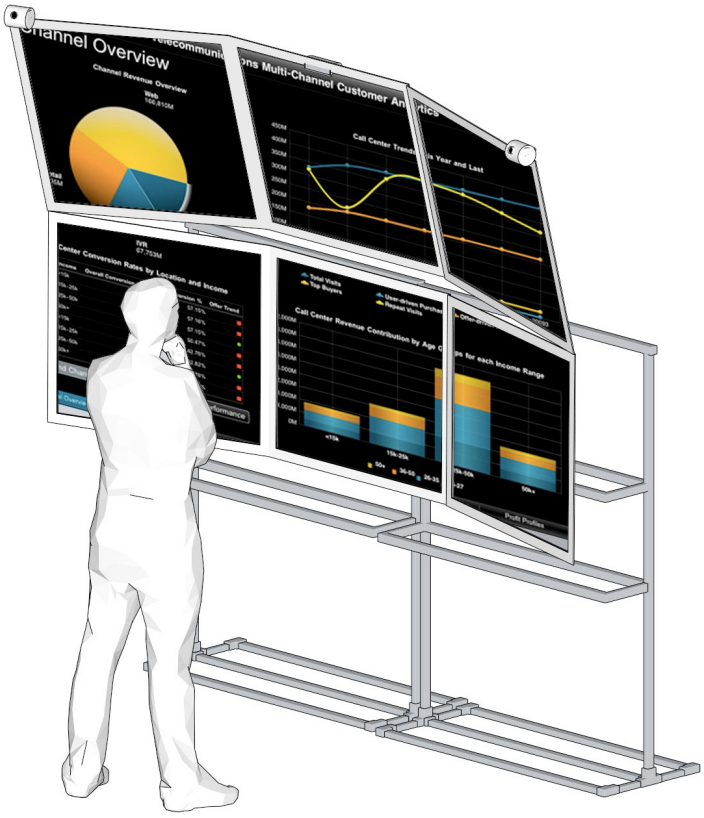
Instituto Tecnológico de Costa Rica
Escuela de Matemática / Escuela de Ingeniería en Diseño Industrial

iReal
eScience
group



iReal 2.0 : Interfaces en ambientes de realidad virtual

Informe final de proyecto de investigación

Autores

Ph.D. Jorge Monge-Fallas. Escuela de Matemática
Ph.D. Franklin Hernández-Castro. Escuela de Diseño Industrial
Ing. David Segura Solís. Escuela de Diseño Industrial

Contenido

iReal
eScience
group

Nombre del Proyecto

iReal 2.0: Interfaces en ambientes de realidad virtual (2013-2014)

Autores y direcciones

Ph.D Jorge Monge-Fallas(**coordinador**) jomonge@itcr.ac.cr. Escuela de Matemática,
Ph.D Franklin Hernández-Castro, franhernandez@itcr.ac.cr. Escuela de Diseño Industrial
Ing. David Segura Solís, dsegura@itcr.ac.cr. Escuela de Diseño Industrial

Resumen

El objetivo de iReal 2.0 era desarrollar una herramienta que permitiera visualizar y analizar los datos generados por los sensores del proyecto eBridge en el laboratorio de visualización inmersiva del programa de eScience. Para el proyecto se tenía que definir una estrategia sobre el uso y el desarrollo de los elementos de la interface, así como el software y hardware necesarios para proyectar, en tiempo real, ambientes tridimensionales en los que se pueda experimentar fenómenos espaciales de forma que el usuario esté inmerso en el ambiente, ya sea física o virtualmente.

Estas interfaces tridimensionales ya fueron exploradas inicialmente por varios integrantes del grupo eScience (incluyendo a los investigadores Franklin Hernández y José Castro), quienes visitaron en marzo del 2010 el encuentro PRAGMA 18 en San Diego California. En esta visita se pudo observar el estado del arte en varios países de los más avanzados en esta área, entre ellos Estados Unidos, Canadá, Japón, India y Corea.

Estas interfaces fueron investigadas y se desarrollaron varias iniciativas en el proyecto iReal 1.0., lo que claramente fue la base para el desarrollo de iReal 2.0.

El proyecto iReal ha ido consolidando el laboratorio de visualización que se había planteado años atrás. El desafío consistía en la visualización de información (en alta resolución) en forma de ambientes tridimensionales virtuales y aun más retador: la manipulación de esos sistemas. Este último es el objetivo fundamental alanzado en iReal 2.0

Métodos: uno de los aspectos importantes en este proyecto era contar con una configuración adecuada del sistema de visualización inmersiva "*cave*". Esta configuración adecuada a la tecnología elegida en su momento (Alioscopy), nos permitió tener un sistema más robusto y estable, de tal forma que centramos nuestros esfuerzos en dos líneas: desarrollo de contenido 3D Aliocopy (autoestereoscopico lenticular) y el desarrollo de la interface.

En cuanto al contenido para visualizar en el "*cave*", primero se hizo la configuración del cluster, poniéndose a funcionar el sistema completo, para posteriormente iniciar pruebas en el desarrollo de contenido 3D de Alioscopy. En este caso, este fue el mayor desafío y el principal aporte de este proyecto.

La configuración final para controlar, vía software el clúster del "*cave*", se realizó a través de la implementación del framework de Realidad Virtual "CalVR", un sistema desarrollado por la Universidad de California, San Diego. Proyecto que inicio en el año 2010 y el cual ha sido pionero en el campo de la investigación visual (http://ivl.calit2.net/wiki/index.php/CalVR).

Para la etapa relacionada con la interface, se utilizaron las herramientas de desarrollo que utilizan las aplicaciones para OS X sistema operativo de Apple y el iOS 7 sistema operativo de las iPads, iPods y iPhone. Esta decisión se basó principalmente en el hecho de que este sistema es el más maduro del mercado en el uso de gestos y por tanto la parte de la investigación que se desarrolló sobre estos sistemas touchscreens se vio beneficiada por esta condición. El grupo de datos base para el desarrollo de la interface fue el proyecto de eBridge.

Resultados: A nivel de hardware, se logró una configuración estable del laboratorio de visualización científica "*cave*", con 6 nodos que cuentan con GPUs de alto desempeño, con comunicación de red basada en fibra óptica. Por el lado del software se logró implementar el framework de visualización inmersiva CalVR, el cual nos permite por medio de algunos parámetros, definir las configuraciones para la autoestereoscopia que requiere los monitores Alioscopy.

A nivel de interface, se logró desarrollar una aplicación para iPhone y iPad para controlar el "*cave*" (disponible en App Store de Apple llamada **iReal)**, basada en los requerimientos de visualización del proyecto eBridge. Además la aplicación incorpora la funcionalidad de comunicarse con el "*cave*" a través de un socket de forma inalámbrica.

A nivel del clúster y el desarrollo de contenido 3D, se logró sistematizar el desarrollo y despliegue de contenido Alioscopy en el "cave" que nos permitirá diversificar las aplicaciones de este sistema de visualización.

Además, paralelamente se a instalado una nueva configuración para el control del TDW (sistema normal de visualización) a través de SAGE ("Scalable Adaptive Graphics Environment") y OmegaLib. La idea de este trabajo en paralelo es contar con un sistema de visualización más robusto y complementario.

El SAGE es una arquitectura de transmisión de gráficos que habilita un acceso interactivo, una visualización y la capacidad de compartir gran cantidad de datos, en una variedad amplia de resoluciones, formatos, fuentes, con mucha facilidad de uso, permitiendo hacer pruebas de concepto con el objetivo de llevar todo el sistema a una nueva configuración del "*cave*".

OmegaLib es un Framework de aplicaciones de multivista para ambientes de realidad híbrida (HREs), que consiste en una combinación de ambientes immersivos, con mosaicos de monitores de alta resolución. Un HREs logra crear un ambiente 2D/3D transparente que soporta un análisis rico en información, como también una simulación en realidad virtual.

Conclusión: El objetivo de desarrollar una herramienta que permitiera visualizar y analizar los datos generados por los sensores del proyecto eBridge en el "*cave*" fue satisfactoriamente alcanzado.

Palabras claves

Visualización de información,interfaces inmersivas, realidad virtual OpengGL, Alioscopy, iOs, TDW, autoestereoscopia, HREs, lenticular.

1. Introducción

Este proyecto se encuentra bajo el programa de investigación de eScience y nació como una extensión natural de cuatro proyectos anteriores realizados principalmente por dos de los investigadores (Jorge Monge-Fallas y Franklin Hernández-Castro), a saber: Visualización Tridimensional de Información Jerarquizada, Refinamiento de algoritmos en árboles de conos, Sistemas de interfaces intangibles e iReal (Sistemas de interfaces en ambientes de realidad virtual).

Todos estos proyectos giraron alrededor de la visualización de estructuras complejas en ambientes virtuales. En el caso del proyecto iReal se logró desarrollar la tecnología para dotar al TEC de una instalación de realidad virtual. Para el proyecto se definió una estrategia sobre el uso y el desarrollo de los elementos de la interface, del software y hardware necesarios para proyectar en tiempo real ambientes tridimensionales en los que se pueda experimentar fenómenos espaciales de forma que el usuario esté inmerso en el ambiente tanto física como virtualmente. Además en los últimos tres años se ha estado trabajando en “super computing”, aspecto fundamental para el óptimo funcionamiento de estos sistemas.

La experiencia adquirida por la participación de la Escuela de Computación con su clúster, la Escuela de Electrónica y las dos escuelas con que se han trabajado trabajo en visualización por años (Diseño y Matemática) han evolucionado de una forma natural en la consolidación de esta temática. Con este proyecto se pretendía generar una plataforma que sirviera de base de visualización para otro proyecto de eScience: Predicción remota de fallas en puentes: “eBRIDGE” y que el conocimiento adquirido a nivel de software y hardware junto con el desarrollo de interfaces, permitiera poner este laboratorio a disposición de proyectos de visualización científica para enfocar otros problemas de la realidad nacional de instituciones como: Ovsicori, CCSS, MOPT entre otras.

Este proyecto está explorando nuevas posibilidades de interfaces tridimensionales, aspecto que está muy poco desarrollado en el mundo. Como mencionamos previamente, varios de los integrantes del grupo eScience (incluyendo a los investigadores Franklin Hernández y José Castro) participaron en el 2010 el encuentro PRAGMA[1] 18 en San Diego California, donde se constató que el tema de interfaces estaba poco desarrollado.

El TEC de Costa Rica, lleva más de 10 años trabajando en proyectos conjuntos con las escuela de Ingeniería en Diseño Industrial, Matemática y Computación, y en el último periodo; la Escuela de Electrónica.

En el Tecnológico se está trabajando en muchas investigaciones actualmente; con la consolidación de este laboratorio de visualización y la experticia adquirida, el proyecto se convierte en una solución tanto a nivel de visualización como en el uso de interfaces, a prácticamente toda la generación de información de estos proyectos de investigación, desde visualizaciones geo-referenciadas en ingeniería agrícola hasta configuraciones de encimas en biotecnología.

En el país existe una necesidad creciente de este tipo de sistemas, ejemplos de aplicación inmediata son el monitoreo de sistemas nacionales como el de electricidad, agua, teléfono, tráfico vehicular o manejo de riesgos (inundaciones o terremotos por ejemplo). Es decir, instituciones nacionales como el ICE, CCSS, Ovsicori, AyA, MOPT y CNE son solo algunos de los posibles ejemplos de aplicación inmediata de estos sistemas.

Para tener una idea tanto de los objetivos planteados como de los logros alcanzados, mostramos en la siguiente tabla un resumen de los mismos.

1 http://www.pragma-grid.net/

1.1 Cuadro resumen de los objetivos, actividades y productos alcanzados

En esta tabla se muestra el en resumen, el proyecto en términos de los objetivos alcanzados y los productos obtenidos:

Objetivo general: Generar una herramienta que permita visualizar y analizar los datos generados por los sensores del proyecto eBridge en la instalación inmersiva del programa de investigación eScience

Objetivos específicos	Productos	Actividades	Período ejecución	Responsable	Grado de Avance
1. Diseñar la estrategia de visualización y control de los datos	1.1. Estrategia clara de uso de los datos actuales 1.2. Algoritmo de traducción de los datos	1.1.1 Análisis de los formatos que se usan actualmente en los datos 1.1.2 Definición de la posible interface para manipular estos datos en tiempo real 1.2.1 Diseño de un algoritmo de traducción de los datos y su visualización 1.2.2 Definición del modo en que se comunicarán estos datos	1.1. 2013 - 1.6. 2013	Franklin Jorge David Asistentes	100%
Presentación de informe	Informe	Elaboración de informe de avance del proyecto	1.8. 2013 1.9. 2013	Jorge, Franklin David	100%
2. Desarrollar las sub-rutinas necesarias para poner en práctica las estrategias diseñadas en el objetivo anterior	2.1 Un conjunto de sub-rutinas que sea capaz de interpretar los datos, visualizarlos y manipularlos	2.1.1. Diseño del sistema en forma de módulos escalables 2.1. 2. Diseño de la visualización de la información 2.1.3 Programación de las sub-rutinas 2.1.4 Programación de la visualización de la información 3D	1.6. 2013 - 1.12. 2013	Jorge y Franklin David y asistentes	100%
3. Implementar las sub-rutinas en una aplicación funcional que se pueda correr en forma autónoma	3.1 Una estación en funcionamiento donde se exploren las capacidades de la nueva herramienta.	3.1.1. Conexión de las herramientas generadas en las etapas anteriores 3.1.2 Pruebas de conexión entre los componentes del sistema, móviles y estacionarios	1.1. 2014 - 1.6. 2014	Jorge Franklin David Asistentes Johan	100%
4. Validar con un focus group la versión alfa de la herramienta	4.1 Lista de aciertos y debilidades de la herramienta	4.1.1 Selección de los tester 4.1.2 Diseño del test 4.1.3 Aplicación del test 4.1.4 Análisis de los datos	1.6. 2014 - 1.12. 2014	Jorge, David, Franklin	100%
Presentación de informe	Informe	Elaboración del informe final del proyecto	1.10. 2014 - 1.12. 2014	Jorge, David Franklin	100%

2. Marco teórico

2.1 Definición de la tecnología y la interface

Para definir qué tecnología se iba a utilizar en el sistema completo de interface, se establecieron 4 fases a realizar en el primer año del proyecto:

- fase 1: definición del tipo de datos a visualizar y su escala
- fase 2: definición detallada del tipo de visualización que se propone
- fase 3: definición detallada del tipo de interacción que se propone
- fase 4: visualizaciones similar a la aplicación iReal (en iTunes store) en todas las plataformas, incluyendo el cave, el TDW, el iPad y la comunicación wifi entre ellas.

