los cursores “irían persiguiendo” a nuestros dedos con un desfase).

La pérdida de paquetes es controlable gracias a los mensajes FSEQ, que siguen un aumento secuencial. Incluso los pequeños artefactos creados por estas pérdidas podrían ser reducibles mediante un sistema de predicción de movimiento, activado cuando dos paquetes TUIO tienen mensajes FSEQ no consecutivos.

6.3.2 *WINDOWS 7*

El último sistema operativo de Microsoft, Windows 7, introduce capacidades multi-touch *built-in*, lo que quiere decir que además de soportar dispositivos multitouch de una forma “artificial” utilizando APIs externos al propio sistema, como *Windows Touch Platform*, internamente gestiona como un elemento más de entrada los toques y gestos sobre un dispositivo de estas características (tratándolo de una forma parecida a un ratón o un teclado).

6.3.2.1 Sistema de Mensajes de Windows

Quizá una de las formas más importantes de comunicación en Windows sean los mensajes. Un programa normal comienza en la función *main()*, va ejecutando líneas de código y eventualmente termina. En Windows el concepto es diferente, ya las aplicaciones responden a eventos, que se conocen como mensajes.

Los mensajes pueden significar muchos eventos, originados por el usuario, el sistema operativo u otros programas. Un evento puede ser un movimiento de ratón, una tecla presionada, o una ventana escalada.

En una aplicación de Windows, el encargado de gestionar estos eventos es el bucle de mensajes (*Message Loop*), que se ejecuta en el thread principal de la aplicación y que procesa tareas en función de dichos mensajes. Ejemplos de éstos pueden ser WM_QUIT, que termina la aplicación, WM_LBUTTONDOWN, que significa que se ha pulsado el botón izquierdo del ratón, o WM_PAINT que implica que la ventana debe ser redibujada.

6.3.2.2 Mensajes Multitouch

En Windows 7 se definen dos nuevos mensajes para el tratamiento interno de dispositivos multitouch: WM_TOUCH y WM_GESTURE.

El primero de ellos, WM_TOUCH, provee de información “en crudo” de los toques que se efectúan. El bucle de proceso de mensajes recibe estos mensajes con un identificador único de toque, por cada una de las pulsaciones del usuario en el dispositivo, en tiempo real. Mediante la función del API de Windows GetTouchInputInfo, se recoge una estructura tipo TOUCHINPUT con todos los datos posibles adquiridos del dispositivo multitouch.

```
typedef struct _TOUCHINPUT {
    LONG        x;
    LONG        y;
    HANDLE       hSource;
    DWORD       dwID;
    DWORD       dwFlags;
    DWORD       dwMask;
    DWORD       dwTime;
    ULONG_PTR   dwExtraInfo;
    DWORD       cxContact;
    DWORD       cyContact;
} TOUCHINPUT, *PTOUCHINPUT;
```

Los componentes más importantes de esta estructura son:

- **x,y**: coordenadas en píxeles físicos del toque.
- **hSource**: identificador del dispositivo proveniente.
- **dwID**: identificador del toque.
- **dwFlags**: marcadores en bit con información diversa, como si el toque es nuevo, se comienza, se suelta, etc.
- **dwTime**: un *timestamp* del evento, para cálculos de tiempo.
- **cxContact, cyContact**: el tamaño del toque, si los soporta el dispositivo, en centésimas de píxel.

Por otra parte, los mensajes de tipo WM_GESTURE gestionan directamente gestos pre-definidos en el sistema operativo. En particular, se reconocen:

- Zoom
- Desplazamiento de uno y dos dedos
- Rotación

- Toque de dos dedos a la vez
- Presión con un dedo y toque

Dado que en este proyecto la capa de aplicación es la encargada de gestionar los gestos, estos mensajes WM_GESTURE no se tendrán en cuenta.

Toda esta información proviene directamente de la documentación oficial de Microsoft del API de Windows (Microsoft Corp., 2009).

6.3.2.3 Adaptación a vlmt

Para adaptar la información proveniente de los mensajes de Windows tipo WM_TOUCH a la librería descrita vlmt, sólo es necesario fijarse en los espacios de coordenadas en los que cada uno trabaja. Esto supone 2 adaptaciones: por un parte, hay que tener en cuenta que los mensajes WM_TOUCH entregan las coordenadas en píxeles reales del escritorio de Windows. Como vlmt trabaja en coordenadas de la parte *renderizable* de la ventana, hay que sumar el desplazamiento de la ventana particular del proceso dentro del escritorio y los marcos que ésta pueda tener.

Por otra parte, vlmt utiliza coordenadas normalizadas entre 0 y 1, así que aquellas procedentes del proceso anterior deben ser divididas entre la resolución de escritorio, tanto en ancho como en alto.

De esta forma, la transformación queda como sigue

$$x' = \frac{x - (pos_{vent_x} + ancho_{marco})}{ancho_{ventana}}$$

$$y' = \frac{y - (pos_{vent_y} + alto_{marco})}{alto_{ventana}}$$

donde la x e y son las coordenadas de WM_TOUCH, x' e y' las transformadas para vlmt (el significado del resto de términos queda definido por su nombre).

Como a todas las ventanas (que lo soliciten) del sistema les llegan los mensajes de toque, sin tener en cuenta si están siendo producidos dentro o fuera de ellas, hay que obviar aquellos que no caen en la ventana de *render*. Esto es muy útil para aplicaciones que no se están ejecutando en pantalla completa. En este caso, si se cumple la condición

$$0 \leq x' \text{ e } y' \leq 1$$

quiere decir que el toque se ha producido en la ventana en la que está funcionando la aplicación. En caso contrario, se descarta el mensaje por producirse el toque fuera de la misma.

Por último, los identificadores de Windows se adaptan a los de vlmt con un simple contador, y el parámetro dt que marca el *timestamp* de los toques se utilizan directamente.

6.3.2.4 Procesos externos y Hooks

En esta capa de adaptación, la pretensión de este proyecto es precisamente poder adaptar el sistema a cualquier aplicación arbitraria. Desgraciadamente, no siempre es posible tener acceso al bucle de gestión de mensajes. Dependiendo de la estructura software de la aplicación, el acceso a los mensajes puede no ser inmediato, o incluso por cuestiones de diseño, puede no interesar acceder a esa parte del código.

Windows soluciona este problema mediante una entidad llamada **hook**, que es un punto del mecanismo de gestión de mensajes donde otro proceso puede instalar una subrutina para monitorizar el tráfico de mensajes y procesar algunos de ellos, antes de que

lleguen al bucle principal. Para poder “colgar” uno de estos *hooks* es imprescindible conocer el identificador único del thread principal del proceso padre. El siguiente código se encarga de hacer la inicialización del hook, y debe ser ejecutado en el mismo proceso que la aplicación padre, para recuperar su ID.

```
bool DispWin7::inicializar() {

    BYTE digitizerStatus = (BYTE)GetSystemMetrics(SM_DIGITIZER);
    if ((digitizerStatus & (0x80 + 0x40)) == 0) //Stack Ready + MultiTouch
    {
        std::cout << "No hay dispositivos MultiTouch" << std::endl;
        return false;
    }

    _maxToques = (BYTE)GetSystemMetrics(SM_MAXIMUMTOUCHES);

    RegisterTouchWindow(_ventana, TWF_WANTPALM);
    _miThis = this;
    _inputs = new TOUCHINPUT[_maxToques];
    _id = 1;
    for(unsigned int i=1; i<=_maxToques; i++) {
        _tablaIDs[i] = 0;
    }

    RECT r;
    GetClientRect(_ventana, &r);
    _anchoVentana = r.right;
    _altoVentana = r.bottom;

    /////////// INSTALAR HOOK PARA TOUCH
    _hook = ::SetWindowsHookEx(
        WH_GETMESSAGE,
        GetMsgProcTouch,
        NULL,
        GetWindowThreadProcessId(_ventana, 0)
    );

    if(_hook==NULL) {
        std::cout << "No se ha podido colgar el hook" << std::endl;
        return false;
    }

    return true;
}
```

Con la function `SetWindowsHookEx` se cuelga el hook, preprocesando todos los mensajes que lleguen a la ventana mediante la función `GetMsgProcTouch()` definida como

```
static LRESULT CALLBACK GetMsgProcTouch(int nCode, WPARAM wParam, LPARAM lParam);
```

Con el diseño anteriormente visto de vlmt, se procesan los toques y se integran en el sistema.

7 CAPA DE APLICACIÓN - PARADIGMAS DE MOVIMIENTO 3D

Este capítulo describe la Capa de Aplicación. Esto es, de qué forma se puede integrar la estructura vista previamente con aplicaciones arbitrarias.

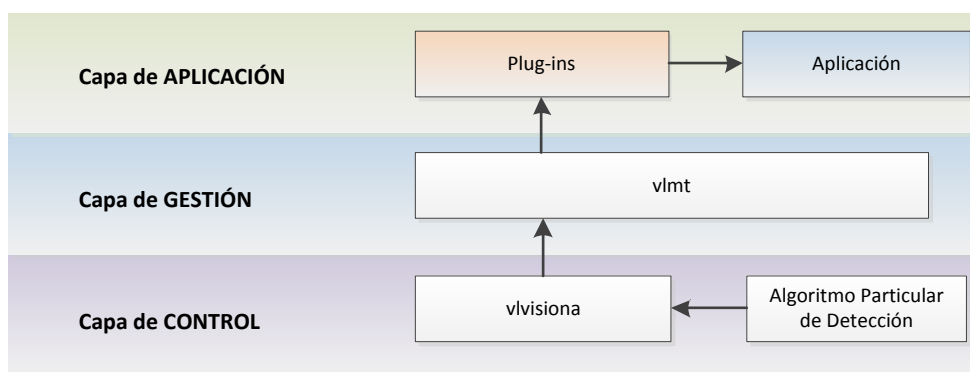


Ilustración 33. Estructura de capas. Capa de Aplicación.

La temática escogida ha sido el uso de estos dispositivos aplicado al mundo de la visualización 3D. Por una parte, en este tipo de aplicaciones se extiende la típica interfaz bidimensional de áreas de interacción (como botones que puedan ser pulsados), a la tercera dimensión, donde hay que interactuar con espacios tridimensionales.

Además, y quizá lo más importante, es el manejo de la cámara virtual que define la vista de la escena. A modo ilustrativo, describamos algunos ejemplos comunes:

- En un juego de tipo *shot'em up* (juegos bélicos de vista subjetiva), con el teclado se puede manejar la posición del personaje, mediante movimientos en sus ejes relativos, y con el ratón el ángulo con el que está mirando.

- En un simulador de vuelo, un joystick define el cabeceo y altura de un avión, tratando de amoldarse a un modelo físico de aviación.
- En una aplicación de mapas, el click y posterior movimiento de ratón “arrastra” el mapa como si el usuario lo agarrase y desplazase.

En este caso, se desarrollaron dos nuevos paradigmas de movimiento: el primero de ellos, PRotaZoom, diseñado para visualización de terreno y arquitectura digital. El segundo, Walk-Touch, para poder dar “paseos virtuales” por los interiores de modelos arquitectónicos.

Ambos fueron implementados e integrados bajo el *framework* del visualizador SANTI.

7.1 DISEÑO DE ESTRATEGIAS DE VUELO EN SANTI

En el capítulo dedicado a fundamentos se presentó ya de este visualizador de terrenos y modelos arquitectónicos. Sin entrar en demasiados detalles del funcionamiento interno de SANTI, diremos que existen entre sus numerosos módulos uno dedicado principalmente al movimiento de cámara y la interacción con dispositivos, llamado **santiOSG**, y otro que contiene el motor fundamental de geometría y textura, llamado simplemente **santi**. Son los que necesitaremos para implementar los paradigmas de vuelo que veremos.

Gracias al correcto diseño orientado a objetos del proyecto SANTI, integrar nuevas funcionalidades suele ser una tarea sencilla. En este caso, todo el control de cámara viene gestionado por una clase denominada **Vuelo**, que maneja toda la información matemática necesaria para situar el punto de vista sobre el espacio 3D.

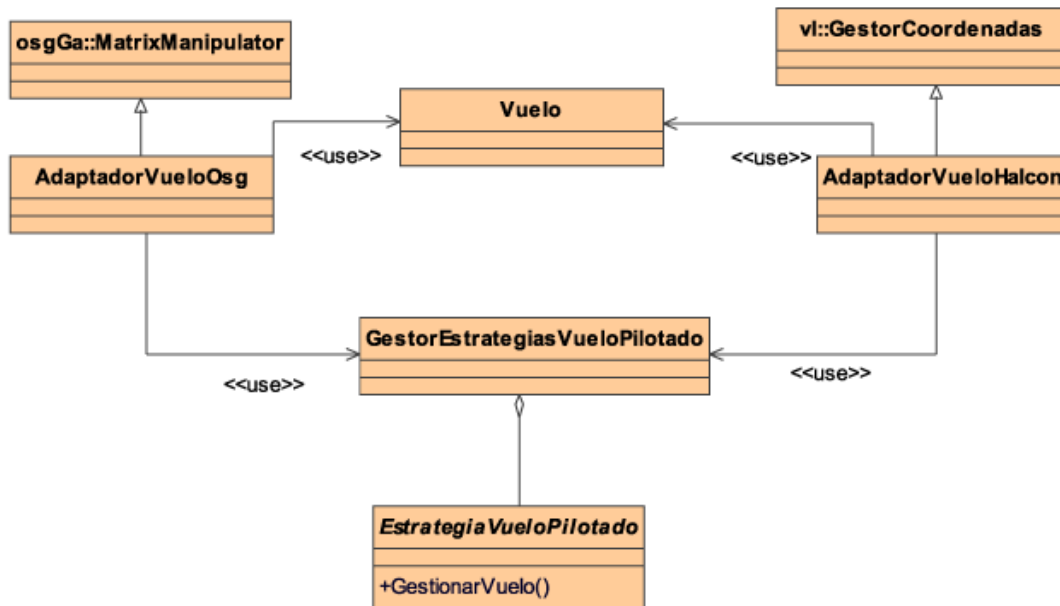


Ilustración 34. Diagrama simplificado de la parte de Vuelo de santiOSG.

La clase Vuelo define un interfaz muy completo y complejo que trabaja con la cámara a bajo nivel. Para gestionar entonces el movimiento, se define la clase *GestorEstrategiasVueloPilotado* y la clase virtual pura *EstrategiaVueloPilotado*. Como se ve en el diagrama de la Ilustración 34. Esta es la forma estándar de definir partes ampliables del código de VidealAB. Se trata de un híbrido entre los patrones de diseño Fábrica y Estrategia, agrupados en una clase tipo Singleton, que solemos denominar *Fábrica de Estrategias*. En el siguiente diagrama (Ilustración 35) se puede observar su estructura genérica.

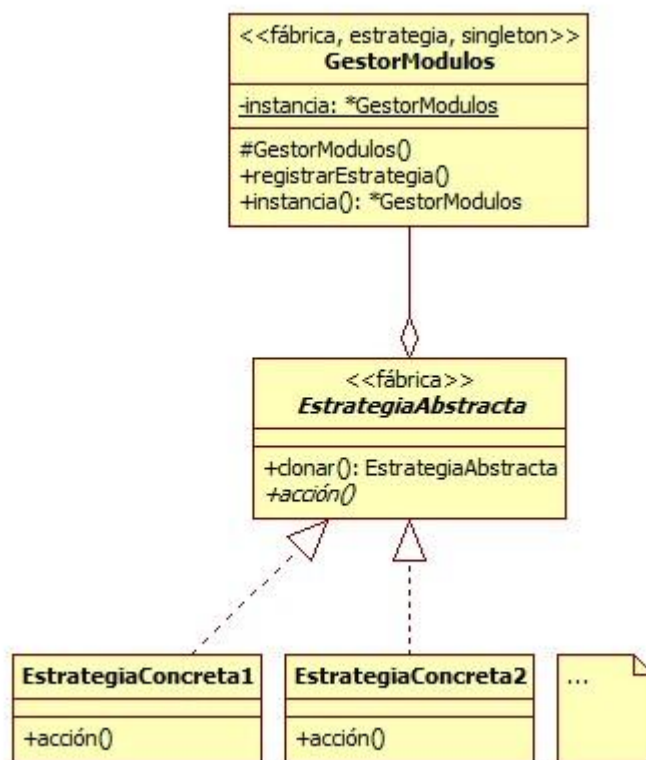


Ilustración 35. Diagrama de las Fábricas de Estrategias (genérico).

Por lo tanto, el desarrollo de los *plugins* de movimiento se reduce a implementar dos nuevas clases, concreciones de *EstrategiaVueloPilotado*, que se conecten con el sistema multitouch descrito anteriormente y que integren los paradigmas de movimiento que se exponen a continuación.

La clase *EstrategiaVueloPilotado* puede verse en el siguiente diagrama (Ilustración 36).

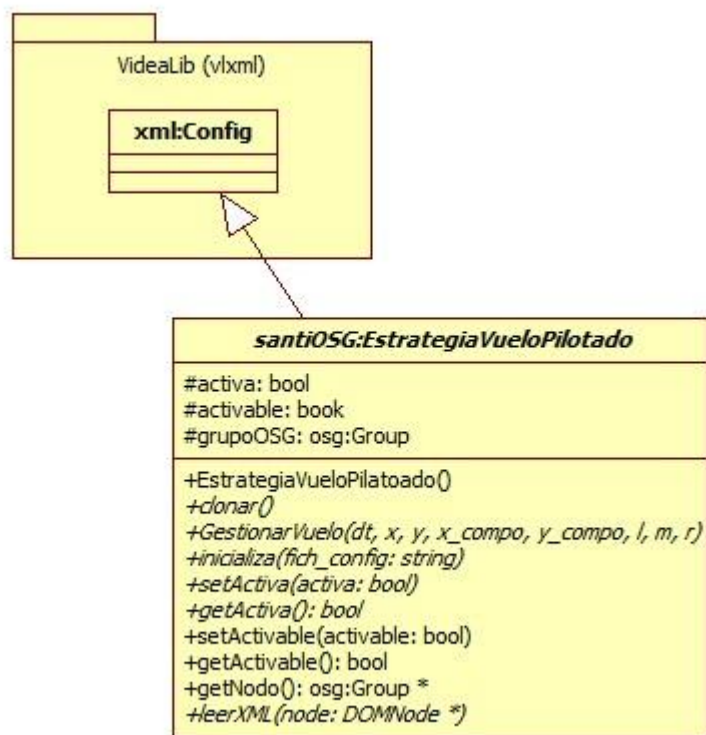


Ilustración 36. Detalle de la clase EstrategiaVueloPilotoado.

Deben entonces implementar el interfaz público de esta clase virtual, que fundamentalmente consiste en los siguientes métodos:

- **gestionarVuelo:** es la más importante de ellas. Se llamará periódicamente durante el tiempo de ejecución, según una serie de *timers* internos que tienen que ver con el *framerate*. Lleva como parámetros todos los datos del dispositivo básico, que es el ratón, y la diferencia de tiempo con el *frame* anterior.
- **inicializa:** la función de inicialización de la estrategia concreta, a la cual se le puede pasar un fichero de configuración.

- **setActiva** y **getActiva**: funciones para especificar si esa es la estrategia que se está utilizando en ese momento.
- **leerXML**: implementa el interfaz de `vl::Config`. Esta es una clase de soporte dentro de la librería genérica `VideaLib` poder leer fácilmente configuraciones de ficheros tipo XML. De aquí sacara en la inicialización todos los parámetros de funcionamiento.

El siguiente diagrama muestra algunos módulos ya existentes de distintos modos de movimiento que SANTI utiliza comúnmente.

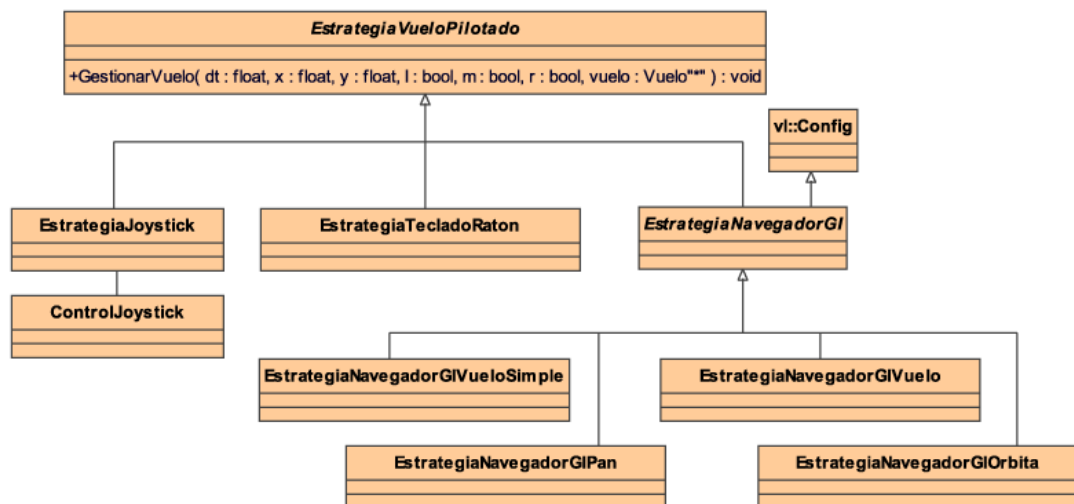


Ilustración 37. Ejemplos de clases concretas de Estrategias de Vuelo.

7.2 PROTAZOOM

PRotaZoom es un paradigma de movimiento de cámara tridimensional, pensado para la navegación sobre terrenos y sobre exteriores de modelos arquitectónicos. Su nombre

viene de Pitch-ROTate-ZOOM (cabeceo, rotación y zoom), haciendo referencia a la combinación de movimientos que permite.

Está inspirado en ciertas operaciones típicas al trabajar con escritorios virtuales de dispositivos multitáctiles (Ilustración 38), y el entrenamiento es cada vez menor ya que estos gestos se extienden a otros visualizadores de terreno como *Google Earth* (Google, 2011).

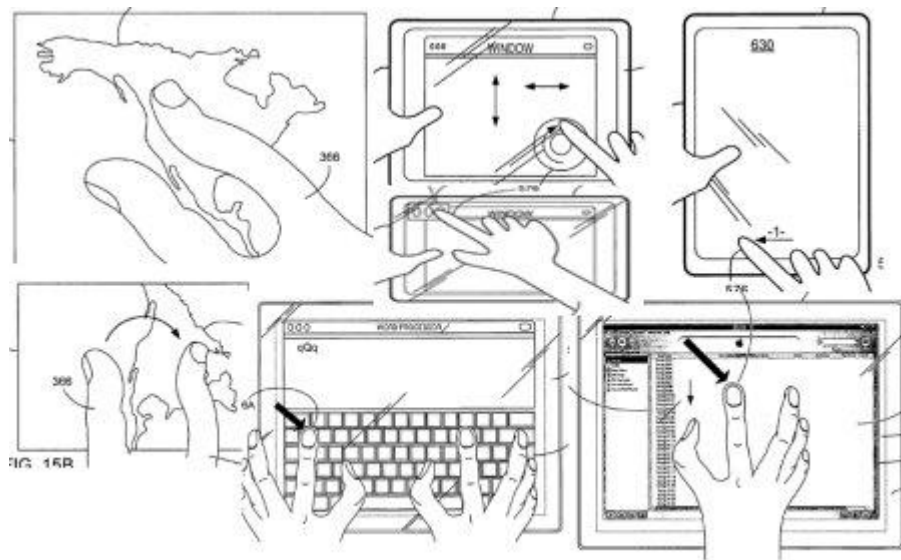


Ilustración 38. Gestos típicos sobre un escritorio virtual.

Sobrevolando un terreno o una “maqueta virtual” de un edificio (a cualquiera de ambas cosas denominaremos en adelante “**modelo**”), hay ciertas operaciones que se vuelven naturales. Por ejemplo, si el usuario toca en un punto del modelo y lo arrastra, se espera un comportamiento natural de “arrastrar” el modelo debajo de su dedo a la nueva posición, como si estuviera “tirando” de él.

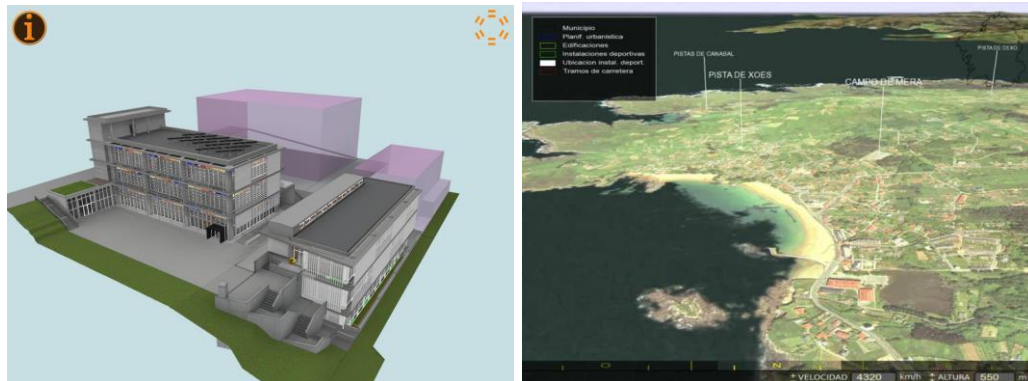


Ilustración 39. Ejemplos de SANTI, visualizando arquitectura y territorio.

Si toca con dos dedos y los aleja entre sí, la metáfora radica en separar esos dos puntos, como trayendo el modelo hacia el usuario (como si cogiéramos una foto y la acercásemos a los ojos: los puntos de sujeción, a nuestra vista, se separarían, y la foto “se haría más grande”). Podemos invertir este caso para el efecto de alejar el modelo. Y a rotar dos dedos entre sí, se espera que ese movimiento de rotación se traslade al modelo como si se tratase de girar un vinilo en un tocadiscos. Quizá el menos natural, aunque extendido, sea el cabeceo moviendo dos dedos en perpendicular arriba y abajo.

Habitualmente todos estos movimientos se realizan de manera individual, es decir, el usuario comienza el gesto, que el programa interpreta como uno de ellos y lo ejecuta en función de los movimientos del usuario. **La novedad que aporta este sistema, es que los gestos de giro, zoom y cabeceo se realizan al mismo tiempo, interpretando matemáticamente las posiciones y velocidades relativas de los dos toques a la vez.**

Esto supone un incremento más que notable de la agilidad y suavidad con la que el usuario puede examinar el modelo. Por la contra, el algoritmo que interpreta el gesto es sumamente complejo, y tiene varios parámetros (umbrales normalmente) que deben ser bien ajustados para su correcto funcionamiento (ajustados sólo durante el desarrollo, una vez nada más).



Ilustración 40. Gestos básicos del PRotaZoom.

7.2.1 PUNTO DE CONTROL

Todos los movimientos anteriormente descritos necesitan un **punto básico de pivote**. Sobre él se realizarán las translaciones, rotaciones, zooms, y demás operaciones. Podemos pensar en este punto como el **centro de atención del usuario**, que se recalcula cada vez que hay un nuevo toque. Le llamaremos **punto de control**. Es importante recalcar que el punto de control se encuentra en el espacio del modelo tridimensional, es decir, es un punto con tres coordenadas en el mundo 3D en el que trabajemos (no en el espacio bidimensional de la pantalla).

Para calcularlo, debe hacerse una operación bien conocida en los gráficos por computador, llamada **Ray Casting** o “trazado de rayos” (Ilustración 41). Sobre la información de la sección anterior, dados los datos que definen tanto la posición (p) y rotación (α , en ángulos de Euler) de la cámara, podemos calcular el Plano de Proyección de la imagen

renderizada. Si posicionamos la coordenada del toque (c) en dicho plano, y “lanzamos un rayo” desde la posición de la cámara p que pase por el punto c , éste puede intersectar con el modelo en el punto I , perteneciente a la superficie del mismo. La intersección también calcula la normal de esa superficie en el punto I .

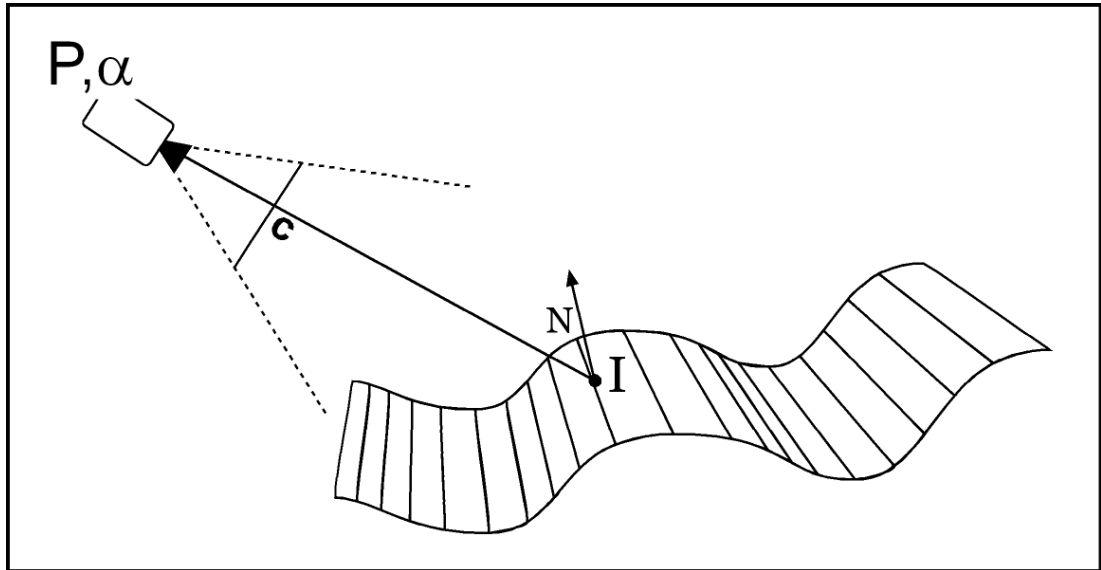


Ilustración 41. Ray Casting para el cálculo del punto de control.

En caso de existir tal intersección, I será el punto de control. Si el rayo pasase de largo, o intersecase con una parte del modelo que no interese, se descartaría ese toque.

El algoritmo de intersección entre una línea y un conjunto de polígonos es más que conocido, y se puede consultar en la bibliografía (Glassner, 1989).

7.2.2 UN TOQUE: ARRASTRE

Cuando el usuario toca la superficie con un solo dedo, el punto de control se calcula con el algoritmo anteriormente descrito, lanzando el rayo desde la coordenada del toque.

En caso de intersección, a partir de ese momento el objetivo será que ese punto de control se mantenga siempre debajo del dedo, sea cual sea la posición relativa del modelo y de la cámara.

Dicho movimiento, denominado *pam*, se consigue haciendo únicamente modificaciones en la posición de la cámara; su orientación no varía. Este problema, por simple que parezca, es sumamente complejo, teniendo en cuenta las posibles irregularidades del modelo.

Existen dos configuraciones diferentes del algoritmo (ver Anexo I) según se esté visualizando terreno (vuelo aéreo) o un modelo arquitectónico.