A continuación se muestra un esquema de los tópicos involucrados y sus correspondientes responsables para el inicio del 2013:

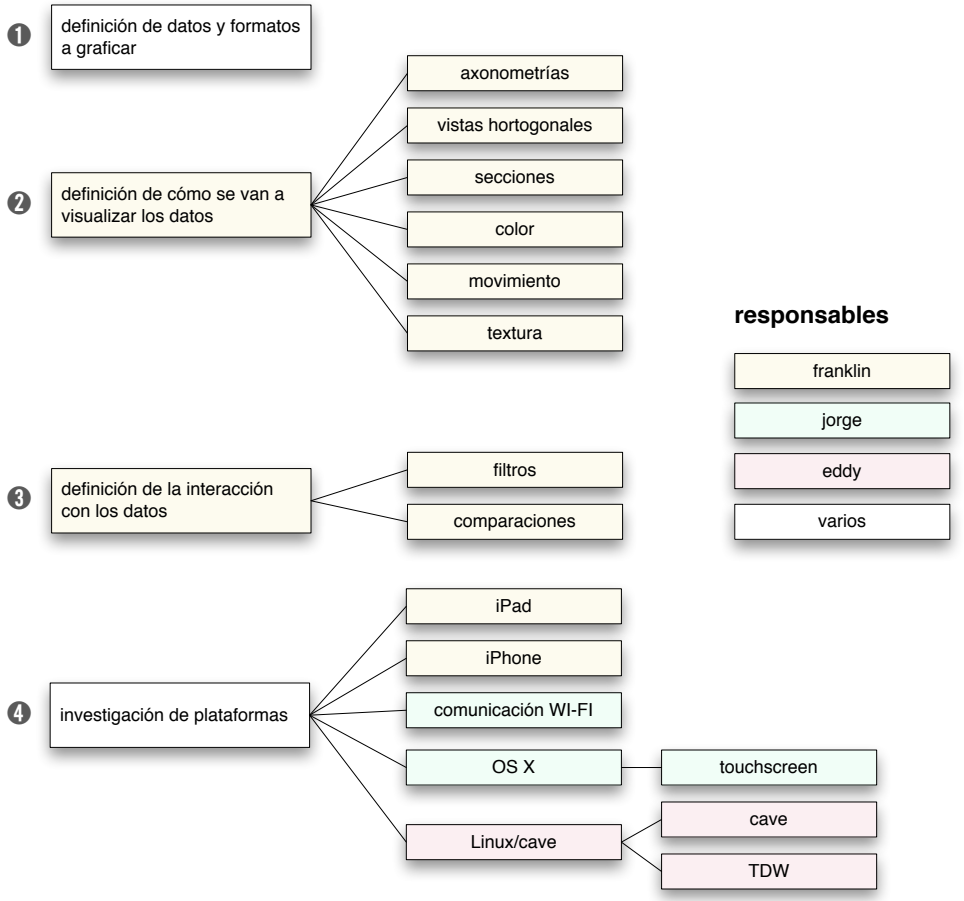


Figura 1. Fases y responsables iReal 2.0 (el nombre de “Eddy” corresponde al M.Sc. Eddy Ramírez, ingeniero en informática, en ese momento profesor de la carrera de computación, con quien se empezó el proyecto, a partir del año 2014, fue el ingeniero David Segura quien se encargó de esta parte)

Entre las tecnología involucradas tenemos el Tiled Display Wall (TDW). Un TDW consiste en la unión de equipos de visualización (generalmente pantallas), los cuales se unen formando un sistema de mayor tamaño.

Utilizando este enfoque es posible indicarle a cada sección del sistema que muestre una parte específica de la imagen, permitiendo representar los datos con mayor precisión y detalle. Aplicando estos conocimientos es posible crear aplicaciones de visualización inmersiva para diversos fines, tales como simulación de erupciones volcánicas, corrientes marinas o condiciones climáticas.

Existen varios esquemas en los que se pueden presentar las imágenes para generar este efecto. Uno de los esquemas más interesantes para lograrlo son los dispositivos **auto estereoscópicos**, los cuales logran separar las imágenes distintas que se le presentan a cada ojo directamente en la pantalla, esto da una gran flexibilidad gracias a que no se necesitan lentes para observar las imágenes con profundidad. Es precisamente este enfoque el que utilizan los displays de Alioscopy que conforman el “cave” de iReal.

La visualización inmersiva utilizando pantallas especializadas se basa en la estereoscopía lenticular, la cual consiste en una serie de métodos que crean la ilusión de profundidad al ver una imagen. La imagen presentada está compuesta realmente por varias imágenes ligeramente distintas que se le presentan a cada ojo por separado, el cerebro toma la información de ambas direcciones y las interpreta como si hubiese profundidad.

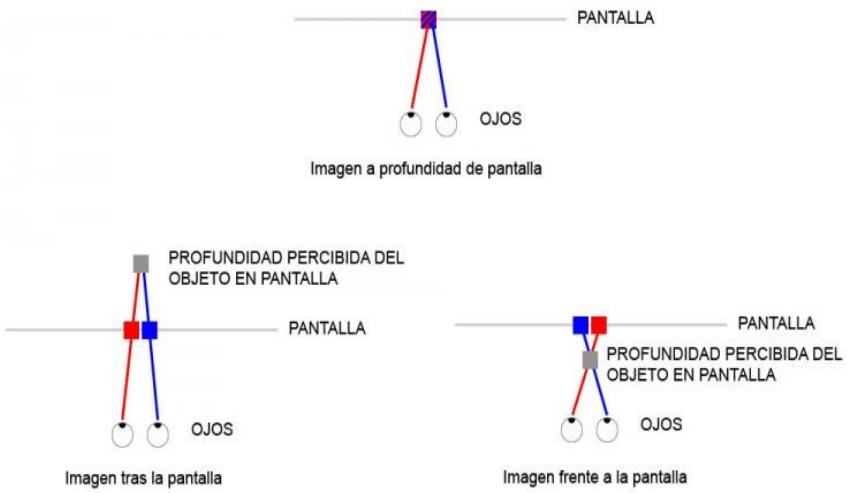


Figura 2. Percepción de profundidad mediante estereoscopia

Alioscopy es una tecnología desarrollada en Francia, la cual se especializa en la creación de displays auto estereoscópicos de múltiples dimensiones. Sin embargo, no se está hablando simplemente de pantallas auto estereoscópicas, sino que poseen una cualidad no vista antes en el mercado, la cantidad de puntos de vista desde los que se puede ver el efecto de profundidad no está limitado a uno o dos, sino que las pantallas Alioscopy son capaces de generar ocho vistas distintas a la vez. Es por esta combinación de características propias de la tecnología que al utilizar las pantallas Aliscopy se habla de Real-3D: la generación de imágenes que superan los métodos convencionales de creación de contenido 3D, dándoles la sensación de ser más reales.

En la siguiente imagen se puede ver cómo se realiza, en esta tecnología, la distribución de los puntos de vista en sus displays. En lugar de un pixel común, un pixel en un display Alioscopy se comporta como un lente curvo, en el cual se generan una imagen para el ojo izquierdo y otra para el ojo derecho en cada una de sus vistas. El resultado de esto es una pantalla que genera imágenes estereoscópicas desde ocho distintos puntos focales.

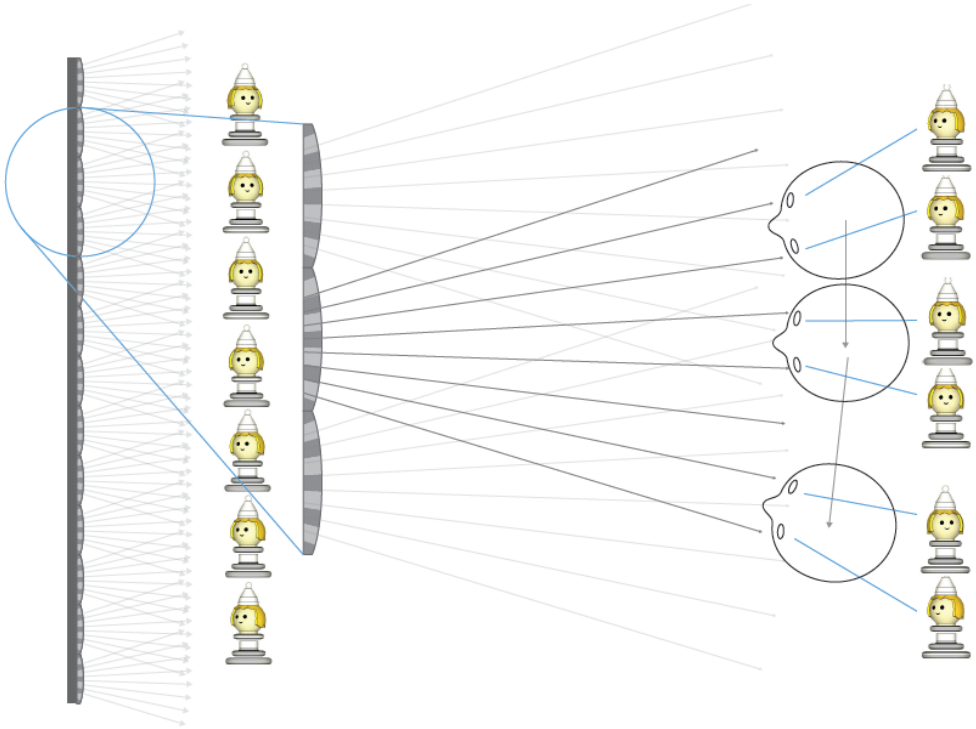


Figura 3. Puntos focales de la tecnología Alioscopy

La forma en como se proyecta una imagen en 3D utilizando la característica particular de la forma de los pixeles de Alioscopy, demanda que se produzcan ocho imágenes del mismo objeto mostrado como se mencionó previamente. Cada una de estas imágenes se logra al mover un poco el punto de vista desde el cual se aprecia la imagen.

En el fondo lo que recibe el sistema para poder proyectar una imagen de n x m es en realidad, una imagen de tamaño n x 8m, de modo que el mismo *shader* (nombre que se le da al consolidado de imágenes) se encarga de cortar esa imagen y distribuirla en los diferentes pixeles. Si por error, se envía una imagen de tamaño mayor o inferior, el shader siempre toma la octava parte, provocando imágenes completamente cortadas y des-coordinadas.

La forma de como se genera esta imagen consolidada es pegando sub-imágenes del mismo objeto pero con una vista ligeramente diferente.

Para la implementación orientada a objetos, se separaron las diferentes partes del sistema, de modo que los componentes se aislaron en forma de “capas” como se muestra en el siguiente diagrama.

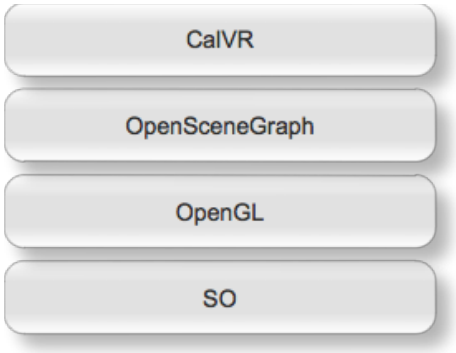


Figura 4. Capas de los componentes de software del Cave

Lo que se destaca en esta implementación es el hecho de haberse implementado el conjunto de herramientas gráficas 3D de OpenSceneGraph (OSG), siendo esta herramienta utilizada en aplicaciones que tienen que ver con la simulación, juegos, realidad virtual, modelado y visualización científica.

Dada esta implementación en el CAVE, dio la posibilidad de crear un esquema de trabajo muy versátil, con el cual no es necesario adentrarse en el código fuente del programa y hacer los modelos en OpenGL, sino que, basta con obtener modelos generados por programas de modelados tipo CAM, CAE, como SketchUp o SolidWorks para posteriormente ser cargados por el CAVE, a partir de aquí el efecto auto-estereoscopico es generado de una forma automática para cada uno de los objetos modelados.

Es importante mencionar, que gracias a OSG el CAVE se potencian muchas otras aplicaciones, por ejemplo, se logró hacer lectura de un escaneo tridimensional generado localmente, lo que posibilita que una nube de puntos de cualquier objeto y se logre cargar en el CAVE y ser visualizado con los mismos efectos tridimensionales y auto-estereoscopico.

El CalVR es una plataforma basada en OSG, es decir, lo que se hace con el OSG, es lo que hace el CalVR, pero en este caso, el CalVR agrega otras funciones adicionales, como soporte a cluster, tracking o rastreo por medio de dispositivos basado por VRPN, interfaz de usuario, una interfaz de programación modular (plugins), modos de navegación, efectos de sonido por medio de un servicio OpenAL , entre otras características.

2.2 Los nodos y el front-end (clúster)

El cave se construyó utilizando el siguiente hardware:

- 6 displays Alioscopy 3D HD 42" (2x3)
- 6 tarjetas madre Intel Desktop DX58SO
- 6 procesadores Intel Core i7
- 24 memorias RAM Corsair Intel Blue Dominator 8 GB
- 6 tarjetas de video EVGA GeForce GTX 580
- 12 Discos duro Seagate Barracuda Green 2 TB
- 8 Fuentes de poder Ultra LifeTime Series Pro 750W
- 1 switch Arista Data Center Switch 7124SX 10G SFP
- 1 Rack Blackbox Freedom con 12-24 Rails, 45U, 19"
- 6 cases Maxtop ICX-4889B-20 4U Rack Mount Style Rackmount Server Case

Algunos de estos componentes se dañaron por lo que fue necesario realizar la sustitución. Se sustituyeron dos tarjetas de video por una tarjeta EVGA GeForce GTX 550 y una EVGA GeForce GTX 650 ya que no se contaba con los recursos para comprar tarjetas de video del mismo modelo.

Con estos equipos se crea una configuración estilo clúster. En una estructura de clúster se tiene un computador que cumple la función de nodo control (Front-End): en este nodo se realizan las operaciones de coordinación con los nodos y la captura de eventos del dispositivo iOS, para luego enviar las instrucciones y comandos a otros equipos dentro de una red interna; estos equipos son conocidos como nodos y están sujetos a las labores que el Front-End les indique realizar (figura 5). Se utilizó esta configuración ya que se necesitaba que los CPUs del cave se comunicaran entre ellos y pudieran realizar una distribución equitativa del trabajo, y un modelo tipo clúster es ideal para este tipo de funciones.

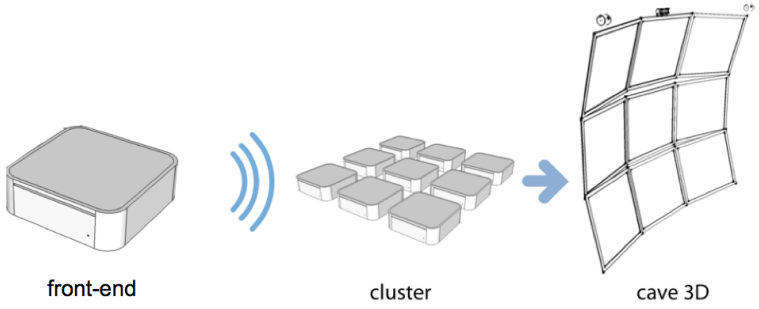


Figura. 5 Papel del front-end y configuración de los nodos

Para que el clúster pudiera considerarse como tal, no basta con la instalación de los equipos independientemente, sino que debía buscarse un medio mediante el cual estos puedan interactuar entre ellos de manera rápida. Se configuró una red óptica a 10G interna con un switch Arista que engloba a todos los equipos que conforman al clúster y el Front-End del mismo. Gracias a esta red, el Front-End puede comunicarse con los nodos y estos pueden interactuar entre ellos.

La hoja técnica de los displays Alioscopy indica que la potencia consumida por cada display es de 200W, las fuentes de poder tenían problemas al tratar de mantener más de dos pantallas encendidas al mismo tiempo, por lo que se debía realizar el diseño de la red considerando tener sólo dos pantallas por fuente de poder. Una estructura en la que varias fuentes de poder se encuentren conectadas en serie tampoco servía, por lo que se debió realizar una conexión que utilizara una fuente de poder para cada par de pantallas y cada par de CPUs, para un total de seis fuentes de poder, las cuales se conectarían directamente a la red eléctrica del laboratorio.

Para montar las pantallas se utilizó una estructura de metal que pudiera sostenerla. Se colocaron en pares de dos pantallas en un arreglo de 2 x 3 (dos filas, tres columnas) aprovechando de una mejor forma las dimensiones de las pantallas.

Después de la configuración del hardware del cave, se definió el software para el control del clúster.

Para realizar este control se opto por utilizar como base la versión 14.1 de Slackware a 64bits, una de las distribuciones mas longevas de Linux, muy utilizada en sistemas que requieren estabilidad y alto desempeño. La siguiente figura muestra los componentes de software que se implementaron.

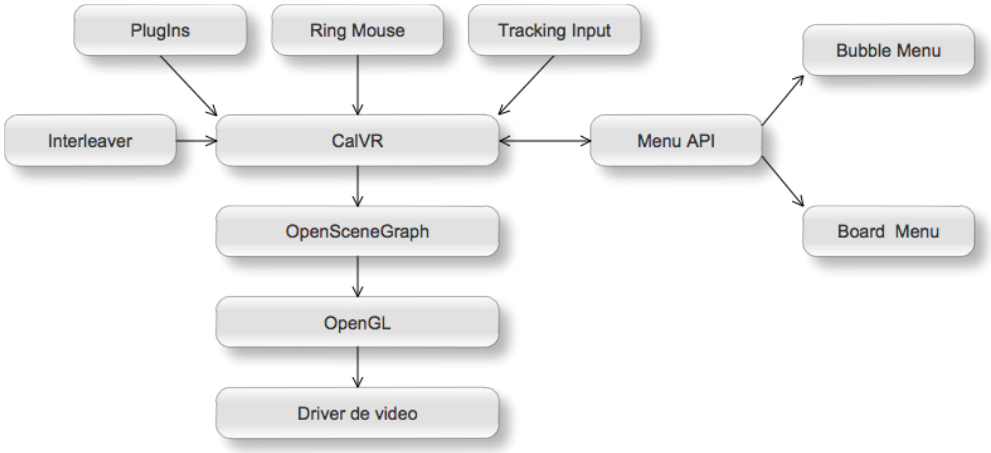


Figura. 6 Componentes de Software

El CalVR representa el núcleo de coordinación tanto en los aspectos de computación distribuida, como la visual y los medios de entrada. El CalVR logra potenciar y definir funcionalidades especificas gracias a los PlugIns (elementos que se desarrollan por aparte), dicho de otra manera, el CalVR es el núcleo, y es por medio de los plugins donde se desarrollan las aplicaciones personalizadas.

Estos PlugIns van desde librerías matemáticas, hasta sistema de información geográfica; dependiendo de la disponibilidad dentro de la comunidad Open Source de Linux con el lenguaje C++. Del mismo modo, estas funcionalidades especificas pueden ser operadas por los medios de entrada, como Ring Mouse y el Tracking Input. El Ring Mouse es el medio estándar de operar el núcleo del CalVR, del mismo modo que el Tracking Input es el API que logra implementar la interacción con dispositivos mas especializados, como el Kinect, controladores de consolas como el Wii Remote, o bien el controlador de PS3 y cámaras de video, entre otros.

Al CalVR se le incorporo la libreria Interleaver, encargada de ejecutar todos los algoritmos necesarios para la obtención del efecto auto-estereoscopio, siendo estos configurados por una serie de parametros.

El CalVR se logra operar gracias al API del menú, siendo este el intermediario entre el usuario y el contenido proyectado en los displays. Este API ofrece las funcionalidades de navegación dentro del mundo virtual, activación o desactivacion de PlugIns y hasta controlar los elementos que se desean cargar o ejecutar dentro de ambiente.

Como anteriormente se menciono, el CalVR se basa en las capacidades gráficas del OpenSceneGraph, que posee un extenso conjunto de rutinas para diversas funcionalidades visuales. Es gracias al OSG que el CAVE puede llegar a solventar muchas necesidades visuales de los proyectos de investigación.

Generado el contenido por el OSG, esto es traducido y transferido a la tecnología OpenGL, donde se genera todo el contenido tridimensional y la visualización de cada una de las cámaras requeridas, esto es posible gracias a los drivers de video de nVidia, (tarjetas de video las utilizadas por nosotros y por muchos proyectos similares).

2.3 Conceptualización a nivel de interface

2.3.1 Análisis de las variables a visualizar

Para empezar a diseñar la interface lo primero que se realizó fue una reunión de trabajo con los investigadores del proyecto eBridge. El objetivo de esta reunión fue definir con mucha exactitud cuál es la naturaleza del proyecto y de ahí identificar cuáles son las necesidades de visualización que tiene.

El proyecto eBridge se basa en dos partes principales, una parte en la que se monitorea los puentes por medio de sensores remotos (para tener claro el comportamiento del puente en el uso diario) y otra en la que se define un modelo de comportamiento teórico.

Posteriormente se desea también comparar ambos escenarios (el teórico y el práctico) con el fin de afinar el modelo teórico y mejorar su capacidad de predicción (figura 7).

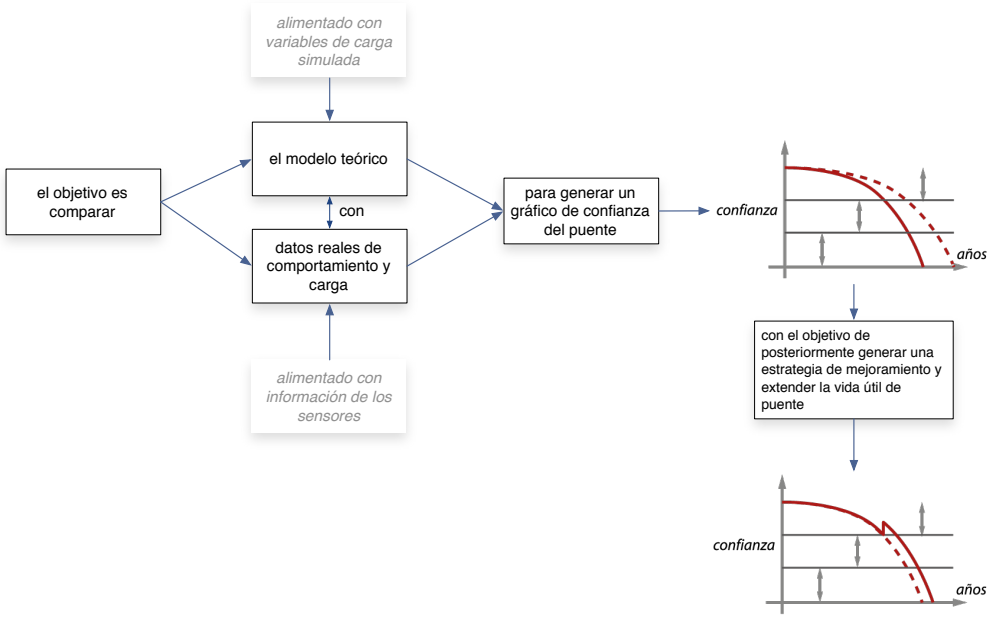


Figura 7. Diagrama básico del proyecto eBridge

Con esto claro, se definen los dos tipos de variables que eran importantes en este proyecto, el primer grupo son las variables que leen los sensores. En este grupo se miden cuatro variables importantes:

- deformación (medida en mm),
- deflexión (medida en mm),
- vibración (medida en m/seg.2) y
- desplazamiento de juntas (medida en mm).

Todas estas variables se miden en una dimensión en el tiempo, es decir se define un intervalo de muestreo y en cada uno de estos se miden las variables.

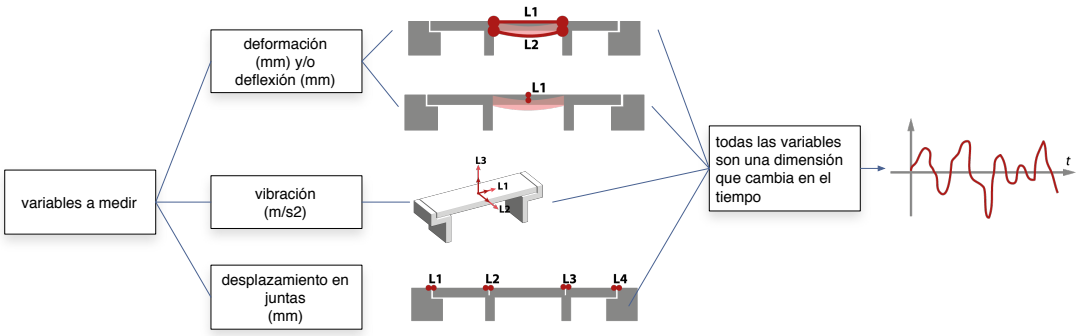


Figura 8. Conjunto de variables medidas por los sensores en intervalos de tiempo regulares.

El segundo grupo de variables que es necesario tener en cuenta es el conformado por las variables de las condiciones de uso del puente en ese momento. Las tres variables medidas aquí son: cantidad de vehículos por hora, cantidad de peso por ejes y velocidad de los mismos.