Para el vuelo aéreo, es importante mantener la altura de la cámara fija. Ya que ésta viene determinada por la coordenada z del vector posición, sólo hará falta modificar la x y la y . Su desplazamiento vendrá determinado por simular el cálculo de un nuevo punto de control en cada *frame*, y aplicar la misma translación que hay entre el auténtico punto del control y el simulado a la cámara.

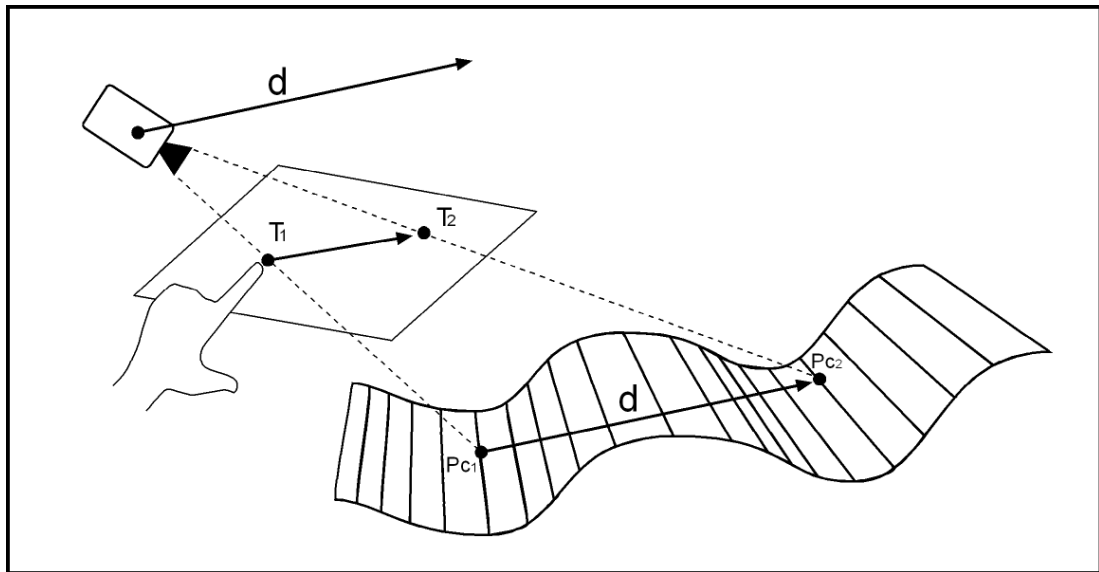


Ilustración 42. Arrastre en visualización de terreno.

En la Ilustración 42 se ve representado este caso. T_1 y T_2 representan el arrastre de un toque T , en dos *frames* consecutivos. PC_1 y PC_2 son sus respectivos puntos de control, y d es la distancia de desplazamiento en el espacio tridimensional, que se aplica a la cámara.

En el caso de visualizar modelos digitales arquitectónicos, el problema es ligeramente diferente, ya que el “arrastre” tiene normalmente como objeto examinar partes concretas del modelo, y no recorrer extensiones de terreno como en el caso anterior. Entonces, en este caso el algoritmo es similar, pero la translación aplicada a la cámara, en vez de estar bloqueada en un plano fijo (una altura determinada en la visualización de terreno), está bloqueada en un plano definido por la posición inicial de la cámara y el vector normal de la superficie en el punto primero de control. De esta forma se consigue que todo movimiento de “arrastre” sea siempre paralelo a la superficie arrastrada.

Una forma amena y tremendamente visual de entender este movimiento es el nombre con el que se ha bautizado coloquialmente: modo “spiderman”. La Ilustración 43 lo representa, siguiendo la misma nomenclatura que la anterior, y añadiendo N como la normal del primer punto de control.

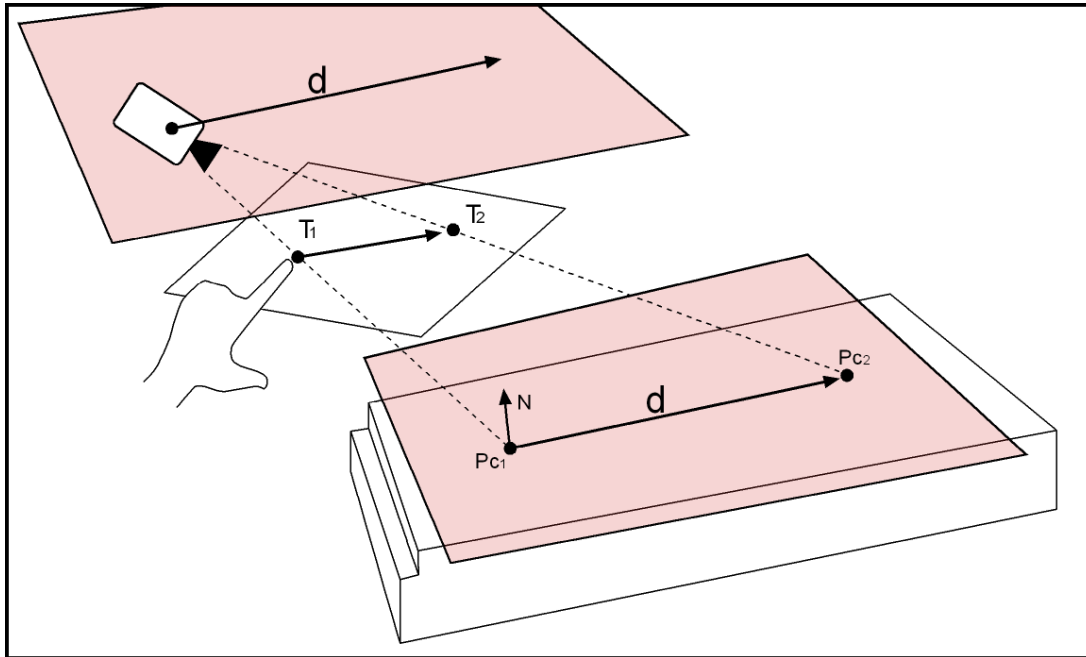


Ilustración 43. Arrastre en la visualización de modelos arquitectónicos.

7.2.3 DOS TOQUES: PROTAZOOM

Como se ha descrito anteriormente, el PRotaZoom aúna el movimiento de Pitch, Rotación y Zoom en un solo gesto, combinación de tres más.

El punto de control, en este caso, se calcula en la coordenada de la superficie resultante de la media aritmética entre las coordenadas de ambos toques. Para tratar de explicar el movimiento, se sugiere al lector que piense, dentro del espacio tridimensional en el

que se alberga el modelo, en una semiesfera imaginaria centrada en el punto de control y de radio la distancia de la cámara al mismo. La cámara siempre se moverá en la superficie de dicha semiesfera.

Ya se ha considerado que el punto de control debe ser tomado como el centro de interés del usuario. Entonces, la orientación de la cámara será calculada tal que ésta siempre mire hacia el punto de control.

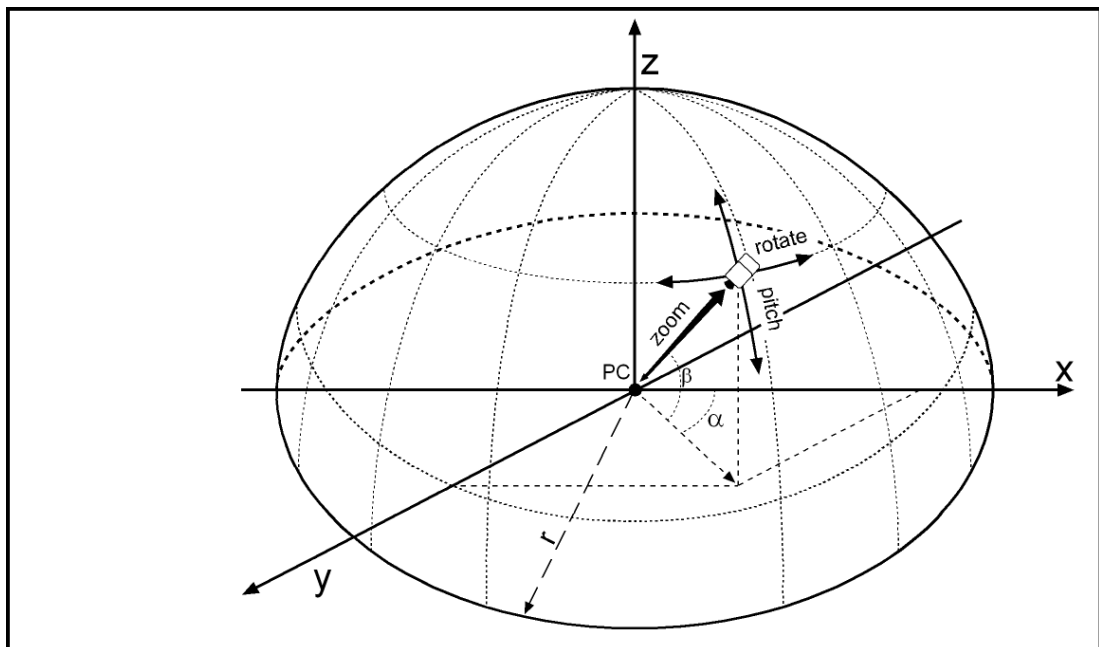


Ilustración 44. Semiesfera definida mediante meridianos y paralelos.

Dados los meridianos y paralelos de una esfera, como la de la Ilustración 44, la variación de la posición de la cámara será como sigue:

- Para la **rotación**, se moverá sobre su meridiano.
- Para el cabeceo o **pitch**, se moverá sobre su paralelo.
- Para el **zoom**, simplemente se aumentará o disminuirá el radio de la semiesfera.

Veamos ahora como aplicar estos tres movimientos al mismo tiempo según las sucesivas relaciones geométrica entre los toques, utilizando el esquema de la Ilustración 45. Donde T_1 y T_2 son los dos toques iniciales, y T'_1 y T'_2 son los mismos toques, un instante de tiempo después. PC y PC' son sus respectivos puntos de control.

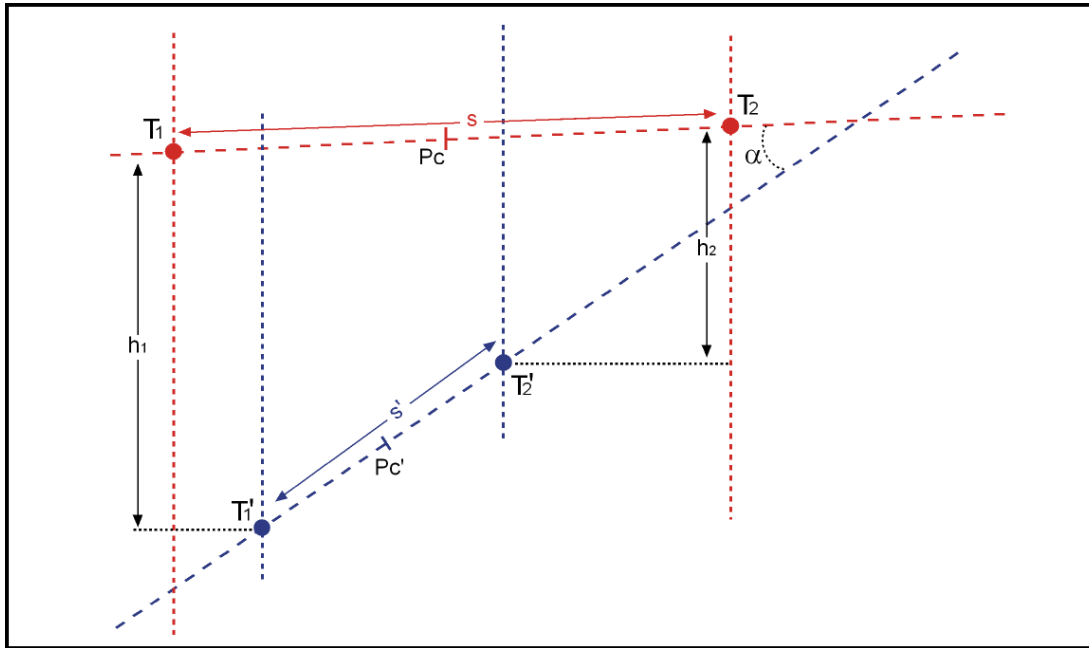


Ilustración 45. Esquema de relaciones geométricas entre dos toques para el PRotaZoom.

Trataremos de dar una explicación muy somera, sin incidir en las múltiples excepciones y umbrales, para introducir el algoritmo. La variación del ángulo α formado entre la recta que une T_1 y T_2 y la que une T'_1 y T'_2 define la rotación. La diferencia de las distancias entre dichos puntos, s y s' , define el zoom. Por último, la media de las diferencias de alturas en las verticales de los toques, h_1 y h_2 definen el pitch. Esquemáticamente, tenemos

$$\begin{cases} \mathbf{zoom} \equiv s - s' \\ \mathbf{rotación} \equiv \Delta\alpha \\ \mathbf{pitch} \equiv \frac{h_1 + h_2}{2} \end{cases}$$

7.2.4 INERCIA

Como se ha comentado, el paradigma de vuelo PRotaZoom trata de maximizar la agilidad y suavidad con la que el usuario se mueve por escena 3D.

A las personas nos cuesta asimilar los cortes bruscos en cualquier aspecto de nuestra vida, dada la naturaleza lineal del universo físico en el que nos desenvolvemos. En este caso, si cuando el usuario terminase una interacción (un “arrastre”, una rotación, etc.) el movimiento se detuviese “de sopetón”, sería demasiado brusco, y antinatural. Es por esto que al terminar de tocar la superficie debe existir, aunque sea de una forma muy sutil, una pequeña **inercia**.

En cualquier momento del movimiento, la cámara posee dos vectores de velocidad (y, por tanto, otros dos de aceleración): uno para su posición y otro para su posición. Sin inercia, al detener la interacción, estos dos vectores caen instantáneamente a cero. La inercia consiste en forzar una caída logarítmica de la gráfica de velocidad cuando la interacción cesa, con una duración muy pequeña (por ejemplo, un segundo). Es logarítmica porque de esta forma la modificación de la posición es mínima, y sin embargo el efecto de suavidad que proporciona es muy confortable.

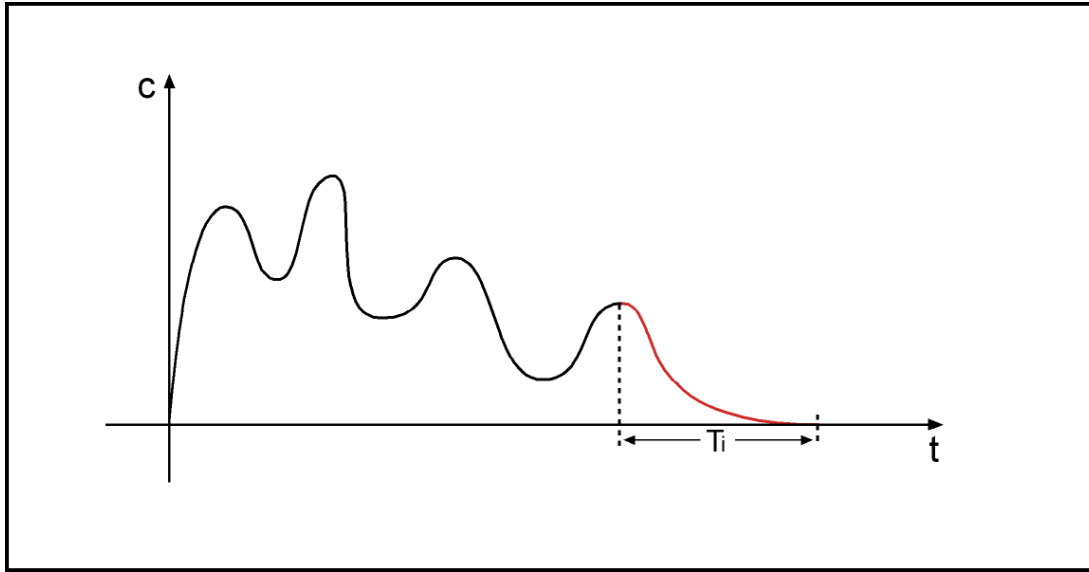


Ilustración 46. Inercia generada para terminar el movimiento suavemente. Las ordenadas representan el tiempo y las abscisas cualquiera de las coordenadas de la cámara.