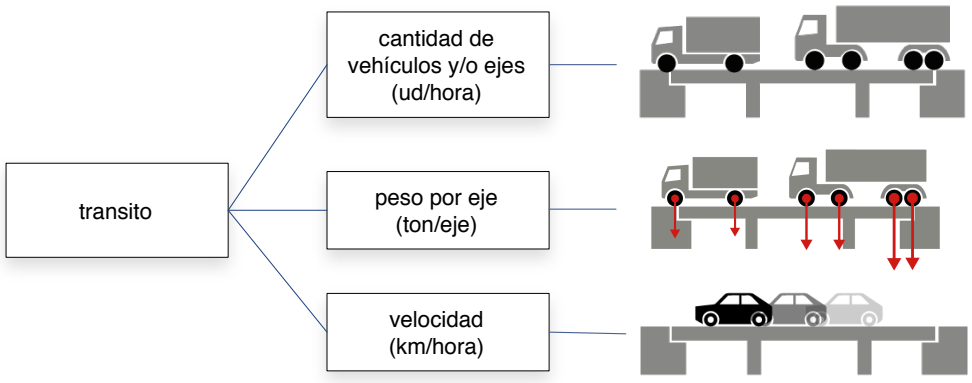


Figura 9. Conjunto de condiciones medidas por los sensores en los mismos intervalos de tiempo.

2.3.2 Desarrollo del primer prototipo

Con estas condiciones claras se procedió a desarrollar un prototipo de la visualización de estos datos con el fin de validar el resultado con los investigadores del proyecto eBridge. En este caso se trabajó con el lenguaje de programación Processing en su versión 2.01

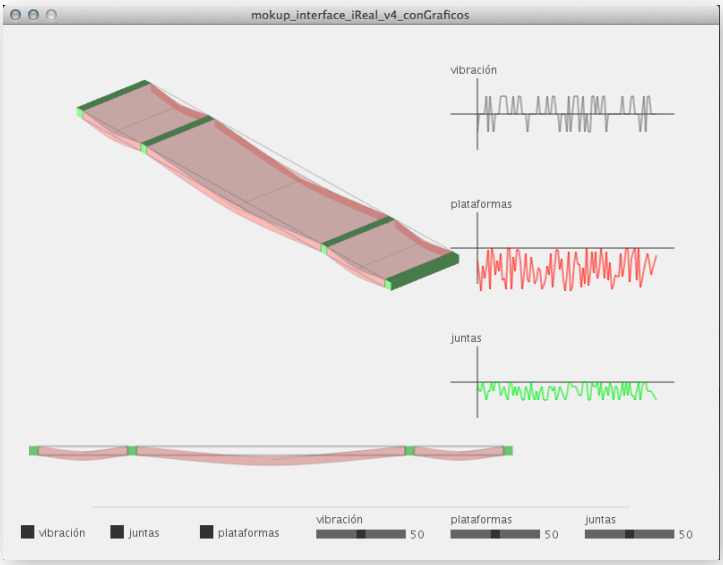


Figura 10. Imagen del primer prototipo mostrando las variables y su primer propuesta de visualización.

Este prototipo se analizó con los investigadores del proyecto al que se le da servicio y se procedió a especificar el tipo de interface que se necesitaría en el proyecto final.

Para el Encuentro de Investigadores del 2014 se desarrolló un segundo prototipo que utilizó el sensor Leap Motion® como entrada de datos, a continuación se muestra una pantalla (figura 9) de este.

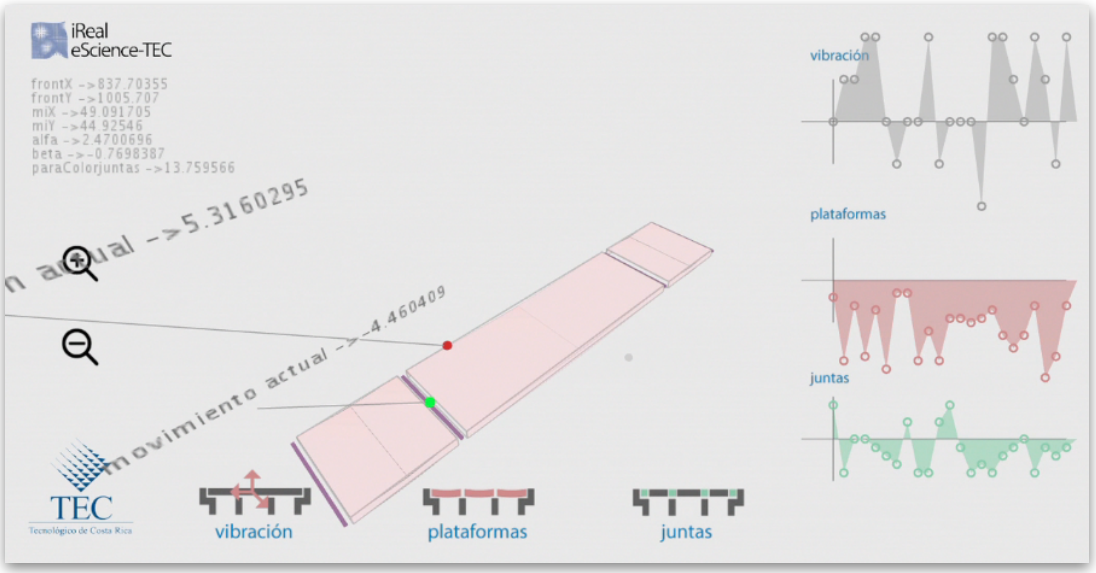


Figura 11. Imagen del segundo prototipo mostrando la propuesta de visualización.

2.3.3 Definición de la arquitectura de la interface

Posteriormente correspondió definir la primera versión de la arquitectura del proyecto. En esta etapa se define que la interface tendrá tres escenarios correspondientes a los objetivos primarios del proyecto:

- análisis de datos teóricos
- observación de datos recolectados
- comparación entre ambos tipos de datos

El primer escenario tiene como objetivo el análisis de los datos teóricos, se usa principalmente para evaluar los resultados que da el modelo, el usuario introduce la cantidad de vehículos por hora (o la velocidad promedio por hora), después define lo que desea ver (losas, vigas, juntas y/o vibración) y lo visualiza. Veamos un esquema de esta arquitectura:

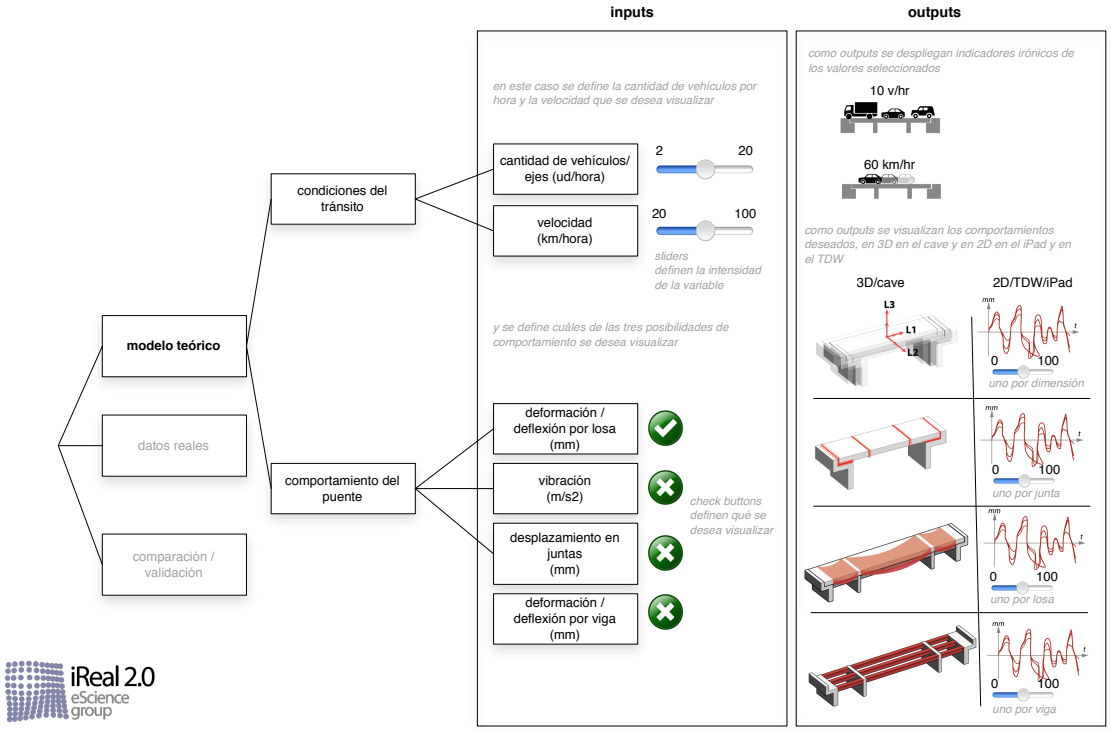


Figura 12. Esquema básico del primer escenario de la interface iReal 2.0 mostrando las entradas y salidas del escenario para analizar los datos del modelo teórico.

El segundo escenario a definir era el que recoge los datos que se obtienen de los sensores, este escenario se usa para ver los datos que se han recolectado. El usuario introduce la fecha, después define lo que desea ver (losas, vigas, juntas y/o vibración) y finalmente lo visualiza.

Se debe notar que en este escenario las condiciones están definidas por el caso real y por tanto al definir la fecha se leen automáticamente todas las demás variables.

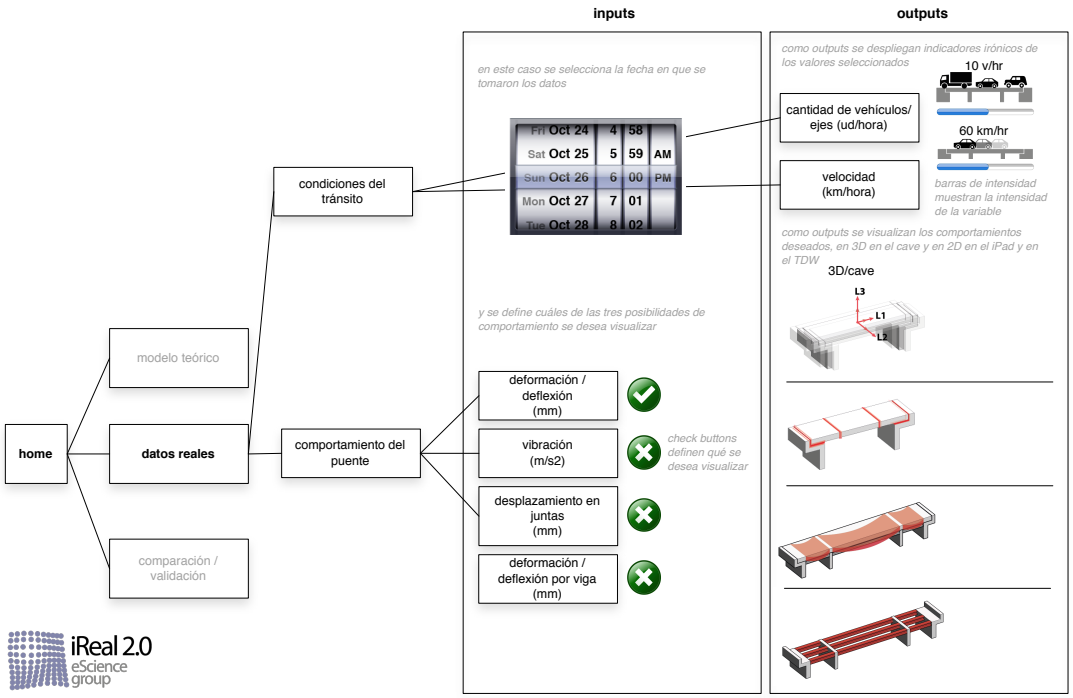


Figura 13. Esquema básico del segundo escenario de la interface iReal 2.0 mostrando las entradas y salidas del escenario para analizar los datos recogidos por los sensores.

El tercer y último escenario fue el de comparación, este escenario se usaría para comparar los resultados que se obtienen a partir del modelo teórico y los datos reales, el usuario introduce la fecha, después define lo que desea ver (losas, vigas, juntas y/o vibración) y visualiza la comparación. En este caso cada una de las variables de salida; deformación (medida en mm), deflexión (medida en mm), vibración (medida en m/seg.2) y desplazamiento de juntas (medida en mm) se muestran dos veces, una vez como están los datos de los sensores y otra como dice el modelo teórico, la idea general es poder comparar ambos comportamientos con el fin de saber que tan afinado está trabajando el modelo desarrollado.

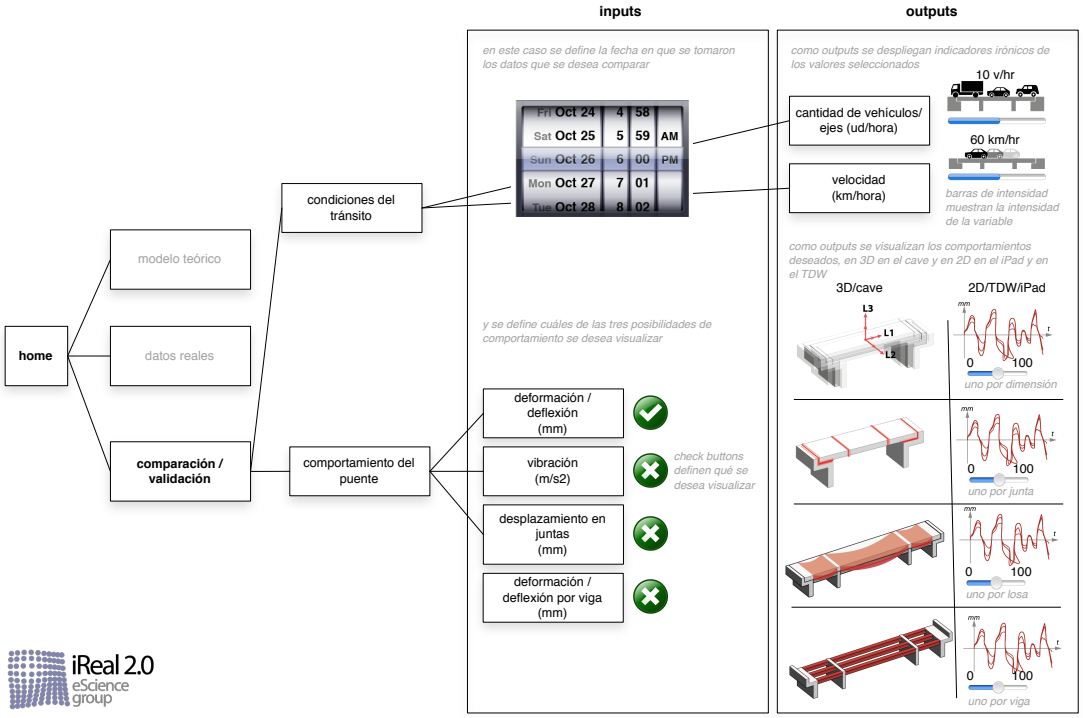


Figura 14. Esquema básico del tercer escenario de la interface iReal 2.0 mostrando las entradas y salidas del escenario.

Al final se implementa un cuarto escenario en el que se reproduciría la aplicación que se generó en el proyecto anterior iReal 1.0, en ese se muestra el puente y sus alrededores en forma tridimensional con el fin de observar de cerca las características del mismo en el *cave*.

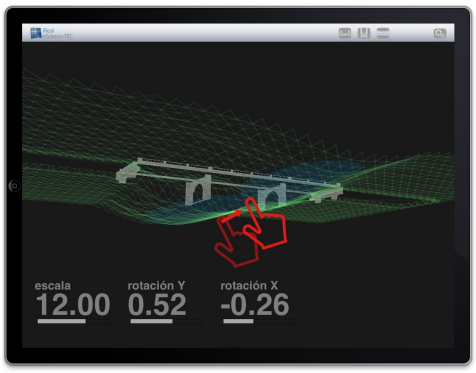


Figura 15. Imagen de la aplicación iReal 1.0, parte del resultado del proyecto anterior

2.3.4 Look & feel

En esta etapa se analiza aplicaciones similares que estaban en funcionamiento en ese momento, dando especial interés a aplicaciones que manejan datos complejos y hacen comparaciones entre ellos. A continuación se muestran algunas de ellas:

Figura 16. Algunos de las aplicaciones de referencia que se analizaron.

Con estas dos etapas concluidas se procedió a definir el look & feel de la futura interface, la cual se desarrolló en iOs.

2.3.5 Implementación en iOS

2.3.5.1 Ambiente específico de programación

En esta etapa se trasladó todo el diseño que se definió en las etapas anteriores al ambiente final en el que la aplicación sería ejecutada, o sea, iOS 7 para iPad.

A continuación se muestran las primeras imágenes resultado de todo el proceso de definición de la interface, nótese que en esta etapa ya se pueden ver las primeras definiciones de las pestañas de navegación, así como las zonas en las que se podrá controlar la interface:

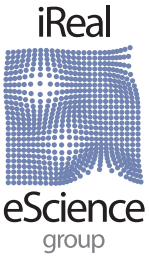
Figura 17. Primeros escenarios con la arquitectura y el look & feel propuestos

Dentro de esta plataforma el primer trabajo que se debió hacer fue definir en cuál de los ambientes de dibujo se debía programar, pues en esta plataforma hay varias opciones.

Figura 18. Diseño gráfico de la futura interface en iPad

Para el diseño propuesto (véase la figura 17) se necesitaron al menos dos ambientes, uno en dos dimensiones en el que se puedan trabajar los gráficos y los botones y uno 3D donde se pueda visualizar la posición actual del puente, que a su vez se visualizará en el cave de iReal, recordemos que esta interface sirve a modo de “control remoto” para el uso del “*cave*” 3D y del TDW.

2.3.5.2 Frameworks, usos y posibilidades



Para dibujar en iOS existen varias posibilidades, en cualquiera de ellas se usan bibliotecas definidas para ese objetivo. Se puede dividir estas bibliotecas en internas o externas, es decir, en bibliotecas propias de Objetive-C, o bibliotecas generadas por terceros y que se pueden usar al interior de este ambiente.

Con las bibliotecas nativas (propias del ambiente) solo es posible dibujar en dos dimensiones, para dibujar en tres dimensiones generalmente se usa OpenGL-ES que es una versión del OpenGL estándar, adaptada para este ambiente. De hecho todos los juegos que corren en tres dimensiones en iPhone y iPad usan esta única biblioteca.

Para los gráficos en 2D

Para dibujar los gráficos de las variables y los botones se decidió usar solamente de las bibliotecas nativas, esto principalmente por su eficiencia, y eficacia comprobada, pues tiene acceso al las rutinas más internas del sistema operativo iOS.

Se puede decir que en este ambiente hay tres bibliotecas principales que se usan con este fin [2]:

1. UIKit
2. Core Animation
3. Core Graphics

UIKit

UIKit (abreviatura de *User Interface Kit*) es la biblioteca que permite crear y manipular las “vistas” o “*canvas*” en las que se dibujan los objetos 2D, es decir, los elementos que contienen las interfaces como botones, imágenes, *sliders*, etc.

En realidad UIKit es, a su vez, una interface de Quartz 2D para el ambiente de portátiles (iPhone y iPad). Quartz 2D por su parte, es la biblioteca que corre en OSX (el sistema operativo de los ordenadores de apple®) con este mismo fin de manejar los elementos de las interfaces. Este hecho garantiza algo de compatibilidad entre el código de los portátiles (iPhone y iPad) y los equipos de escritorio (laptops y torres).

Con UIKit se crean, borran, mueven y manipulan las mismas vistas o contenedores (*Views*) a través de la clase llamada *UIView* (UI vienen de UIKit). La aplicación misma corre en una *UIView* principal o raíz, en la que se agregan “a manera de hijos” otras *views* que contienen el resto de los elementos. Al interior de algunas de éstas vistas se puede dibujar o simplemente cargar una imagen o dibujo vectorial (generalmente en .svg [3]).

Core Animation

Esta es una biblioteca orientada a la animación de las diferentes *Views*, que como se ha dicho son los contenedores en los cuales se “dibujan” los distintos objetos.

En estos casos Core Animation se encarga de hacer animaciones entre estos diferentes “contenedores”, por ejemplo, si se desea que un botón cambié de lugar al presionarlo, el mejor modo de hacerlo es usando una animación ya definida en esta biblioteca. Claro que ésta misma tecnología se puede usar para mover un dibujo; en este caso el dibujo se puede cargar en una *View* (como imagen o como vector) y hacer que esta se mueva a través de la pantalla con funciones definidas en Core Animation, lo que sería bastante eficiente pues las funciones de esta biblioteca son de bajo nivel. Este enfoque es muy usado en juegos, donde algún dibujo (como un animal o “alien”) se mueve a través de la pantalla con algún fin.

---

[2] Nahavandipoor, N. (2011)Graphics and Animation on iOS. Sebastopol, CA:O’Reilly Media, Inc.

[3] .sgv abreviatura de Scalar Vector Graphics, que son dibujos en vectores que son interpretados y dibujados automáticamente por objetive-c

Core Graphics

Nuestra tercera biblioteca es Core Graphics, que es la biblioteca encargada finalmente de dibujar. Es decir, después de tener una estructura de vistas, botones, sliders y demás; cuando finalmente se desea dibujar un “triángulo verde” en la pantalla (por ejemplo), la encargada de hacer eso es la biblioteca Core Graphics. Para lo cual se vale de funciones específicas que veremos en las siguientes secciones.

UIView y DrawRect

Como se dijo anteriormente, la clase *UIView* es la clase que alberga todos los elementos necesarios para las aplicaciones, y por supuesto, los dibujos realizados en tiempo real. Cuando se crea un controlador esta clase ya viene asociada al mismo y por tanto no aparece en primera instancia en el conjunto de archivos de la aplicación. En realidad la vista o *UIView* es una propiedad de cada controlador, y como tal, existe desde que el controlador fue creado (generalmente por la plantilla [4]).

Sin embargo, uno de los objetivos era el de implementar las rutinas de dibujo en la clase menciona, pero para esto era necesario tenerla explícitamente. Para lograr este objetivo se eligió crear aparte un objeto de la clase *UIView* y luego asociarlo al controlador, dicho de otro modo, se crea un objeto *UIView* y se declaró como la propiedad del controlador en el que se esta trabajando.

Una vez con el objeto creado y asociado, usamos el método de nombre *drawRect ( )* [5]. Es en este método donde se escriben las instrucciones para dibujar. Es decir, *drawRect ( )* es la función o método que será llamado para hacer los dibujos necesarios.

En caso de ser implementado el método *drawRect ( )* va a ser llamado por el sistema cuando haya momentos de desuso del procesador o cuando suceda alguna interacción de parte del usuario, tocar un botón o cambiar de vista, por ejemplo. Por esta razón se determina llamar este método con el fin de refrescar el dibujo, esto se hace a través de las instrucciones *setNeedsDisplay ( )* o *setNeedsDisplayInRect ( )*, según se desee refrescar total o parcialmente la vista. Esto es útil cuando se tiene una animación. Por ejemplo, en casos de animaciones como en los juegos, se crea un *timer* [6] que llama a este método *drawRect ( )* cada 1/24 de segundo para mantener la fluidez del movimiento.

Contexto gráfico

Debido a que se puede dibujar en cualquier objeto del tipo *UIView*, un objeto puede dibujar en otro. Por ejemplo, podría suceder que un objeto controlador que calcula una gráfica específica dibuje sus resultados en otro objeto (o en varios) que usa sólo de pantalla de salida de datos a manera de pizarra (*output*), por esta razón es necesario definir un “contexto de dibujo” para ejecutar la mayoría de las instrucciones de este tipo.

Es decir, un contexto de dibujo, es la definición de un lugar donde dibujar. Además, un contexto gráfico también guarda parámetros como el color de las líneas y del relleno de las formas, el grosor de las líneas, las transparencias al dibujar, etc.

Los contextos de dibujo son creados automáticamente por Quartz 2D y se pueden leer con la función: *UIGraphicsGetCurrentContext ( )*, de este modo pueden ser especificados como parámetros al dibujar algo, así la función sabe dónde se desea dibujar y con qué características.

Debido a que el contexto puede definir y guardar estas características de dibujo, también se usa para definir una serie específica de condiciones con las que se desea dibujar algo y luego regresar a otras.

En este caso se hace rutinas específicas para los gráficos rellenos, los vacíos y los botones, por esta razón antes de definir estas características se recurre a guardar el contexto de dibujo en el que está, después cambiar las características, dibujar el gráfico con perímetro por ejemplo y luego volver al

---

[4] La mayoría de las veces los proyectos en iOS empiezan con una estructura que se crea a partir de plantillas (templates) que se optimizan para cada uso como juegos o tablas, etc.

[5] Apple Inc. (2011) Drawing and Printing Guide for iOS: Graphics & Animation: 2D Drawing. Cupertino, CA

[6] Hernandez-Castro, F. Monge, J. (2011). Flujo de control en iOS.Tecnología en Marcha. Volumen 25, No. 5. Editorial Tecnológica de Costa Rica

contexto guardado de modo de que no se vean afectadas otras partes del dibujo con las condiciones del cuadrado. Esto fue posible con instrucciones como *CGContextSaveGState ( )* y *CGContextRestoreGState ( )*.

Colores en iOS

Dentro de este mismo concepto de contexto gráfico es posible, por supuesto, definir los colores con los que se desea dibujar. Para esto existen los espacios cromáticos RGBA, CMYK y gray scale, es decir, rojo-verde-azul-alfa, cían-magenta-amarillo-negro y escala de grises. Todos los espacios se define en ámbitos [0-1] y no en el habitual [0-255] que se usa en casi todos los ambientes gráficos. Si no se define el espacio cromático el mismo por omisión será RGBA.

Una vez definido el espacio de color se pasa a la creación del objeto mismo que identificará el color. Como trabajamos estrictamente orientado a objetos el color es también un objeto, en este caso de la clase *UIColor*, parte de la biblioteca *UIKit*.

En este caso se usa RGBA y variaciones sobre el alfa de un color específico:

- CGColorSpaceRef colorspace = CGColorSpaceCreateDeviceRGB();
- CGFloat components [ ] = {0.8, 0.8, 0.0, 1.0};
- CGColorRef color = CGColorCreate(colorspace, components);

Por supuesto una vez definido el color se debe declaró como color de fondo o de perímetro de las formas que se dibujarán inmediatamente después, esto se hace a través de funcione de la biblioteca Core Graphics:

- *CGContextSetFillColorWithColor(unContexto,miColor);*
- *CGContextSetStrokeColorWithColor(unContexto,miColor);*

Path y polígonos

Finalmente se llegó a las funciones donde se dibujan los objetos, por ejemplo los polígonos. En este caso hay funciones pre-establecidas en Core Graphics para dibujar polígonos específicos.

En este caso se usa dibujos a través de puntos, que es un modo mucho más adaptable a los gráficos que se necesitan según el diseño original . En este caso los puntos se concatenan en un recorrido (*path*) y este recorrido luego se añade al contexto.