La imagen anterior (Ilustración 46) muestra la generación artificial de un movimiento suave de decaimiento al cero, para terminar naturalmente cualquiera de los movimientos. El Tiempo T_i máximo es configurado de antemano, y como se decía, no debería ser muy grande. Matemáticamente, es muy sencillo definir esta inercia como una función recursiva hasta llegar a un umbral de “quietud”.

$$\begin{cases} p = \frac{(p_{nueva} - p)}{2} + \frac{velocidad}{2} & \text{si } p_{nueva} - p < umbral \\ 0 & \text{otro caso} \end{cases}$$

7.2.5 EXTENSIONES DE TRES TOQUES

Este paradigma de movimiento ha sido utilizado en diversos proyectos, en los que en alguna ocasión se ha tenido que extender la interacción incluso a los tres toques. En

estos casos, por simplicidad, no se tenían en cuenta las posiciones relativas de los toques, sino simplemente el hecho de que estuvieran presentes. Para calcular sus movimientos, se hizo considerando una única coordenada correspondiente al centro de masas de los toques.

Algunos ejemplos en los que se ha extendido a tres toques son:

- **Vista libre de cámara:** es decir, modificar el ángulo h (heading) y p (pitch), siguiendo el eje horizontal y vertical de la superficie multitáctil, respectivamente.
- **Sección libre del modelo:** aplicar un plano de corte que seccione el modelo en una parte visible y otra no visible, según el eje en que se mueva el centro de masas de los toques.

7.3 TOUCH-WALK

El segundo paradigma de movimiento para la visualización 3D se bautizó como **Touch-Walk**, y tiene como objeto poder **recorrer los interiores de modelos arquitectónicos digitales como si se caminase sobre ellos**.



Ilustración 47. Visualización "a pié" de espacios arquitectónicos dentro del SANTI.

Esta cuestión plantea fundamentalmente dos problemas. Por una parte, y siguiendo las directrices de los interfaces naturales, el usuario espera poder hacer movimientos similares a los que ejecuta al pasear físicamente por un espacio arquitectónico. Estos movimientos se ciñen normalmente a caminar, en la dirección que el propio cuerpo marque, y, al mismo tiempo, dirigir la mirada hacia lugares de interés, tanto como el cuello permita. Resolver esta dualidad de movimientos será el primero de los problemas.

El segundo consiste en la “transitabilidad” del modelo tridimensional. Estos modelos son simples representaciones visuales, que no físicas, del espacio. Al caminar, debe tenerse en cuenta que no se pueden traspasar muros, que una escalara puede subirse, etc. La creación de la información de soporte y el algoritmo necesario será el segundo de los problemas.

En la siguiente imagen (Ilustración 48) pueden verse los controles que se han definido para este paradigma de movimiento: pulsar y arrastrar, para dirigir el cuerpo en la dirección deseada, e introducir un segundo toque para, al mismo tiempo, girar libremente la vista.



Ilustración 48. Gestos básicos del Walk-Touch.

En las siguientes subsecciones se hará referencia a los términos de la siguiente imagen (), donde h es la altura del punto de vista con respecto al suelo, P_v es el punto de vista, V_{c1} es el vector de colisiones 1, V_{c2} es el vector de colisiones 2, d es el vector de avance (o la dirección) con ángulo α , y β es el ángulo de visión (euleriano).

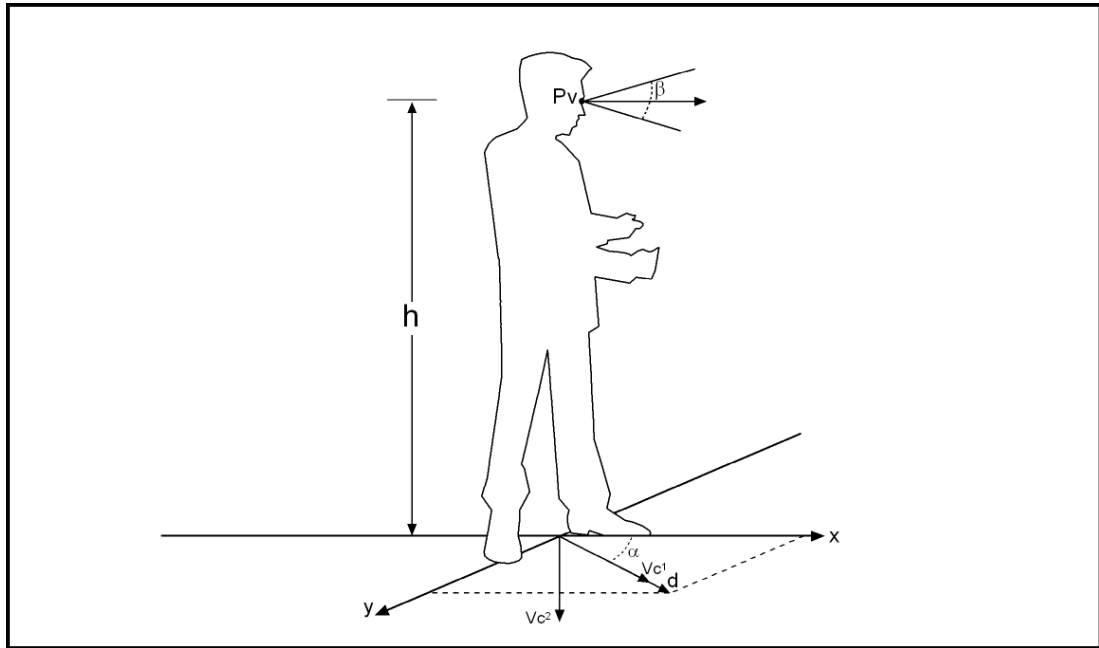


Ilustración 49. Entidades matemáticas que intervienen en el paradigma Walk-Touch.

7.3.1 PUNTO DE VISTA Y MOVIMIENTO

Ya que tratamos de simular la visión humana caminando por un espacio, el punto de vista (cámara) P_v se encontrará a una altura determinada a partir del suelo, parámetro predefinido h .

Cuando el usuario toca la superficie multitáctil, define un centro de coordenadas bidimensionales, y cualquier arrastre del dedo que ejecute creará un vector, con un módulo

y ángulo determinado. El módulo significará la velocidad del avance, y el ángulo su dirección. La nueva posición de la cámara en un paso de tiempo vendrá entonces definida por

$$p(x, y, z) = p + d$$

Al introducir un segundo toque, se creará en ese lugar un segundo centro de coordenadas bidimensionales, a partir del cual el arrastre de este toque influirá directamente en la vista de la cámara, aunque sólo en sus parámetros h (*heading*) y p (*pitch*). El parámetro r (*roll*) no se tiene en cuenta en este caso al considerarse irrelevante. El valor del vector resultante (v) al arrastrar el toque en el eje de la ordenadas modulará el *heading*, y el de las abscisas, el *pitch*. Entonces,

$$h = |v|\cos(\beta)$$

$$p = |v|\sin(\beta)$$

Pero es posible que al mismo tiempo se esté moviendo el “cuerpo virtual” del usuario, con el gesto inicialmente descrito, que está modificando a su vez el valor del *heading* de la cámara. Con lo cual, el valor final del valor h de *heading* queda como

$$h = h + \alpha + |v|\cos(\beta)$$

7.3.2 COLISIONES

Para solucionar la cuestión de simular el paseo del usuario por un suelo físico, toparse con paredes o subir escaleras, se utiliza la técnica de las **colisiones** (Eberly, 2003), ampliamente conocida en el campo de los videojuegos.

Esta consiste en gestionar dos vectores V_{c1} y V_{c2} , denominados vectores de colisiones. El primero de ellos tiene su origen en el punto de vista (P_v) y apunta exactamente hacia abajo ($-z$), con un módulo ligeramente superior a la altura del P_v respecto al suelo (h). El segundo apunta siempre en la dirección del avance.

Podemos pensar en estos vectores como en la vara de un hombre invidente. En los cálculos de cada *frame*, antes de actualizar la posición de la cámara, se comprueba si estos vectores intersecan con la geometría del modelo 3D.

V_{c1} debe colisionar continuamente con el suelo. Si no lo hiciera, se generaría un vector de aceleración hacia abajo (con módulo la gravedad) que actualizase la posición hasta que V_{c1} volviese a colisionar. De esta forma simularíamos las caídas. Para poder subir “rampas”, debe medirse la cantidad de vector que penetra en la geometría. Si ésta desciende, quiere decir que el usuario “baja” por una rampa, y debemos actualizar la altura del punto de vista. En caso contrario el usuario “sube”.

En cuanto a V_{c2} , su estado normal es no intersecar con ninguna geometría. En el caso en que lo hiciera, deberá bloquearse el avance en el sentido de la marcha, y sólo permitirse en las direcciones oblicuas, para que el usuario pueda salir de dicho estado.

El algoritmo de intersección es similar al descrito anteriormente de *Ray Casting*. El problema es que, en este caso, deben realizarse una gran cantidad de comprobaciones por frame (en cada polígono del modelo, dos veces, una por vector de colisión). Para solventar este problema de CPU, se utilizan los denominados **modelos de colisión**.

Un modelo de colisión consiste en una geometría que representa las superficies sólidas del modelo. Es muchísimo más simple que este, y no se usa en absoluto para visualiza-

ción, sino sólo para realizar los cálculos que tengan que ver con colisiones. La siguiente imagen muestra un modelo de colisión (Ilustración 50).

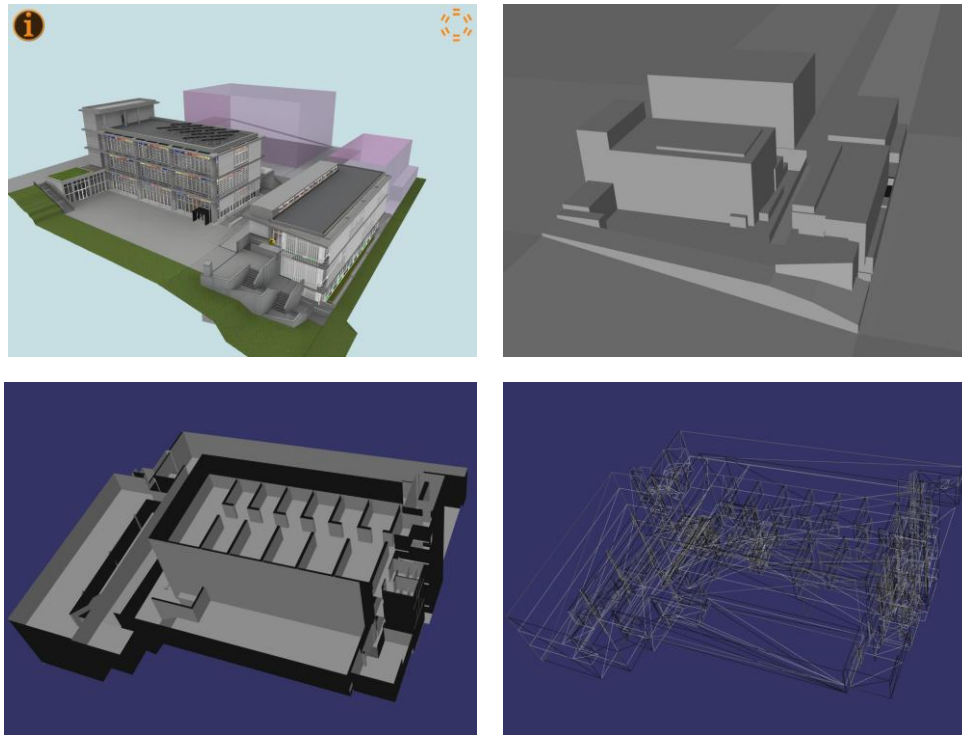


Ilustración 50. Modelos de colisión.

7.3.3 TELETRANSPORTADORES

En ciertas ocasiones se ha experimentado que algunos modelos de arquitectura no tienen unas características demasiado adecuadas para este modo de movimiento, ya bien sea porque no existen accesos tipo rampas o escaleras a algunas zonas, son demasiado grandes para recorrer a pie, etc.

Para ayudar al usuario en la “exploración”, se ha llegado a utilizar unas “ayudas”, en forma de iconos tridimensionales en el espacio, que “teletransportan” el punto de vista

a otra ubicación predefinida. Los denominamos teletransportadores, y en la siguiente imagen (Ilustración 51) pueden verse un par de ejemplos.



Ilustración 51. Ejemplos de teletransportadores.

8 DESARROLLO DEL PROTOTIPO HARDWARE

8.1 FUNDAMENTOS

Multitud de dispositivos multitouch se basan en soluciones de tipo óptico para conseguir su propósito. La escalabilidad, el bajo coste y la facilidad de construcción son algunos de los motivos de su popularidad. Existen muchas técnicas de tipo “light sensing”, como veremos a continuación, para conseguir detectar los toques del usuario sobre una mesa interactiva. Todos ellos consisten en un sensor óptico, normalmente una cámara, una fuente de luz infrarroja, y una forma de mostrar imágenes como una pantalla LCD o un proyector.

Antes de comentar las distintas tecnologías ópticas, y particularmente la utilizada en este proyecto, es necesaria una pequeña introducción hacia la fuente de emisión y la recepción del elemento más importante en estas tecnologías, la luz infrarroja.

8.1.1 FUENTES DE LUZ INFRARROJA

La luz infrarroja (IR en adelante) es una porción del espectro lumínico que se ubica justamente antes de la luz visible por el ojo humano. Su rango de longitud de onda es mayor que la luz visible, y determina una pequeña banda conocida como NIR (Near InfraRed, Cerca del Infrarrojo) que típicamente se consideran entre 700 nm y 1000 nm (nanómetros). La mayor parte de los sensores de las cámaras digitales son excitables en este rango, y a menudo vienen armados con filtros que lo eliminan, para no interferir

con la luz visible. La simple sustitución de este filtro por otro que elimine la luz visible convierte cualquier cámara digital en una cámara exclusivamente de luz infrarroja.