Para dibujar de este modo hay dos instrucciones principales, una para moverse a algún punto deseado sin dibujar y otra para dibujar entre el punto en que se está y el que se define:

- CGContextMoveToPoint(unContexto, 100, 100);
- CGContextAddLineToPoint(unContexto, 150, 150);

Con estas dos funciones es posible dibujar cualquier forma con líneas rectas.

Sin embargo, en este tipo de gráficos es necesario hacer curvas entre los puntos. Para dibujar formas curvas, se hace desde las primitivas definidas con este fin como círculo y arco, hasta el uso de puntos de control tipo Bézier.

La clase *UIBeziertPath* ofrece la posibilidad de dibujar curvas Bézier cuadráticas y cúbicas, con sus respectivos puntos de control, la figura muestra cómo se definen los puntos:

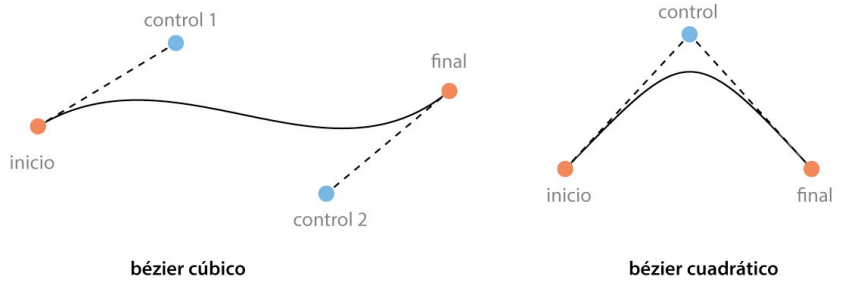


Figura 19. Curvas que se pueden dibujar con la clase UIBeziertPath[7]

en el caso de la cuadrática la sintaxis involucrada se vería algo así:

- *CGContextMoveToPoint (unContexto, 10, 500);*
- *CGContextAddQuadCurveToPoint (unContexto, 150, 10, 300, 500);*

donde el orden de los parámetros sería: el punto de inicio (10,500), el punto de control (150,10) y el punto final (300,500)

el caso de la cúbica se vería así:

- *CGContextMoveToPoint (unContexto, 10, 10);*
- *CGContextAddCurveToPoint (unContexto, 0, 50, 300, 250, 300, 400);*

donde el punto inicial es (10,10), los puntos de control primero y segundo serían (0,50) y (300,250) respectivamente y el punto final (300,400).

Esta tipo de funciones se usaron para dibujar los gráficos con variantes de color y resultado como relleno o vacío, a continuación se muestra una de ellas con sus comentarios:

```
- (void) dibujeUnGraficoFilledCon:(CGContextRef) unContexto yCon: (NSArray*) unaMatriz {
   // antes de cualquier función de dibujo lo mejor es guardar el contexto anterior
   CGContextRef contextoActual = UIGraphicsGetCurrentContext();
   CGContextSaveGState(contextoActual);

   // define el grosor de línea
   CGContextSetLineWidth(unContexto, 1.0);
   // ahora vamos a especificar un color (azul) usando el espacio rgb
   CGColorSpaceRef colorspace = CGColorSpaceCreateDeviceRGB();
   CGFloat components[] = {0.8, 0.8, 0.0, 0.2};
   CGColorRef miColor = CGColorCreate(colorspace, components);

   // ahora se especifica que el color se desea usar en el stroke
   //CGContextSetStrokeColorWithColor(unContexto, miColor);

   // se declara el color como el que se va a usar de relleno
   CGContextSetFillColorWithColor(unContexto,miColor);

   float gap = (anchoScreen+400) /porcentajeH;
   CGPoint myPoint1 = [[unaMatriz objectAtIndex:0] CGPointValue];
   CGPoint mi1rePunto = myPoint1;
   CGPoint myPoint2;
   CGContextMoveToPoint(unContexto, -gap, myPoint1.y);
```

[7] Apple Inc. (2011) Drawing and Printing Guide for iOS: Graphics & Animation: 2D Drawing. Cupertino, CA

```
    int alturaV = 650;
    for (int n = 0; n < porcentajeH - 1; n ++ )   {
        myPoint2 = [[unaMatriz objectAtIndex:n+1] CGPointValue];
        myPoint2.x = gap*n;
        myPoint2.y = alturaV - [self map:myPoint2.y
                        minimoFuente: (float) 1 maximoFuente: (float) 130
                        minimoTarget: (float) 0 maximoTarget: (float) (porcentajeV * 4)]+50;

        CGContextAddCurveToPoint(unContexto,
                            (myPoint1.x +((myPoint2.x-myPoint1.x))/2), myPoint1.y,
                            (myPoint1.x +((myPoint2.x-myPoint1.x))/2), myPoint2.y,
                            myPoint2.x, myPoint2.y);
        myPoint1 = myPoint2;
    }

    CGContextAddLineToPoint(unContexto, anchoScreen, altoScreen);
    CGContextAddLineToPoint(unContexto, 0, altoScreen);
    CGContextAddLineToPoint(unContexto, mi1rePunto.x, mi1rePunto.y);

    // se dibuja el relleno del path
    CGContextFillPath(unContexto);

    // soltamos las variables
    CGColorSpaceRelease(colorspace);
    CGColorRelease(miColor);

    // al final de cualquier función de dibujo hay que restaurar el contexto anterior
    CGContextRestoreGState(contextoActual);
}
```

Para los gráficos en 3D

Para visualizar la posición deseada del puente, se hace necesario echar mano del ambiente OpenGL-ES que es manejado desde un *framework* de nombre GLKit, el cual debe ser importado a la aplicación en forma específica y declarado en cada “clase” u “objeto” en el que se desee usar.

Una vez importado el *framework* es posible declarar un controlador del tipo *GLKViewController* que vendrá con una propiedad View del tipo *GLKView*, también es posible usar solo la vista y añadirla al controlador principal de la aplicación.

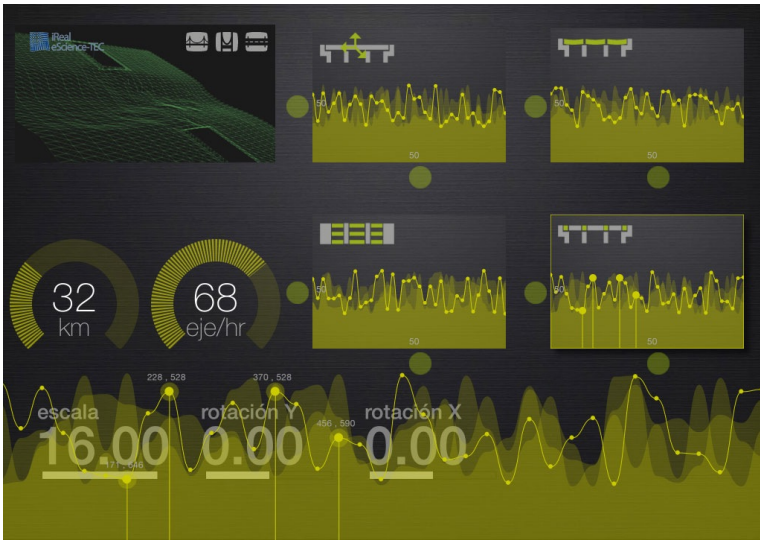


Figura 20. Estado inicial (versión alpha) de la implementación de la interface en iPad mostrando los gráficos y una GLKView donde se puede observar el terreno modelado en OpenGL.

En este caso se monta los objetos por aparte, debido a que ya estaban generados por el proyecto anterior. De este modo hay siete objetos modelados por separado:

- las bases de puente
- la carpeta asfáltica del mismo
- las barandas
- el terreno con su topografía
- el agua del río
- las carreteras de entrada y salida

Este enfoque permite darle colores y texturas distintos a cada objeto mejorando la visibilidad y lecturabilidad de la interface.

Arquitectura

Con estas clases, se diseña una arquitectura simple con tres niveles básicos. El primero es el controlador principal en el que se desarrollan casi todos los elementos o clases de la interface:

- las escenas de los 4 gráficos individuales
- la escena del gráfico grande (abajo) con el nombre de escena principal
- los dos velocímetros
- los botones arrastrables
- y la GLKView para el openGL

al interior de la “escena principal” hay otra clase llamada “HotPoints” que son los botones pequeños que se localizan en cada punto del gráfico de abajo y que se pueden activar para ver sus coordenadas. La figura a continuación demuestra, en forma de diagrama de Ven, las relaciones jerárquicas de la arquitectura.

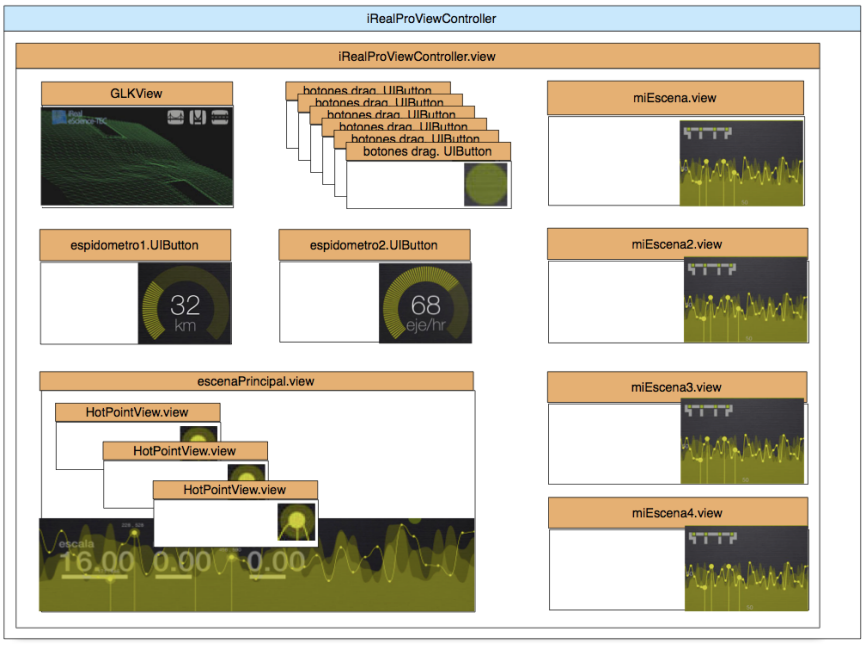


Figura 21. Diagrama de Ven mostrando las estructuras jerárquicas de la interface

Coordenadas polares vrs. coordenadas ortogonales

Para la implementación del dispositivo arrastrable en forma de arco, llamado velocímetro en nuestra arquitectura (ver figura 19), es necesario hacer una transformación entre coordenadas polares y coordenadas ortogonales.

Figura 22. Dispositivo arrastrable en forma de arco

Esto debido a que las coordenadas del "*tap*" o sitio donde se toca el objeto vienen dimensiones ortogonales: el x y el y de la posición en la pantalla. Para poder saber si esas coordenadas están dentro del círculo se decidió hacer una transformación de las mismas a coordenadas polares y con estas nuevas coordenadas comparar el ángulo y el radio de las nuevas coordenadas polares para saber si están o no, dentro del arco.

El diagrama a continuación muestra el análisis trigonométrico hecho para este efecto:

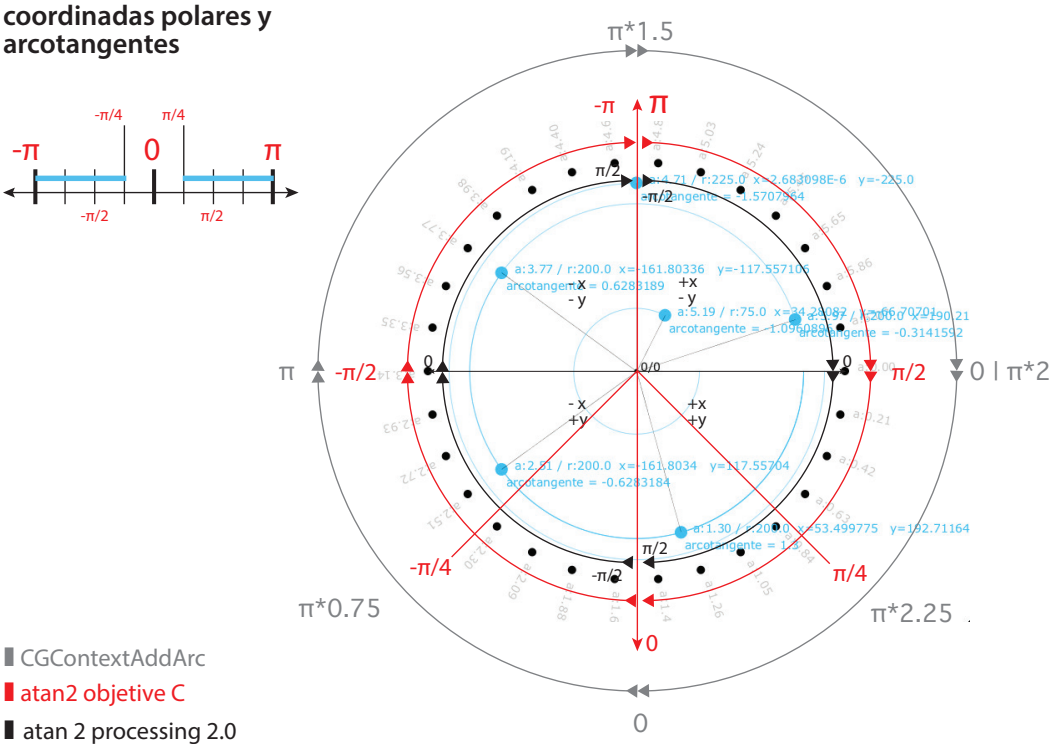


Figura 23. Análisis trigonométrico del dispositivo de arco

Esta idea se implemento con el uso de la función *atan2( )* que viene a ser una versión de la función arcotangente normal pero con un dominio doble para evitar los valores ausentes en la función convencional, generando un ámbito de [0 2π] a diferencia del intervalo convencional de [-π/2 π/2].

El código a continuación implementa está idea:

```
- (void)wasDragged:(UIEvent *)event {
    UITouch *touch = [[event touchesForView:self] anyObject];
    CGPoint localidadDrag = [touch locationInView:self];
    float actualX = localidadDrag.x - 90;
    float actualY = localidadDrag.y - 90;
    float radioActual =  sqrt( pow(actualX, 2) + pow (actualY,2));

    CGFloat rads = 200;
    if (actualY!=0) rads = atan2(actualX, actualY);

    if (radioActual > 60 && radioActual < 90) {
        if ( rads > -M_PI && rads < -M_PI/4) {
            gradoDeAvance = [self map: rads
                        minimoFuente: -M_PI maximoFuente:-M_PI/4
                        minimoTarget: M_PI*1.5  maximoTarget: M_PI* 0.75];
            miValor = [self map: rads
                  minimoFuente: -M_PI maximoFuente:-M_PI/4
                  minimoTarget: 50  maximoTarget: 0];
        }

        if (rads >  M_PI/4 && rads <  M_PI) {
            gradoDeAvance = [self map: rads
                        minimoFuente: M_PI/4 maximoFuente: M_PI
                        minimoTarget: M_PI*2.25 maximoTarget: M_PI*1.5];
            miValor = [self map: rads
                  minimoFuente: M_PI/4 maximoFuente: M_PI
                  minimoTarget: 100  maximoTarget: 50];
        }
    }
    [self setNeedsDisplay];
}
```

2.3.6 Implementación en OS X

Además del trabajo en iOS, se homologó parte del trabajo en la plataforma mac Os X. A la par del trabajo en programación con Objetive-C y cocoa, se vio la necesidad del trabajo en OpenGL específicamente para el modelo 3D.

En la programación se está siguiendo la línea de los desarrolladores de Apple bajo el esquema Model Control View(MCV). La siguiente figura muestra el avance del modelo del puente sobre Os X desarrollado en OpenGL.

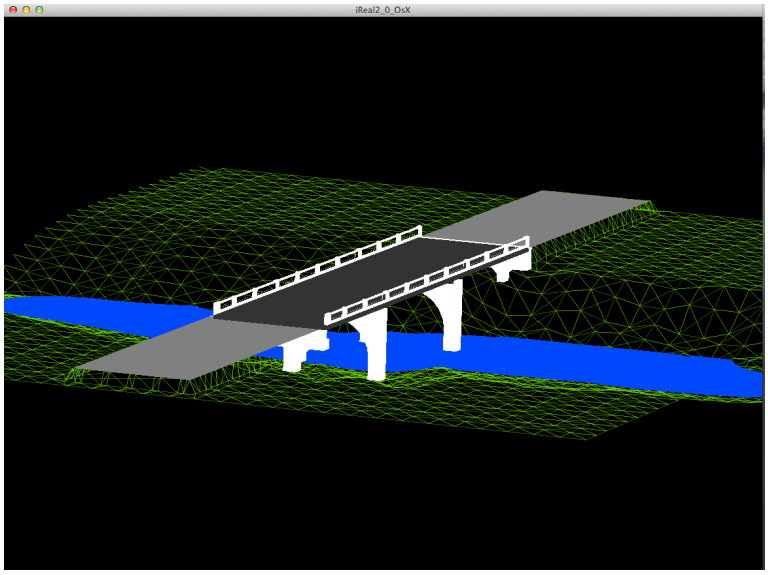


Figura 24. Modelo del puente en 3D sobre la plataforma mac Os X

Además del modelo en 3D, se han implementado las operaciones de zoom y paneo, avanzando en las prestaciones del sistema. La figura 25 muestra el resultado de un zoom hecho al modelo.

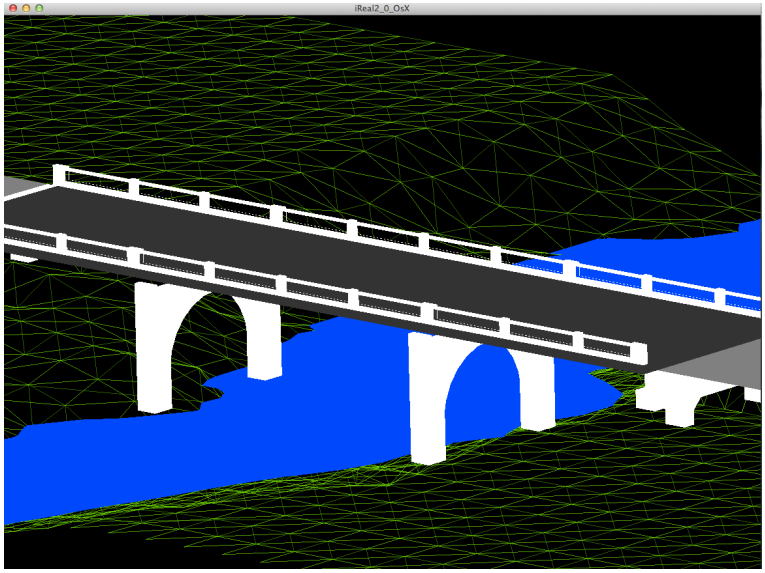


Figura 25. Zoom aplicado sobre el modelo del puente 3D sobre la plataforma mac Os X

2.4 Sistema de visualización inmersiva “cave”

2.4.1 Generación de contenido 3D Alioscopy

Si bien Alioscopy incluye, con la adquisición de sus displays, programas para la generación de imágenes 3D estereoscópicas, estos resultan inadaptables a las necesidades que requiere el proyecto, por ejemplo, el software de Alioscopy solo corre en la plataforma Windows, no ofrecen un API de desarrollo, lo que significa que no se puede acoplar a cualquier otra tecnología o necesidad (código cerrado).

Estos dos factores obligan a una extensa investigación de proyectos similares que se han estado desarrollando en otros grupos de investigación de otras universidades y lo mismo que al estudio de plataformas de Realidad Virtual de código propietario. Con este proceso se llego a la conclusión que lo más apropiado a nuestras necesidades era optar por una plataforma Open Source, pues esto permitiría amoldar la plataforma a las necesidades del proyecto de iReal 2.0. Además es importante mencionar el hecho que los monitores Alioscopy no son una manera estándar para generar contenido estereoscopico, el cual este no es normalmente soportado por plataformas de visualización de código cerrado.

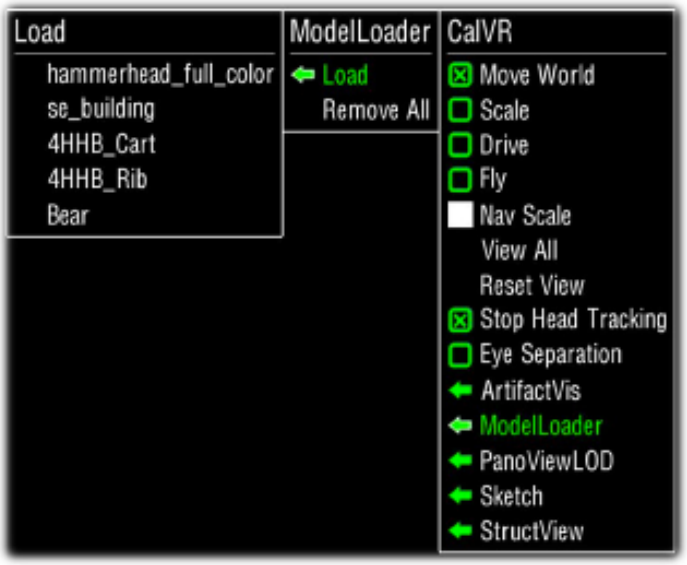


Figura 26. Sistema de Menu del CalVR

La forma clásica en como se logra dar el efecto de 3D, consiste en presentar dos imágenes superpuestas y mediante un par de anteojos, se logra filtrar cierto tinte que llevan cada una de las imágenes. De esta manera nuestro cerebro recibe las dos imágenes, una diferente en cada ojo y el cerebro se encarga de superponerlas, asumiendo que vemos un mismo objeto y entonces dar la sensación de profundidad.

Alioscopy por su parte funciona un poco diferente, en lugar de proyectar un haz de luz por cada pixel, se proyectan en un mismo pixel ocho diferentes, de tal modo que de acuerdo con el ángulo en que se vea la imagen, un mismo pixel proyecta una imagen diferente a cada ojo, logrando el mismo efecto que se tenía con el uso de lentes.

Esto naturalmente conlleva dos problemas adicionales, uno es que cada uno de los ases de luz deben ser calculados de manera independiente y segundo, que la distancia a la que el observador mira la imagen debe ser uno de los parámetros del programa, dado que la posición de los ojos se “calcula” con el movimiento de las cámaras o puntos de enfoque del objeto proyectado.

Por otra parte es importante que cada objeto sea representable en un eje de coordenadas (x,y,z) para poder entonces calcular el espacio que hay entre una imagen y otra o incluso, saber qué partes de un objeto se ven desde una posición y no desde otra.

2.4.3 Sistematización de la generación de contenido 3D-Alioscopy

El render o la generación en tiempo real del contenido de los objetos visualizados en el CAVE, se logra por un proceso ejecutado en los GPU (Graphics Processing Unit), este es ajustado a las características lenticulares de los monitores Alioscopy.

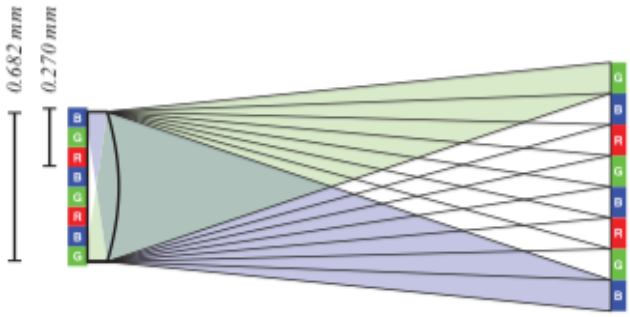


Figura 27. Detalle de una lentícula individual de un monitor Alioscopy

Una matriz de lentes cilíndricos es puesta en la superficie de la pantalla de los monitores Alioscopy, como se muestra en la figura 28. Cada sub-pixel es amplificado por la lenticula y enfocada hacia varias direcciones. Dando como resultado que estos 8 sub-pixels son visibles desde cualquier punto de visión.

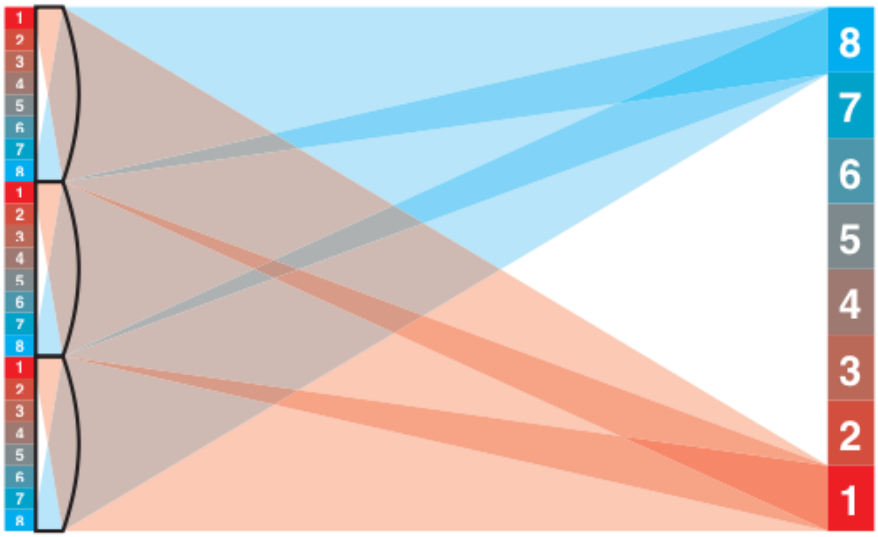


Figura 28. El eje de enfoque individual de las lenticulas (izquierda) causa que todos los sub-pixeles de cada canal, converge a una área común en el plano (derecha)

El renderizar contenido 3D en tiempo real es un objetivo donde la eficiencia es un factor critico. Los sistemas lenticulares poseen una basta cantidad de procesos o cálculos, los cuales son posteriormente unidos en una sola imagen. Hacer un render de 8 vistas del objeto o de la escena, más los costos de entrelazar los sub-pixeles según los parámetros propios a las características de la matriz lenticular del monitor, es un trabajo que demanda muchos recursos del hardware. El desafío de este proceso solo es alcanzable a travez de una programación basada en el GPU.

Este alcance se loga por medio de la generalización de un algoritmo basado en el GPU para render lenticular, presentado por la IEEE Virtual Reality en el año 2007 [Kooima et al. 2007] . Este algoritmo ya ha sido empleado en una variedad de sistemas de visualización y plataformas.