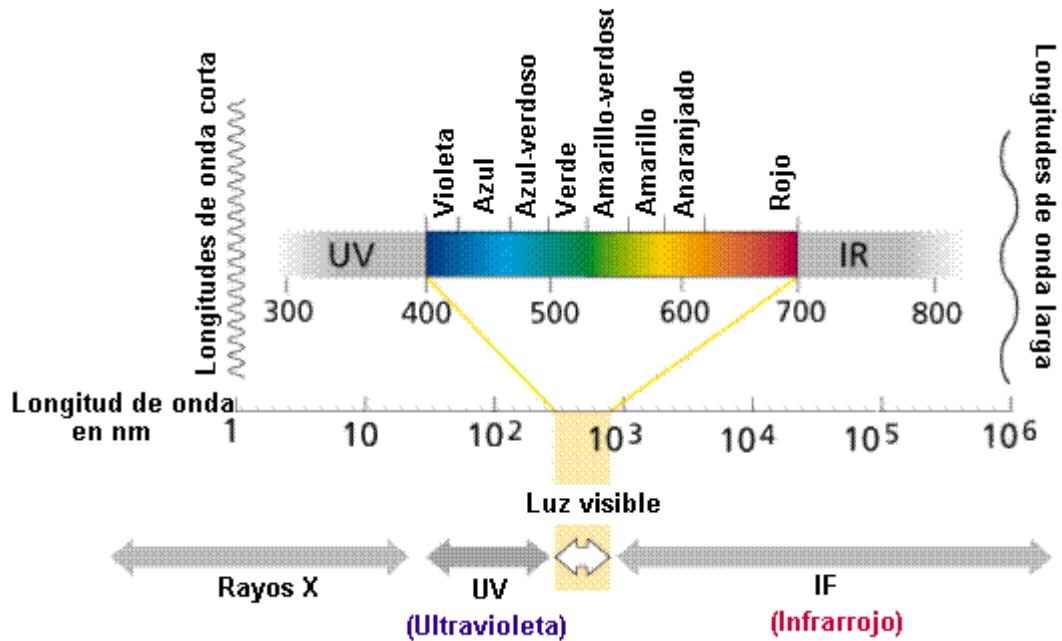


Ilustración 52. Espectro visible e IR de la luz.

En las superficies multitouch se utiliza este tipo de luz para poder separar lo concerniente a la imagen de la pantalla, con la que el usuario interactuará (luz visible), y aquella utilizada únicamente con fines de detección de toques. Aunque ambos haces de luz se mezclen en el mismo dispositivo, es relativamente sencillo trabajar con ellas independientemente utilizando únicamente la banda correspondiente a cada una.

El mayor problema de utilizar esta banda para la detección es evitar la polución lumínica infrarroja, que puede crear falsos positivos a la hora de la detección, y que no suele ser fácil de percibir a priori. Las fuentes de calor, el sol, las lámparas de tungsteno y algunos otros dispositivos producen gran cantidad de luz infrarroja, que debe ser valo-

rada a la hora de construir y ubicar superficies interactivas de este tipo. La industria define unas longitudes de onda estándar que evitan algunas de las bandas más comunes, como por ejemplo la de la luz emitida por el sol (sobre 940nm).

Las fuentes de IR más comunes utilizadas son los diodos emisores de luz, o LEDs. Su bajo consumo, alta luminosidad, direccionalidad del haz emitido y larga duración los convierte en los candidatos idóneos para estos dispositivos. Dependiendo de la técnica a implementar, conviene utilizar diodos sueltos o tiras de diodos, que facilitan enormemente su escalabilidad.

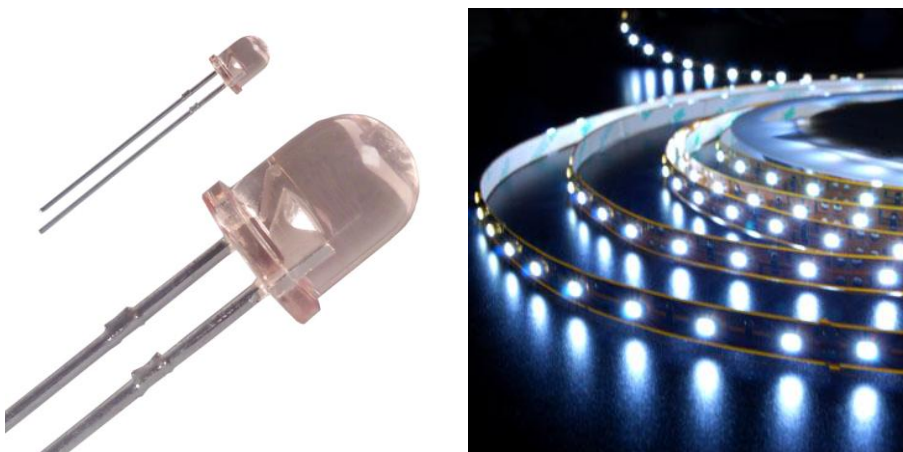


Ilustración 53. LEDs de luz infrarroja, tanto individuales como en tiras ya ensambladas.

A la hora de escoger los adecuados, se tendrán en cuenta tres parámetros que resultan de vital importancia para el correcto funcionamiento del sistema:

- **Longitud de onda:** debe caer en la banda infrarroja, pero además en la que los sensores de las cámaras son más excitables, entre 780 y 940 nanómetros.

- **Intensidad de radiación:** cuanta más, mejor. Con los LEDs actuales de alta luminosidad, es bastante sencillo encontrar a partir de 100mW.
- **Ángulo de emisión:** este parámetro se hace crucial en algunas de las técnicas que comentaremos, y concretamente en la FTIR, que es la utilizada en este proyecto. Significa la amplitud del haz emitido desde el eje central del diodo, y no debe ser menor a unos 50 grados para aprovechar correctamente el fenómeno óptico propuesta por esta técnica.

8.1.2 CÁMARAS

Las cámaras generales de consumo están preparadas para captar luz en el espectro visible. Pero esto no es culpa del sensor (normalmente CCD) de la misma que, como se aprecia en la Ilustración 54, en realidad abarca muchas más frecuencias. Entre la lente de la cámara y el sensor se suele colocar un **filtro** que bloquea aquellas frecuencias que están fuera del espectro visible, para evitar artefactos en la imagen.

Ya que el fenómeno FITR descrito trabaja con frecuencias en la banda de infrarrojos, por debajo del espectro, debe utilizarse una cámara especial equipada con un filtro que trabaje en dicha banda, o bien sustituir el que trae por defecto.

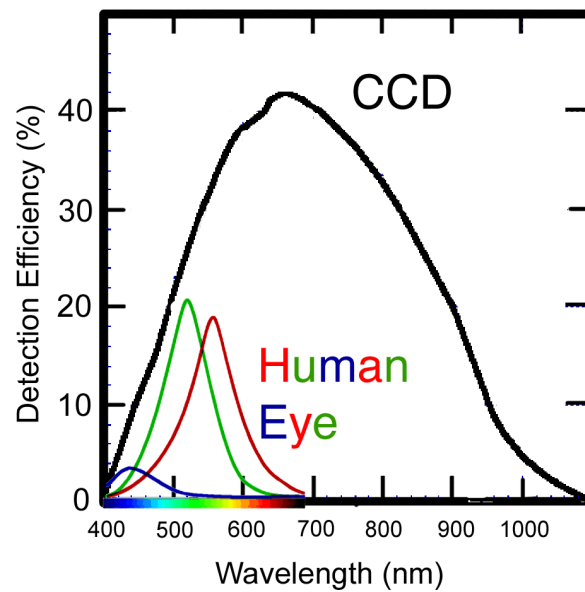


Ilustración 54. Comparación de las frecuencias captadas por el ojo humano y un sensor CCD en el espectro de luz.

8.1.3 REFLEXIÓN INTERNA TOTAL FRUSTRADA (FTIR)

Esta técnica, FTIR (*Frustrated Total Internal Reflection*) se ha convertido en una de las más utilizadas para el tracking de toques en la construcción de superficies interactivas multitáctiles de carácter óptico. Desarrollada por Jeff Han en el 2005 (Han J. , 2005), se basa en un fenómeno óptico conocido como Reflexión Interna Total que poseen algunos materiales cuando la luz entra en ellos desde otro con un mayor índice de reflexión y con un ángulo de incidencia que supere un umbral dependiente de ambos materiales, conocido como el ángulo crítico, y calculable mediante la Ley de Snell (Ilustración 55).

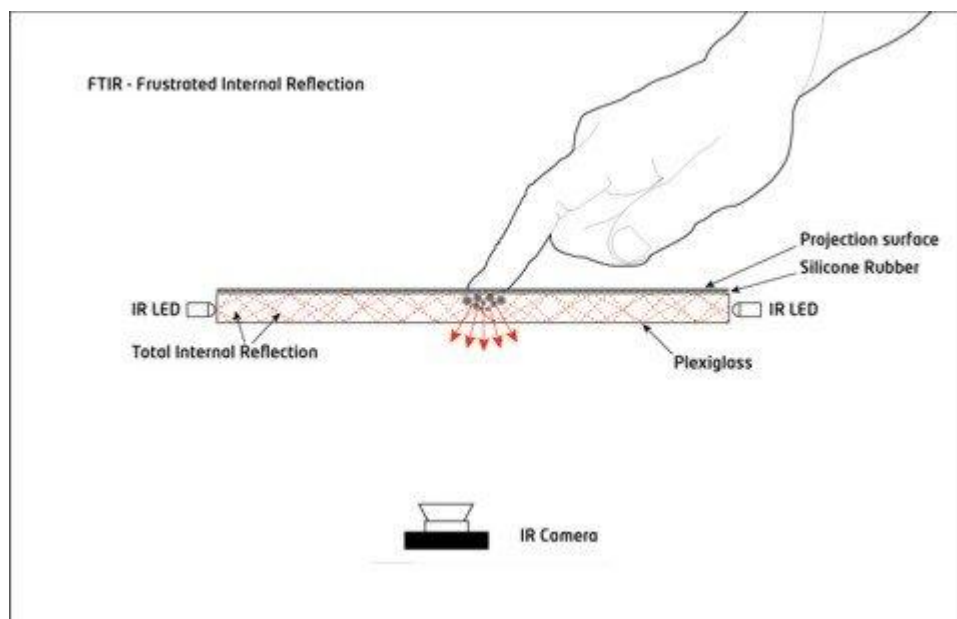


Ilustración 55. Funcionamiento del principio FTIR.

Cuando esto sucede, la refracción en el material se cancela, y toda la luz sale reflejada. El método de Han utiliza este fenómeno utilizando un panel acrílico (normalmente metacrilato) inundado con luz infrarroja, que será reflejada perpendicularmente al posar sobre ella cualquier objeto, como por ejemplo, un dedo. Al entrar éste en contacto con la superficie, se dice que la refracción de rayos infrarrojos dentro del acrílico se frustra, ya que pueden pasar a través del material del objeto, en este caso pie. Esta luz “frustrada” es recogida por una cámara web, armada con los filtros necesarios para descartar todo ancho de banda que no se corresponda con la de la luz infrarroja escogida. Esto resulta en una imagen completamente negra excepto por zonas de luz, que denominamos blobs, y que representan las zonas de contacto, que pueden ser reconocidas y seguidas en tiempo real por un software de visión artificial apropiado (Ilustración 56).



Ilustración 56. Captura de cámara IR, mostrando los blobs generados por FTIR.

8.2 DISEÑO Y CONSTRUCCIÓN

8.2.1 *CONCEPTO*

Inicialmente se desarrolló un boceto conceptual de la mesa. Ésta debía cumplir una serie de especificaciones en sus medidas para cuadrar todos los elementos técnicos que la componen, además de tener en cuenta cuestiones como la facilidad de acceso para su mantenimiento, ventilación, etc. Es de justicia comentar aquí que el diseño y construcción del mueble, propiamente dicho, no fue elaborado por el autor, sino por miembros del laboratorio VidealAB.

La siguiente imagen muestra una recreación tridimensional del concepto.

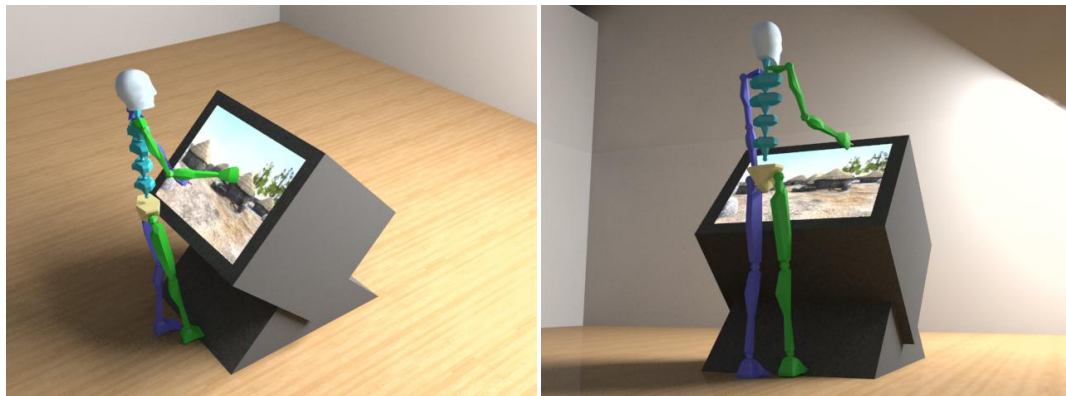


Ilustración 57. Recreación virtual del concepto del prototipo.

8.2.2 CAMPOS VISUALES: PROYECCIÓN Y CAPTURA

Es importante tener en cuenta los campos de proyección y de visión del proyector y la cámara respectivamente. El campo de visión se define matemáticamente como una pirámide truncada; cuanto más distancia haya hasta la cúspide imaginaria, mayor será la base (ver Ilustración 58). El ángulo con el que se lanza la proyección (o ángulo de visión de la cámara) se denomina FOV (Field Of View), y se tiene en cuenta tanto el vertical como el horizontal.

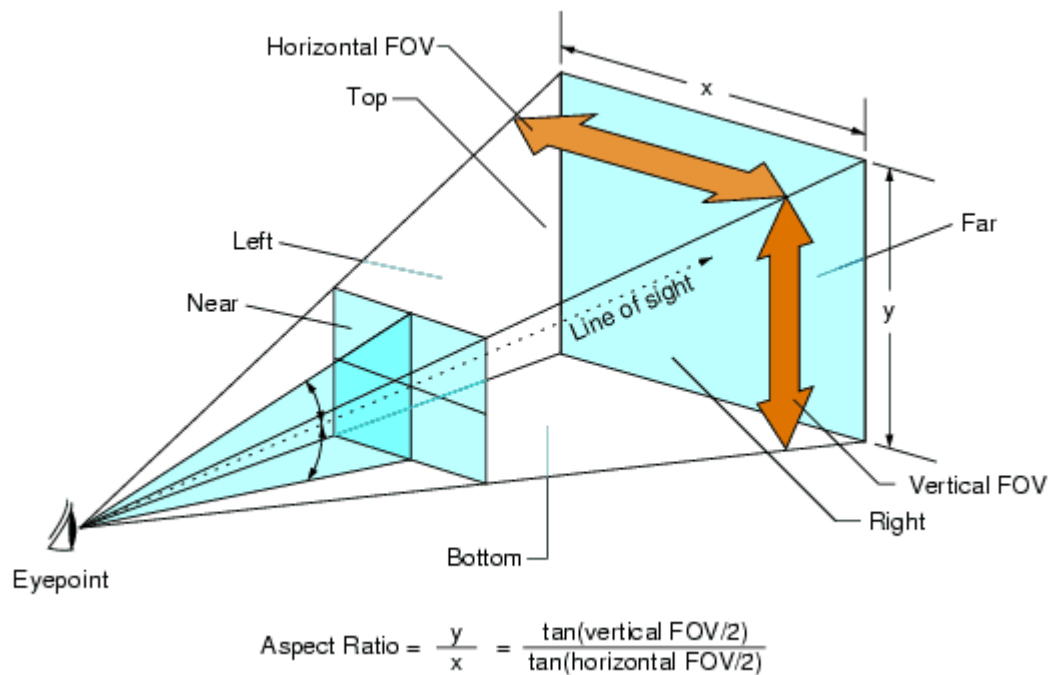


Ilustración 58. Medidas en una proyección.

En cuanto a la cámara, se escogió una con lentes intercambiables, para poder solventar este problema; si bien es cierto que cuanto más angular sea la lente, y más campo coja, mayor es la deformación de la imagen y más difícil su calibración. Dado su ángulo de visión horizontal de aproximadamente $55,5^\circ$, la distancia mínima a colocar de la superficie se calcula como

$$d = \frac{80}{\tan^{-1} 55,5^\circ} \approx 76 \text{ cm}$$

Sobre el proyector, dadas las proporciones del prototipo, se escogió uno especial con lente gran angular, de tipo “short-through”. Estos proyectores tienen la característica de que pueden conseguir enfocar la imagen en distancias muy cercanas. En este caso, su FOV horizontal es de unos 59° , entonces

$$d = \frac{80}{\tan^{-1} 59^\circ} \approx 71 \text{ cm}$$

En la siguiente imagen se pueden ver las medidas del prototipo, teniendo en cuenta los cálculos anteriores.

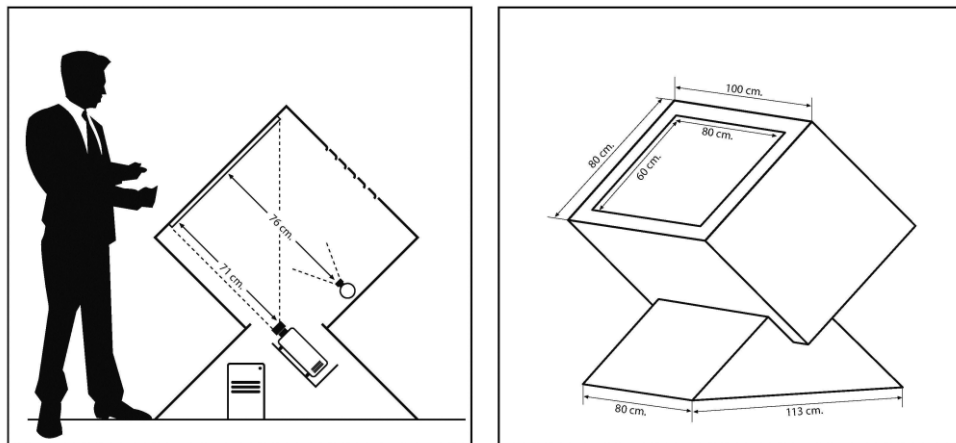


Ilustración 59. Diseños de medidas del prototipo.



Ilustración 60. Detalles del prototipo: proyector y cámara.

8.2.3 LUZ INFRARROJA

En el prototipo se decidió hacer la inserción de luz infrarroja utilizando una tira de leds (más conocida como *IR LED Ribbon*), por su sencillez de manipulación, a pesar de ser algo más costoso económicamente de una red de LEDs ovales individuales.

Es importante destacar que el canto del acrílico, en este caso metacrilato, debe ser pulido para conseguir insertar la mayor cantidad de luz emitida por los diodos.

A continuación se muestran algunas imágenes al respecto (Ilustración 61).



Ilustración 61. Tiras de LEDs IR en el canto del acrílico pulido.

8.2.4 SUPERFICIE

La pantalla se plantea como un rectángulo de proporciones 4:3 y medidas 80 x 60 cm. Anteriormente se describió el principio de FTIR, muy útil para construir superficies multitáctiles, pero no suficiente. Las mesas interactivas, además de la tecnología necesaria para captar los toques, necesitan una forma de mostrar imágenes, sobre las que el usuario interactuará.

La construcción del prototipo utiliza imágenes retroproyectadas sobre un acrílico. Pero el metacrilato normalmente es transparente, y aunque no lo fuera, no permitiría suficiente definición como para utilizarlo de superficie de proyección. Es necesario añadir una segunda capa a la superficie, que denominamos “**Capa Difusora**”, por encima del acrílico. Esta capa puede ser de diversos materiales, y aunque existen soluciones comerciales óptimas para cumplir este cometido, también lo es que son excesivamente caros. Sin embargo, existen algunos materiales mucho más económicos que pueden cumplir la misma función sin perder prácticamente ninguna definición ni calidad de tacto, como veremos posteriormente en la descripción del desarrollo del prototipo.

Solucionando la cuestión de la proyección, surge un nuevo problema, quizá el más complejo de resolver, y es que el fenómeno del FTIR es muy dependiente de la textura, la “viscosidad” del material que ocluye la luz, en este caso las puntas de los dedos. Una piel grasa provocará un reflejo muchísimo más intenso que una piel seca, y por tanto, unos blobs mucho más sencillos de reconocer por el software de visión artificial. La introducción de la capa difusora separa mediante un plástico los dedos del acrílico, haciendo desaparecer por completo el efecto del FTIR. Se hace necesario entonces introducir una nueva capa entre el acrílico y la capa de proyección, que reproduzca o in-

cluso potencie el tacto viscoso de los dedos, y que denominaremos “**Capa de Adaptación**”. Como decíamos, a nivel constructivo esta capa constituye el mayor de los retos en la construcción de la mesa utilizando FTIR, y se suelen utilizar materiales como goma de silicona catalizada, latex líquido, etc.

Por último, y en caso de requerirlo, se añade una cuarta capa encima de todas las anteriores, denominada “**Capa Protectora**” con una doble función: por una parte, protege las anteriores capas en el caso de que estas estén fabricadas con materiales frágiles o excesivamente sensibles. Por otra parte, será la que proporcione el tacto final a la superficie interactiva.

En resumidas cuentas, la superficie se compone de cuatro capas, contando el metacrilato, en el orden mostrado en la Ilustración 62.

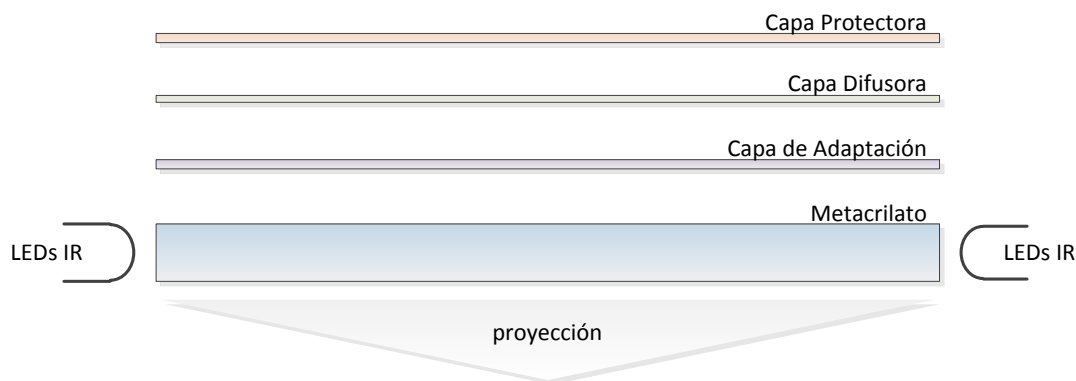


Ilustración 62. Capas de la superficie del prototipo.

El proceso de fabricación sigue el “método Tinkerman” (Tinkerman, 2009), más conocido como método “*rolled-on*”, ya que la capa de adaptación es “pintada” directamente sobre la capa difusora.

Para la capa de adaptación, debe crearse un producto químico a partir de componentes que pueden encontrarse en cualquier droguería, como son un disolvente sintético y silicona neutra. El proceso debe ser tremendamente cuidadoso, porque cualquier imperfección en la mezcla contrastará mucho en la pantalla proyectada.

El proceso entonces sigue el siguiente esquema:

1. Extender el papel utilizado como capa difusora con cierta tensión
2. Hacer entre 4 y 6 capas.
 - a. Mezclar en un tarro limpio la una parte de silicona por dos de disolvente.
 - b. Remover paulatinamente hasta obtener una mezcla homogénea y sin burbujas.
 - c. Volcar la mezcla sobre el papel y extender con un rodillo de poro fino.
 - d. Dejar secar un mínimo de 5 o 6 horas en un entorno **con humedad**.
3. Dejar secar todo durante al menos un día.
4. Dependiendo de la rugosidad de la capa resultante, se ha visto en ocasiones de utilidad esparcir de forma muy somera cantidades ínfimas de algún tipo de polvo fino, como magnesio, talco o harina.
5. Cortar el resultante (capa adaptadora + capa de difusión) a la medida del metacrilato.
6. Posar lentamente sobre el metacrilato, intentando no hacer movimientos en las capas una vez colocada.
7. Forrar con la última capa, la protectora.

Se dejan en la siguiente imagen () una selección de fotografías del proceso.



Ilustración 63. Proceso de fabricación de las capas de la superficie del prototipo.

Sólo resta remarcar que se llevaron a cabo unas 10 pruebas distintas hasta dar con el proceso correcto. Sobre todo fue costoso dar con la mezcla antes indicada, debido a la multitud de tipos de silicona y disolventes que existen en el mercado, y de la gran posibilidad de proporciones.

8.3 MATERIALES Y COSTES

A continuación se presenta una tabla a modo de resumen con los materiales principales utilizados para la elaboración del prototipo y su precio PVP.

Superficie	Plancha de metacrilato	80,00 €
	2 x Silicona ácida	6,50 €
	Disolvente sintético	5,50 €
	Lámina papel arquitectura	3,50 €
	Lámina transparente adhesiva	4,00 €
	Herramientas	20,00 €
	Tira de LEDs IR 890nm (3m)	180,00 €
	Transformador LEDs	20,00 €
Proyección	Proyector Benq MP772ST	600,00 €
Captura	Webcam PS3Eye	30,00 €
	Set de lentes	20,00 €
	Filtros IR	10,00 €
PC	Ordenador de consumo	600,00 €
Mueble	5 x Planchas DM	70,00 €
	Pintura	45,00 €
	Rejillas ventilación	6,00 €
	Material diverso	60,00 €
TOTAL		1.760,50 €

Como puede verse en la tabla anterior, el coste total en materiales es de aproximadamente 1.760€. Conociendo bien el proceso, un técnico no especializado puede construir un prototipo en aproximadamente 2 semanas. Un operario contratado a tal efecto, al ser un trabajo individual, puede cobrar aproximadamente 12€ por hora, así que, de tener que contratar mano de obra para hacerlo, costaría unos 900€.

8.4 INTEGRACIÓN Y RESULTADO

En la Ilustración 64 pueden verse dos fotografías del aspecto exterior del prototipo ya finalizado y perfectamente funcional.

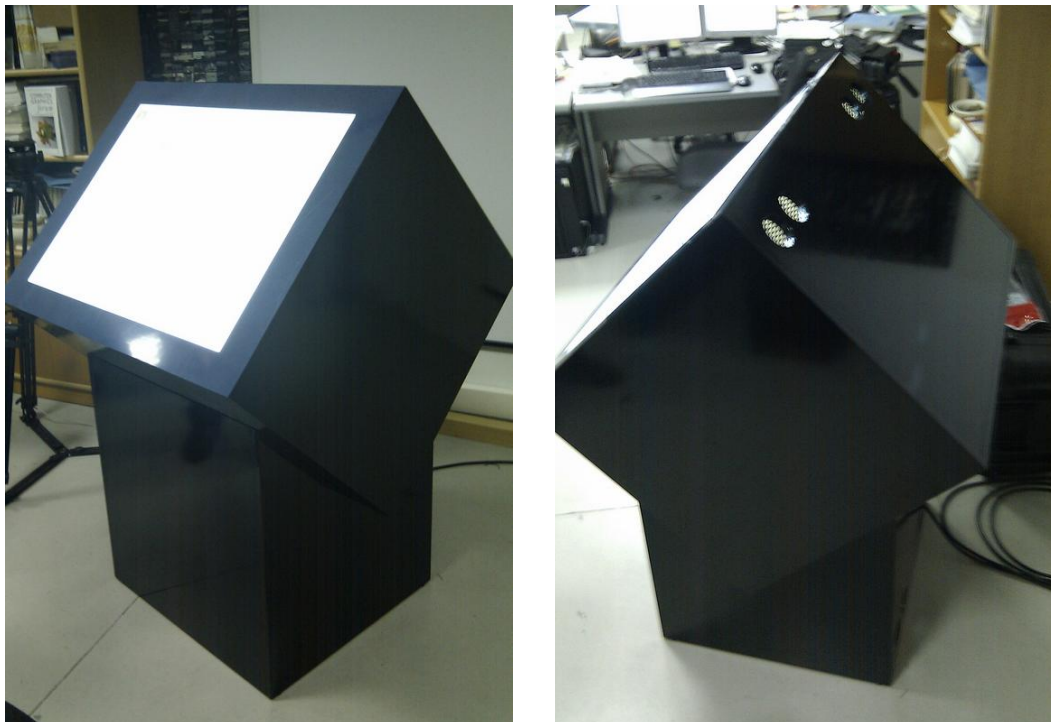


Ilustración 64. Fotografías del prototipo finalizado.

Para su funcionamiento se utiliza el sistema software completo visto en los anteriores capítulos, utilizando CCV como servidor TUIO, vlmr con el dispositivo tipo TUIO y múltiples aplicaciones adaptadas a la interacción multitouch.

Este prototipo ha servido de base para un producto comercial, cuyos usos se describen en el próximo capítulo.

9 CASOS DE ESTUDIO

El sistema presentado en este proyecto ha sido utilizado ya en varios proyectos reales, de carácter tanto permanente como temporal.

Se ha considerado de gran interés mencionarlos en este proyecto, primero por demostrar fehacientemente la efectividad del mismo, y segundo, porque cada uno de ellos utiliza una configuración del sistema completamente diferente. Sirva pues esta sección de casos de uso reales como estudio de la adaptabilidad de cada una de las capas.

9.1 LA FUENTE ENCANTADA

Este proyecto fue propuesto como parte de un concurso público, que consistía en realizar una instalación de carácter museístico, tecnológica e interactiva, en la exposición Galicia Dixital de Santiago de Compostela, para la Xunta de Galicia.

Dadas las características del entorno donde se debía ubicar, en el Monasterio de San Martín Pinario, completamente hecho de piedra, se ideó el concepto de una “fuente mágica” en la que múltiples usuarios podrían interactuar con una serie de escenas relacionadas con la tradición más mística de Galicia.

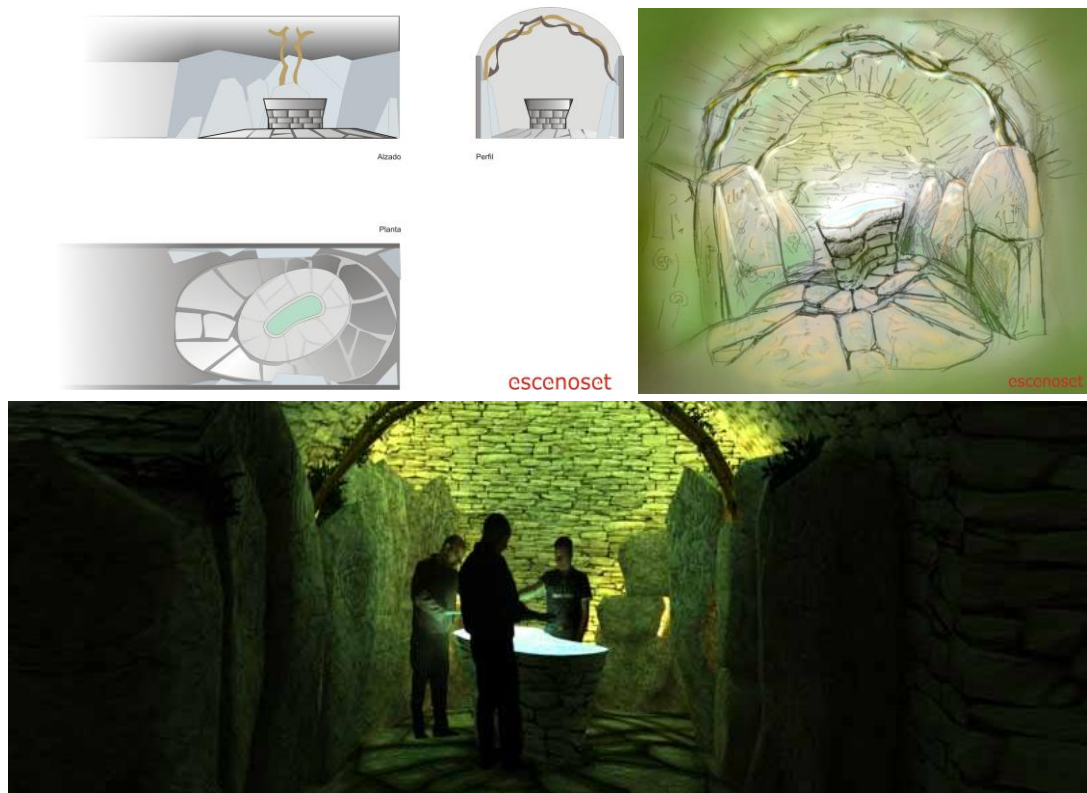


Ilustración 65. Bocetos iniciales y prueba de concepto 3D.

A nivel técnico, el proyecto planteó dos grandes dificultades:

- La retroproyección de la imagen con dos proyectores realizando “blending” en tiempo real, en una forma de pantalla completamente irregular.
- La interacción, donde múltiples usuarios podrían manejar los contenidos con sus manos al mismo tiempo, sin siquiera necesitar tocar la mesa.

El primer punto queda fuera del alcance de este proyecto. Pese a ello, para el interés del lector se hace referencia a él en la bibliografía.

El segundo punto se resolvió utilizando parte del sistema descrito en este proyecto. En particular, la **Capa de Control**, con la librería **vlvisiona** y un **algoritmo reconocedor de manos**, descrito en su correspondiente capítulo.



Ilustración 66. Fotos durante el desarrollo y montaje de La Fuente Encantada.

La instalación lleva en este momento más de dos años en explotación, con varias decenas de usuarios diarios. Durante este tiempo, apenas ha requerido mantenimiento, y aún ahora los visitantes del museo pueden disfrutar de ella.



Ilustración 67. Fotos de la instalación final de La Fuente Encantada.

9.2 AYUNTAMIENTO DE LA CORUÑA: PRESENTACIONES PIM Y FEIL 2010

El Excmo. Ayuntamiento de La Coruña mostró su interés a principios del año 2010 por esta tecnología, y por el sistema desarrollado en el laboratorio VidealAB.

El propio alcalde en ese momento, D. Javier Losada quiso una instalación fija en su despacho en la que se pudiera actualizar en todo momento la vista tridimensional de La Coruña con capas GIS de los proyectos que en ese momento se llevaban a cabo.

Además, las presentaciones públicas tanto del Plan de Inversión Municipal (PIM 2010) y del Fondo de Estatal de Inversión Local (FEIL 2010) se llevaron a cabo utilizando el sistema desarrollado en este proyecto. El alcalde recibió unas nociones básicas de su funcionamiento, que haciendo honor a las premisas de mínimo entrenamiento de los interfaces naturales, fueron muy breves. Para ello, una empresa de terceros construyó una versión más comercial del prototipo, y VidealAB desarrolló una aplicación basada en SANTI y en el sistema multitouch personalizada para el Ayuntamiento. En este

software se utilizan prácticamente todo lo desarrollado en este proyecto, a excepción de la librería vlvisiona (sustituída por CCV) y el paradigma de Touch-Walk.



Ilustración 68. El Alcalde D. Javier Losada presentando los planes municipales en el sistema.



Ilustración 69. Versión comercial del prototipo desarrollado.

9.3 VISUALIZACIÓN 3D DEL NUEVO EDIFICIO CITIC (UDC)

El nuevo edificio de la UDC, el CITIC (Centro de Investigación en Tecnoloxías da Información e as Comunicacions) contiene en su recepción una instalación permanente con el sistema descrito en este proyecto. La aplicación muestra un modelo arquitectónico digital del edificio, con “puntos calientes” de información actualizables.



Ilustración 70. Instalación permanente en el CITIC del sistema.

En este caso, la aplicación utiliza todo el sistema aquí descrito, a excepción de *visiona*, ya que el prototipo funciona con *CCV*. Cabe destacar el uso de ambos paradigmas de movimiento, aplicados esta vez a la visualización arquitectónica. El *PRotaZoom*, configurado en su modo de “arrastre en paralelo”, es ideal para examinar elementos de la fachada. Además, mediante unos enlaces táctiles, se puede entrar en el edificio y caminar virtualmente por él. La siguiente imagen () muestra la pantalla de ayuda, donde se enseña al usuario los posibles gestos de los que dispone.

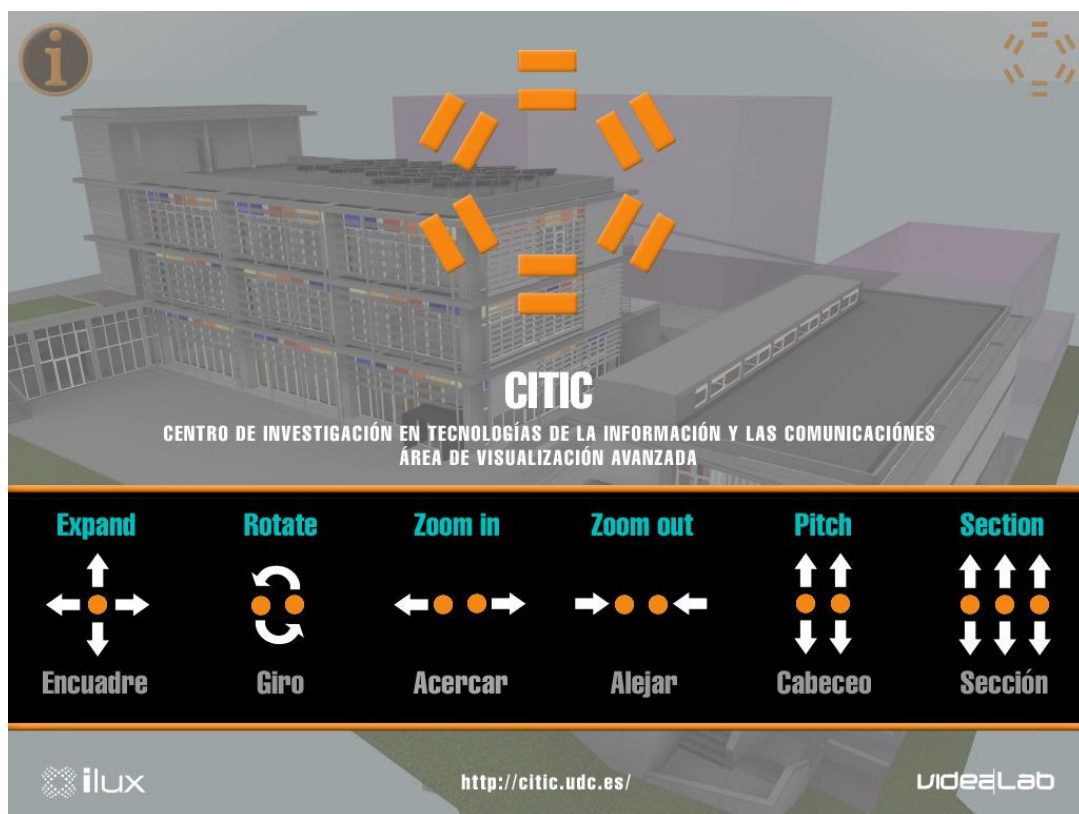


Ilustración 71. Pantalla de ayuda de la aplicación de visualización del CITIC.

9.4 VISUALIZACIÓN 3D INTERACTIVA DE LA CATEDRAL DE SANTIAGO DE COMPOSTELA

En este último caso de estudio se muestra una aplicación realizada con motivo del año Xacobeo 2010 para la Xunta de Galicia. Se trata de la visualización de un modelo arquitectónico altamente fiel de la Catedral santiaguesa.

Esta aplicación añade como novedad un sistema táctil de seccionamiento en ejes arbitrarios, integrado gracias a la flexibilidad del sistema presentado en este proyecto.

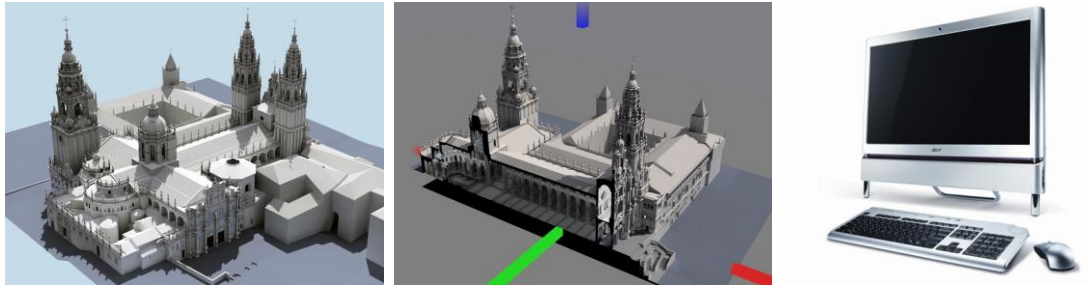


Ilustración 72. Visualización 3D de la Catedral de Santiago, sobre un dispositivo multitáctil de Windows 7.

Este caso es interesante de remarcar, ya que no se utiliza el protocolo TUIO como en los demás. Es un ejemplo de uso de otro dispositivo diferente, en este caso uno virtual manejado por Windows7, en una pantalla modelo ACER Aspire Z5610. Al conectarla a la plataforma Windows, ésta la detecta como un dispositivo táctil, y desde ese momento se puede utilizar con vlmt. Gracias al diseño mostrado de esta librería, el cambio de dispositivo entre el prototipo y cualquiera de estas pantallas es inmediato.

Es además un orgullo indicar que, en el momento en que el autor escribe estas líneas, este proyecto particular se halla nominado a los premios internacionales de arquitectura digital **CG Architect Awads 2011**, en la categoría de *Best Real-Time Application* (Mejor aplicación en tiempo real).

10 CONCLUSIONES Y LÍNEAS FUTURAS

10.1 CONCLUSIONES

En este documento se ha presentado todo el proceso seguido para la obtención de un sistema multitouch completo, extensible y eficiente, aplicándolo además a la visualización 3D mediante el desarrollo de dos nuevos paradigmas de movimiento. Para ello se han seguido los siguientes pasos:

- Gestación de la idea original de conseguir un sistema multitouch aplicado a la visualización 3D.
- Estudio del problema en general, y de sus fundamentos.
- Revisión de las posibles soluciones al problema existentes.
- Elección de la metodología a seguir y planificación de tiempos.
- Análisis del sistema desde un punto de vista global, y diseño de las capas de abstracción.
- Estudio, diseño, e implementación de cada una de las capas.
- Fundamentación matemática e implementación de los paradigmas de movimiento 3D sobre dispositivos multitouch.
- Elaboración del prototipo hardware e integración con el software.
- Exposición de los resultados, en entornos reales de explotación.

En el capítulo introductorio se expresó la intención de cumplir unos ciertos objetivos. A continuación se discute la resolución de cada uno de ellos.

- **Flexibilidad:** el sistema presentado soporta cualquier dispositivo multitouch, presente o futuro, con sólo hacer la implementación correspondiente de una clase heredada.
- **Eficiencia:** las pruebas de los algoritmos *multithread* demuestran una enorme eficiencia en el *stream* de datos, consumiendo menos de un 1% de tiempo de la CPU en una máquina estándar.
- **Simplicidad:** el interfaz para acceder a la librería de gestión vlmt consta únicamente de dos funciones, con una única estructura de datos.
- **Modularidad:** el diseño presentado en capas hace que cualquiera de las partes que lo componen sea perfectamente sustituible, y de hecho, se demuestra con dos ejemplos: un caso que solamente utiliza la capa de control y parte de la de gestión (la Fuente Encantada) y otro donde se utiliza el sistema completo, pero sustituyendo la capa de control por un software de terceros, CCV (el prototipo).
- **Multiplataforma:** todo el sistema ha sido desarrollado utilizando código estándar y librerías de soporte multiplataforma. El sistema ha sido testeado tanto en Windows como en Linux.
- **Robustez:** el sistema ha sido utilizado ya en numerosos casos reales, tanto presentaciones puntuales como instalaciones fijas, donde, en algunos casos, lleva más de un año funcionando sin incidencia alguna.
- **Bajo coste:** el prototipo desarrollado puede construirse entero, incluido el PC que lo controla con un presupuesto menor a 2.000€. Dada las características del dispositivo que se obtiene y los precios de otros productos similares, se le puede atribuir sin problema la etiqueta de “bajo coste”.

10.2 LÍNEAS FUTURAS

Todo parece indicar que la fiebre de las superficies interactivas no ha hecho más que empezar. No ha pasado tanto tiempo desde que películas como *Minority Report* nos dejaban boquiabiertos con interfaces futuristas controlados en pantallas holográficas; y sin embargo, a nadie le extraña hoy que saquemos nuestro móvil del bolsillo y amplie-mos una fotografía con los dedos.

No ha de pasar mucho tiempo para que veamos barras de bar que proyecten la cámara donde estemos sentados, y rodeen nuestra copa con un círculo rojo para indicar que no le queda mucho espinitoso. Tampoco falta mucho para que en el espejo de nuestro probador escojamos alternativas a nuestra prenda.

Tecnologías como la recién introducida Kinect por Microsoft abren campos tanto de investigación como comerciales muy extensos, donde el concepto de interacción seguirá vigente pero sin el sufijo “touch”, ya que el toque se vuelve innecesario.

Probablemente en el siguiente paso en cuanto a software y algoritmia será trabajar de una manera mucho más profunda en el reconocimiento de gestos, ahora ya siguiendo nuestras articulaciones en 3D.

En cuanto a la visualización tridimensional, la adaptación nativa de los dispositivos multitouch con un estudio específico era algo que se hacía necesario. Seguramente estos paradigmas deban ser modificados en el momento (prácticamente la actualidad) en que las superficies interactivas utilicen técnicas estereoscópicas para verlas en tres dimensiones.

Siendo mucho más concreto, este sistema tiene muchas mejoras por implementar. A continuación se sugieren algunas de ellas por capas.

10.2.1 MEJORAS EN LA CAPA DE CONTROL

La capa de control, y particularmente la librería *vlvisiona*, tiene actualmente una dependencia bastante fuerte con OpenCV. Por comodidad, se utilizaron las librerías de alto nivel de OpenCV para trabajar con las fuentes de vídeo (cámaras y ficheros). Seguramente una gestión directa de las cámaras conferiría más rendimiento y estabilidad, así como mayor capacidad de ajuste de las mismas.

En la misma línea, actualmente *vlvisiona* sólo puede trabajar con una fuente de vídeo a la vez. En ocasiones, es posible que una misma interacción requiera coger imagen de dos cámaras (por ejemplo, tratando de *trackear* una superficie muy grande). Extender el soporte de 1 a N fuentes de vídeo, y buscar una forma de “coser” esas imágenes, sería un gran avance.

Por supuesto, ir desarrollando más algoritmos de tracking, como clases hijas de *Procesador*, irá poco a poco creando una biblioteca de algoritmos realmente interesante.

Por último, se sugiere estudiar implementar ciertas partes, ya bien sea de la propia librería o de los algoritmos desarrollados con las nuevas técnicas de programación genérica en GPUs (GPGPU, o *General Purpose Graphics Processing Units*) (Harris, 1995).

10.2.2 MEJORAS EN LA CAPA DE GESTIÓN

La primera y más obvia mejora en la librería *vlmt* es que la clase *Cursores* soporte más información acerca del toque, como la forma de éste. En el planteamiento inicial no se contempló esta característica porque ninguna de las aplicaciones que se había pensado lo requería. Sin embargo, puede ser muy útil en muchos casos (por ejemplo, si los toques representaran “pinceles virtuales”, el usuario podría decidir con qué formas pintar.

Siendo extensible como es la librería, al igual que sucedía con *vlvisiona*, según las necesidades lo vaya exigiendo se irá creando una biblioteca de dispositivos, que soporten nuevos protocolos, además de TUIO.

Por otra parte, a pesar de que el interfaz de uso de esta librería es realmente simple, sólo se puede acceder a ella desde C/C++, o creando bindings a otros lenguajes. Sería un buen avance tener soporte nativo de los lenguajes de programación más populares, incluidos los de *scripting*, como C#, Java, Python, Perl o Lua.

Por último, la mejora más crucial sería crear dentro de *vlmt* soporte para gestos estándar, y no delegar en las aplicaciones la tarea de reconocerlos. De esta manera, igual que con los dispositivos, se iría creando una biblioteca de gestos, reutilizable en diversas aplicaciones.

10.2.3 PARADIGMAS DE MOVIMIENTO 3D

El paradigma denominado *PRotaZoom* ha resultado ser increíblemente útil para inspeccionar terreno y arquitectura digital tridimensional. Desgraciadamente, dada la

compleja naturaleza de las operaciones geométricas que supone, en alguna ocasión (muy contadas) se ha visto descontrolado por algún tipo de operación matemática indebida. Como decimos, esto ha sucedido forzando el algoritmo a casos muy extremos, a los que hasta ahora ningún usuario ha llegado. Esto ha obligado a tener siempre a disposición del usuario un botón de “pánico” que devuelve la vista a una posición estándar, por seguridad. Las pocas veces que se ha llegado a esta situación siempre ha tenido que ver con el uso interno de *quaternions* (estructuras matemáticas de cuatro elementos que representan una operación en el espacio, muy utilizadas en los motores gráficos por su eficiencia).

El algoritmo debe ser sometido a un exhaustivo análisis formal para conseguir depurar estos mínimos errores.

A nivel investigador, sería de gran interés extender estos paradigmas al uso de pantallas interactivas estereoscópicas, viendo cómo el usuario afronta la interacción en entornos 3D reales, no proyectados a superficies 2D.

Apéndice I.FICHEROS DE CONFIGURACIÓN

A continuación se presentan los ficheros XML que configuran distintas partes del sistema, como ejemplos de uso y para describir los parámetros más relevantes.

i. LIBRERÍA VLVISIONA

Se presenta a continuación el fichero de configuración general de la librería vlvisiona, en este caso el utilizado para el proyecto La Fuente Encantada (en versión resumida).

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<visiona>
  <fuente alto="480" ancho="640" fps="20" id="0" tipo="camara"/> -->
  <!-- <fuente tipo="avi" fichero="test04.avi" fps="20" /> -->
  <!-- <proxy tipo="red" ip_servidor="10.10.10.10" puerto="1234" /> -->
  <proxy tipo="local"/>
  <procesador debug_algoritmo="no" tipo="reconocedor manos" ventana_debug="no">
    <parametros>
      <parametro max="100.0f" min="0.0f" nombre="grado erosion" valor="2.0f"/>
      <parametro max="100.0f" min="0.0f" nombre="grado dilatacion" valor="4.0f"/>
      <parametro max="100.0f" min="0.0f" nombre="umbral color discretizacion"
valor="110.0f"/>
      <parametro max="100.0f" min="0.0f" nombre="ancho inicio brazo minimo"
valor="12.0f"/>
      <parametro max="100.0f" min="0.0f" nombre="ancho mano minimo" valor="5.0f"/>
      <parametro max="100.0f" min="0.0f" nombre="longitud minima brazo"
valor="20.0f"/>
      <parametro max="100.0f" min="0.0f" nombre="step seguidor brazo" valor="10.0f"/>
      <parametro max="100.0f" min="0.0f" nombre="correccion direccion seguidor brazo"
valor="0.1f"/>
    </parametros>

    <matriz_transformacion>
      <fila1 w="0" x="-2.7397267818450928" y="0.13233335316181183" z="0"/>
      <fila2 w="0" x="2.0721751070595928e-007" y="-2.7169806957244873" z="0"/>
      <fila3 w="0" x="0" y="0" z="1"/>
    </matriz_transformacion>
  </procesador>
</visiona>
```

```

    <fila4 w="1" x="1.3636323213577271" y="1.1856639385223389" z="0"/>
  </matriz_transformacion>

  <polilinea_deteccion normalizado="no">
    <vertice x="336" y="104"/>
    <vertice x="271" y="88"/>
    <vertice x="212" y="82"/>
    ...
    <vertice x="352" y="105"/>
  </polilinea_deteccion>
</procesador>

<postproceso tipo="filtrado">
  <parametro nombre="muestras" valor="3"/>
</postproceso>

</visiona>

```

Nodos y parámetros:

- fuente: define una fuente de vídeo, donde tipo puede ser o “cámara” o “fichero”.
- proxy: define si se utilizará un nucleo en local o en red.
- procesador: configura el procesador. Permite elegir qué procesador generará la fábrica y si se quiere debug gráfico.
 - parámetros: cada uno de las variables que intervienen en el proceso, y que cada procesador interpretará a su manera. Los valores máximo y mínimo permiten autogenerar un interfaz de configuración con *sliders*.
- matriz_transformación: de fine por filas la matriz de transformación que se utiliza para pasar de espacio cámara a espacio aplicación.
- polilinea_deteccion: define la polilínea que delimita la forma irregular de la mesa.
- postproceso: escoge un algoritmo de postprocesado entre los implementados, en es te caso el filtrado de la señal.

- parámetro: los parámetros propios del postproceso escogido.

A continuación se muestra el fichero de configuración de visión de calibración. De nuevo el utilizado en La Fuente Encantada.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>

  <calibracion_camara desplazamiento_x="-0.41099825501441956"
desplazamiento_y="0.038999989628791809" escala="1.006000280380249"
rotacion="1.5699987411499023"/>

  <marcas_calibracion>
    <marca id="A" x="-0.93945002555847168" y="0.74219000339508057"/>
    <marca id="B" x="-0.93945002555847168" y="-0.75780999660491943"/>
    <marca id="C" x="0.93554997444152832" y="0.74219000339508057"/>
    <marca id="D" x="0.93554997444152832" y="-0.75780999660491943"/>
  </marcas_calibracion>

</config_calibracion>
```

Este fichero está autogenerado por el calibrador. Sus nodos y parámetros son:

- calibración_camara: define la situación de la cámara con respecto a la superficie interactiva.
- marcas_calibración: define en qué coordenadas el usuario ha tocado sobre las marcas de calibración, para después poder generar la matriz de transformación.

ii. LIBRERÍA VLMT

La librería vlmt carece de ficheros de configuración, por sencillez de uso. La aplicación que la utilice, debe simplemente inicializarla indicando el dispositivo que se quiera utilizar, y en caso de requerirlo, sus parámetros (como la IP y puerto en la que el tracker le servirá los datos, en el caso de TUIO).

Las aplicaciones de VidealAB se lanzan mediante el intérprete propio vlad. En este caso, las preferencias de multitouch se pasan por línea de comandos. Por ejemplo, una típica línea para lanzar el intérprete, utilizando el sistema multitouch sería

```
vlad.exe gi.xml -splash splash.png -i interaccion.xml -ceguidir ./cegui -ceguiscript santi.lua -ceguiscript_post santi_post.lua -multitouch w7
```

donde la opción “-multitouch” está definiendo el tipo de dispositivo que se va a utilizar, en este caso “w7” indica un dispositivo tipo Windows 7.

Esta misma línea de comandos para un dispositivo de tipo TUIO sería,

```
vlad.exe gi.xml -splash splash.png -i interaccion.xml -ceguidir ./cegui -ceguiscript santi.lua -ceguiscript_post santi_post.lua -multitouch tuio -tuio_ip 10.10.10.43 -tuio_puerto 3333
```

iii. ESTRATEGIAS DE VUELO

Como se explicó anteriormente, el vuelo del SANTI define estrategias de vuelo, que son configuradas una por una en el fichero de configuración vuelo.xml.

A continuación se expone un ejemplo, con las estrategias de vuelo de los paradigmas desarrollados en este proyecto.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<vuelo>
  <angulo_minimo h="-180" p="-89.9" r="-5"/>
  <angulo_maximo h="180" p="89.9" r="5"/>
  <!-- angulo_inicial h="90" p="0" r="0" -->
  <angulo_inicial h="-55.3687" p="-21.1237" r="0"/>
  <velocidad_angular_maxima h="15" p="15" r="15"/>
  <movimiento_horizontal
```

```

        velocidad_minima="-0.1"
        velocidad_maxima="1"
        velocidad_inicial="0.2"
        aceleracion="0.01"/>
<movimiento_vertical
        velocidad="2"
        aceleracion="0.01"
        offset_z_colisiones="-5000.0"/>
<posicion_inicial x="-31.3342" y="-6.55424" z="18.9827"/>
<posicion_minima x="-200.0" y="-200.0" z="-200.0"/>
<posicion_maxima x="200.0" y="200.0" z="200.0"/>
<estrategias>
    <EstrategiaVueloSimpleOrbitaCamara santi="santi" velocidad="2.0"/>
    <EstrategiaVueloSimple santi="santi" velocidad="2.0"/>
    <EstrategiaMTPRotaZoom
        activable="no"
        modelos_colision="modelos/colisiones_exterior.ive"
        pan_en_plano_perpendicular="si"
        distancia_minima_colision="2"
        tiempo_inercia="1" />
    <EstrategiaCortesMultiTouch
        activable="no"
        min_z="-4.6"
        max_z="18"
        factor="24" />
    <EstrategiaMTWalkTouch puerto_tuio="3333"
        activable="no"
        modelos_colision="modelos/colisiones_interior.ive"
        altura_vista="1.7"
        distancia_choque="0.5"
        desplazamiento_lateral="no"
        velocidad_angular="40.0"
        desliza_al_chocar="no"
        giro_detiene_avance="si"
    />
</estrategias>
</vuelo>

```

Como se puede apreciar, en primera instancia se definen una serie de parámetros que tienen que ver con la configuración global de la cámara, y a continuación se definen los parámetros propios de las estrategias. Pasamos a comentar las dos que nos interesan:

- **EstrategiaMTPRotaZoom**

- **ativable:** parámetro interno de todas las estrategias de vuelo definidas en SANTI.
- **modelos_colision:** modelos geométricos para calcular colisiones y no permitir avanzar en su interior.
- **pan_en_plano_perpendicular:** define si el movimiento de arrastre se hace en el plano perpendicular a la normal de la intersección (visualización de arquitectura) o no (visualización de terreno), según se ha descrito anteriormente.
- **distancia_minima_colisiones:** el mínimo espacio permitido hasta el modelo de colisiones, medido en unidades del espacio modelo.
- **tiempo_inercia:** segundos de duración de la inercia del movimiento.
- **EstrategiaMTWalkTouch**
 - **altura_vista:** altura del punto de vista con respecto al suelo.
 - **distancia_choque:** distancia mínima de acercamiento a una superficie de colisión.
 - **desplazamiento_lateral:** determina si es factible desplazar al usuario en una dirección diferente a la marcada por la dirección del punto de vista. Utilizando la nomenclatura vista en la memoria, decir “no” a este parámetro implicaría que $\alpha=\beta$ siempre.
 - **velocidad angular:** determina la velocidad de giro cuando la posición del punto de vista se está desplazando en curva.
 - **desliza_al_chocar:** determina si se permite “ir a ras” de una pared de colisión, “deslizándose” sobre ella, siempre y cuando el vector de desplazamiento tenga una componente fuera de la perpendicular de esa pared.

- giro_detiene_avance: determina si al introducir un segundo toque, es decir, permitir al usuario mover el punto de vista, se detiene el avance que se llevaba.

BIBLIOGRAFÍA

ACM. (1996). *ACM SIGCHI Curricula for Human-Computer Interaction*. Retrieved from <http://old.sigchi.org/cdg/cdg2.html>

Bittner, Kurt, & Spence. (2003). *Use case modeling*. Addison-Wesley.

Bretzner, L., & Lindeberg, T. (1998). Feature Tracking with Automatic Selection of Spatial Scales. *Computer Vision and Image Understanding*, (pp. 385 - 392).

Buxton, B. (2011, Marzo 21). *Multi-Touch Systems that I Have Known and Loved*. Retrieved from <http://www.billbuxton.com/multitouchOverview.html>

Davies, E. R. (2005). *Machine Vision: Theory, Algorithms, Practicalities*. Morgan Kaufmann.

Eberly, D. H. (2003). *Game Physics*. Morgan Kaufmann.

Fowler, M. (2004). *Refactoring. Improving the design of existing code*. Addison Wesley.

Gallmeister, B. O. (1995). *POSIX 4: programming for the real world*. O'Reilly Media.

Gamma, E., Helm, R., Jhonson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Glassner, A. S. (1989). *An Introduction to Ray Tracing*. Morgan Kaufmann.

- Google. (2011). *Google Earth*. Retrieved from <http://www.google.es/intl/es/earth/index.html>
- Han, J. (2005). Low Cost MultiTouch Sensing through Frustrated Total Internal Reflection. *Symposium on User Interface Software and Technology*, (pp. 115-118). Seattle, WA, USA.
- Han, J. (2010). *Jeff Han homepage*. Retrieved from <http://www.cs.nyu.edu/~jhan/>
- Harris, M. (1995). Mapping computational concepts to GPUs. *ACM SIGGRAPH Courses*. Los Angeles, California: ACM SIGGRAPH Press.
- Intel; Open Source Community. (n.d.). *OpenCV*. Retrieved from <http://opencv.willowgarage.com>
- Kaltenbrunner, Bovermann, Bencina, & Costanza. (2005). TUIO - A Protocol for Table Based Tangible User Interfaces. *6th International Workshop on Gesture in Human-Computer Interaction and Simulation*. Vannes, France.
- Lindeberg, T. (1993). Detecting Salient Blob-Like Image Structures and Their Scales with a Scale-Space Primal Sketch: A Method for Focus-of-Attention. *International Journal of Computer Vision*, 238-318.
- Microsoft Corp. (2009). *MSDN Library*. Retrieved from <http://msdn.microsoft.com>
- Microsoft. (n.d.). *Surface Computing / Tabletop Display Research*. Retrieved from http://research.microsoft.com/en-us/um/people/merrie/surface_research.html

NUI Group. (n.d.). *Community Core Vision*. Retrieved from <http://ccv.nuigroup.com/>

Ramirez, P., Barros, E., & Lima, S. (2002). *Digital signal processing*. Cambridge University Press.

Rerych, M. (1970). *Wasserfallmodell*. TU-Wien: Institut für Gestaltungs- und Wirkungsforschung,.

Seoane, A., Taibo, J., Hernández, L., & Jaspe, A. (2007). Hardware-Independent Clipmapping. *WSCG, 15th International Conference in Central Europe of Computer Graphics*. Piltsen, Alemania.

Tinkerman. (2009). *Rolled Compliant Layer Method*. Retrieved from http://www.peauproductions.com/comp_layer.html

Universitat Pompeu Fabra. (2010). *Music Technology Group*. Retrieved from <http://www.mtg.upf.edu/>

Wright, Freed, & Momeni. (2003). OpenSound Control: State of the Art. *3rd Conference on New Instruments for Musical Expression*. Montreal, Canada.

Wu, Y., & Huang, T. S. (1999). Vision-Based Gesture Recognition: A Review. *Gesture-Based Communication in Human-Computer Interaction, Volume 1739*, 103-115.