Renderizado de una Escena

Antes de desplegar (proyectar) en el monitor, la escena es previamente renderizada, cada una por canal y es almacenada en un buffer. Luego se crea un ciclo entre todos los canales, enlazando un frame en el buffer con el otro, definiendo la perspectiva apropiada para cada canal. Para finalmente mostrar el objeto que esta contenido en el buffer como uno solo.

Interlazado

El entrelazado de los sub-pixeles del renderizado de una imagen es determinado por sus parámetros. El primero es la posición de las cámaras, donde en el caso más sencillo, estas son distribuidas de una forma equitativa en el plano, separadas una de la otra por una distancia inter-ocular como se muestra en la figura 29.

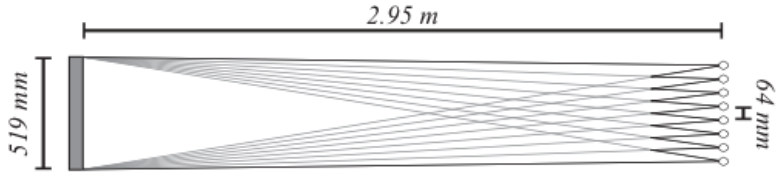


Figura 29. Posición de las vistas (derecha), perspectiva o distancia frustrum usado para renderizar los 8 canales en un Alioscopy (izquierda)

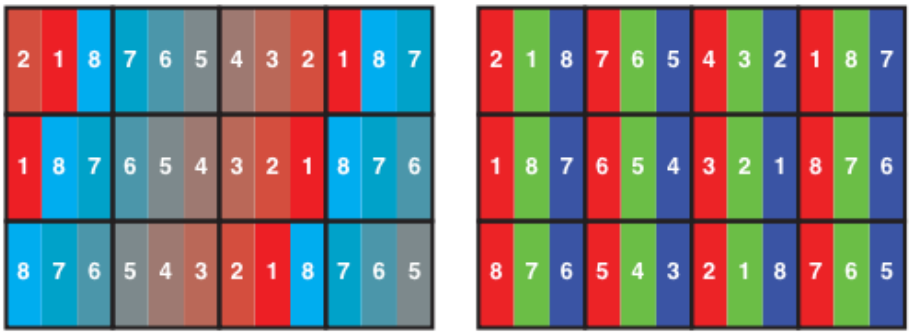


Figura 30. El mapeo de los canales sobre los sub-pixeles, ajustado a 18.435° (izquierda), resulta en una distribución de colores uniforme (derecha)

El segundo parámetro es definido por la naturaleza física de la matriz lenticular del monitor, que descrito de una manera mas sencilla, este tiene 5 términos:

- Pitch: es el tamaño de cada lenticula del monitor (0.682mm), figura 29.
- Angle (angulo): es la rotación de la matriz lenticular relativa a la red de pixeles (18.435°)
- Duty Cycle: es la fracción de una lenticula dedicada a cada canal (1/8)
- Optical thickness (espesor): es la longitud focal efectiva de una matriz lenticular (4.2mm aprox)
- Shift: es el valor de la calibración de izquierda a derecha que ajusta la dirección del enfoque (±0.01)

Clusterización lenticular

La inmersión es una importante propiedad de la realidad virtual, usualmente se alcanzada llenando el campo de visión del usuario con imágenes estereoscopicas. Esta es la principal motivación detrás de ambientes VR de gran tamaño como el CAVE. Pero tomar los beneficios de proyección y escalabilidad de un CAVE normal esta lejos de ser factible para un sistema implementado con tecnología lenticular. Para lograr este objetivo se tiene que aplicar técnicas de software para asegurarse que las funciones estereoscopicas coincidan y aseguren que la tecnología cluster sincronice el contenido.

Super-imposición autoestereoscopica

Normalmente, un display lenticular proyecta sus canales hacia adelante, y así las dos lenticulas adyacentes proyecta sus canales paralelas de una de la otra. Dado una matriz de display lenticular, la mayoría de las posiciones (parámetro shift) se van a repetir de una manera discontinua. El reto con múltiples monitores lenticulares es hacerle mostrar al usuario una sola proyección, donde todos los canales estén correctamente alineados en el plano de enfoque.

Para lograr esto, se escoge uno de los monitores, para ser el central y el nodo primario de este (figura 31). Todos los lóbulos de los otros monitores, son desplazados y alineados al central.

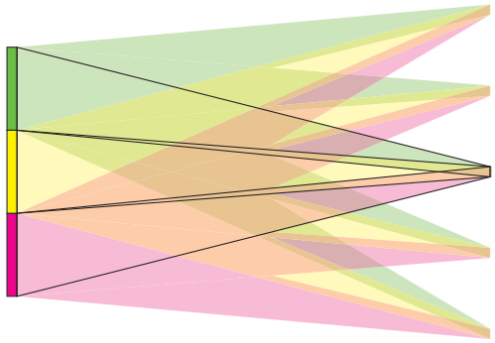


Figura 31. El mosaico de monitores lenticulares (izquierda) es logrado por alinear el nodo primario de todos los monitores al plano del enfoque (derecha)

Sincronización del cluster

Para una matriz de monitores se muestre como un único monitor, el render de la escena debe de ser perfectamente sincronizado. El monitor adyacente debe tener una sincronización extremadamente precisa para que el contenido se vea continuo. Cualquier retardo en la actualización, vendría a desordenar la proyección y por lo que una sincronización basada en el hardware es necesaria.

El CalVR como el núcleo del cluster, viene a tomar todas las prestaciones de hardware para lograr esto. Lo que hace es tomar la taza de refrescamiento del monitor, normalmente 60Hz y la sincroniza con todos los nodos. Esto es logrado basándose en la funciones de las tarjetas de video nVidia G-Sync. Cada nodo corre una copia independiente de la aplicación de renderizado (CalVR - Interleaver Figura 5) con sus respectivas configuraciones. El estado de sincronización y la actualización de la coordinación de la proyección se logra a travez del protocolo TCP.

2.4.4 Metodologías de comunicación entre dispositivos remotos y el clúster

Finalmente en esta última etapa se procedió a generar un protocolo y una codificación capaz de enviar mensajes desde dispositivos remotos hacia el servidor y del servidor hacia los nodos. Los dispositivos remotos que se probaron para esta fase fueron laptops, iPhone 4, iPad mini y iPad air (figura 32).

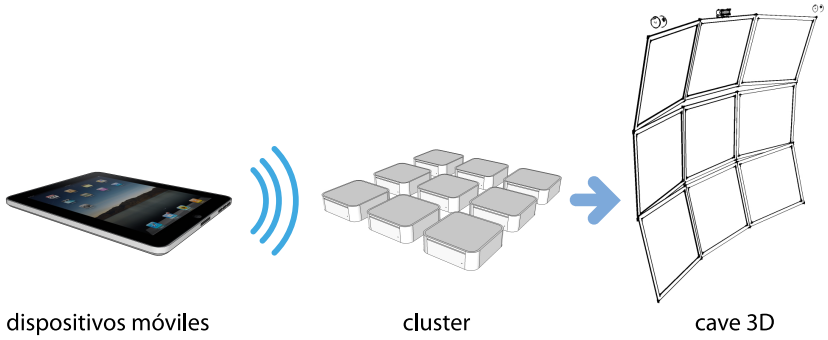


Figura 32. Proceso de comunicación entre la interface y el cave

Inicialmente se procedió a realizar modificaciones en el servidor, para que este fuera capaz de manejar 2 tipos de clientes; el cliente nodo y el cliente remoto. Esto se debe a que dependiendo del tipo de nodo, el servidor realiza actividades distintas al recibir mensajes. Mientras que para el cliente nodo su único mensaje es decir que es un nodo, para el cliente remoto, los mensajes que envía el usuario son reenviados a cada uno de los clientes nodo conectados al servidor.

Seguidamente se procedió a determinar cómo se iba a enviar la información. El enviar solo una cadena de *strings* con los datos fue descartado ya que complica mucho la interpretación de la información recibida. Se debe de tener mucho cuidado con la información que se envía, en qué orden y cómo se va a descifrar esa información.

Inicialmente se intenta en enviar la información en archivos xml, sin embargo para realizar esto se debía pasar por varias fases como: crear el archivo, agregar los elementos, pasar esto a cadena de bytes para enviarlos en el stream de salida y finalmente cuando se recibe en el nodo se debía realizar el proceso inverso. Todo este proceso parecía largo y complejo para enviar cadenas de información que en muchos casos, como el giro de una imagen implicaba enviarlas de manera rápida y en tiempo real.

De esta manera la información que se debía enviar, debía ser una en la cual la información tuviese características de encapsulamiento. Entonces se implementa un *struct*, ya que es un elemento del lenguaje C que puede contener datos. Al enviarlo por el *socket* y recibirlo al otro lado se facilita manejar la información del dato recibido, de manera que se estableció que los clientes remotos envían *structs* de información.

Finalmente en el caso de dispositivos móviles, se utilizó la plataforma de Xcode para desarrollar el código necesario. El equipo de e-Science ya había trabajado en una aplicación inicial. Aprovechando esta aplicación se procedió a entender la sintaxis del lenguaje objective C ya que iOS trabaja con este lenguaje. Basándose en la documentación de *Apple developer* se desarrolló el código para enviar *streams* de datos al servidor. Al ser objective C un lenguaje basado en C, tiene la característica de que

comparte las mismas estructuras básicas de C, de manera que se puede mantener la consistencia de datos entre una plataforma y otra.

Con esta última etapa realizada, se logró obtener una metodología comprobada que permite establecer una comunicación entre dispositivos remotos y un clúster de visualización, utilizando tanto dispositivos móviles como laptops, iPads o iPhones.

2.5 Tareas desarrolladas por los asistentes

A lo largo de este año se ha contado con el apoyo de cuatro asistentes, Hugo Álvarez, Julio Vargas, Jean Carlo y Daniel Cortés. A cada uno se le han asignado tareas diferentes de las muchas necesidades del proyecto. A continuación se detalla una tabla con las diferentes labores que cada uno ha realizado (en muchas ocasiones, han estado trabajando en conjunto aplicando técnicas de eXtreme Programming (XP) para agilizar el trabajo)

Asistente	Tareas realizadas(periodo 2013)
Hugo Álvarez*	- Modulación del código fuente. - Adaptación del código fuente de Windows a Linux. - Investigación de librería GLUT en OpenGL. - Adaptación de ejemplos OpenGL a Alioscopy 3D. - Despliegue de animaciones de OpenGL en varios monitores. - Instalación del Sistema Operativo Rocks 5.5 - Mantenimiento de Rocks.
Julio Vargas*	- Instalación Rocks - Reparación SO Rocks - Configuración Rocks - Instalación aplicaciones ambiente - Desarrollo API manejo 3D alioscopy - Distribución de imágenes en múltiples pantallas - Configuración Fibra Óptica - Adaptación aplicaciones 3D con API 3D alioscopy - Desarrollo aplicación para control remoto de nodos - Desarrollo API comunicación Front-End, con nodos - Control centralizado de acciones (funciones usuario)
Daniel cortés	- Instalación de paquetes al servidor. - Actualización y configuración del Frontend. - Instalación de Rocks en los nodos. - Programación de ejemplo sencillo en 3D usando OpenGL. - Investigación de MPI para sincronizar los procesos entre nodos (topando con pared por problemas de incopatibilidad). - Cotización de memorias para los nuevos nodos. - Corrección de errores y mejora de ejemplo de pirámide en 3D (con la resolución). - Ejecución de archivos de ejemplo “.play”. - Instalación de OpenSG para generación de imágenes en un clúster de visualización. (pendiente de finalizar) - Uso de VNC Server para mejor manejo de los nodos.
Jean Carlo **	- Configuración de la red de fibra óptica - Adaptación de los sockets - Búsqueda de información sobre OpenGL - Configuración de Rocks - Instalación de las interfaces a los nodos - Pruebas de comunicación entre los nodos y el front-end

Asistente	Tareas realizadas (periodo primer semestre 2014)
Alejandro Vargas Chaves	Configuración del equipo que consistió en evaluar el estado del hardware e instalación del software. Generar un control entre el front-end y los nodos, esto consistió en desarrollar toda una metodología para que las simulaciones se visualizaran de manera distribuida y se ejecutaran desde el front-end. Finalmente generar una metodología para la comunicación entre dispositivos remotos y el front-end que esto involucro tanto el uso de laptops y el iphone.
Edgar Rojas Muñoz.	- Puesta en operación de un clúster para visualización de contenido 3D con pantallas auto estereoscópicas. - Investigar metodologías de creación de contenido en 3D, profundizando en los métodos referentes a OpenGL. - Investigar las metodoogías de creación de contenido Real-3D utilizando un shader especializado en la tecnología Alioscopy. - Generación de contenido en 3D y Real-3D.
Johnny Romero Lépiz	Creación de metodologías de visualización y corrida de ejemplos en el cluster 2D Prueba y validación de Rocks y SAGE

*Las labores de Julio Vargas y Hugo Álvarez son muy similares porque han estado trabajando con la metodología de extreme programming (XP).

**Jean Carlo sólo participó durante el primer semestre 2013

3. Metodología utilizada

Desde el punto de vista metodológico se desarrolló al igual que en proyectos previos, cada objetivo en forma secuencial con la idea de poder cumplir uno por semestre.

Primer semestre: Diseñar la estrategia de visualización y control de los datos

En esta primera etapa se estudiaron todos los formatos que se han estado usando para el proyecto eBridge. Se identificó qué posibilidades había de utilizarlos tal como están disponibles, o de traducirlos en un formato que pueda ser utilizado en el *core* que se implementaría en el siguiente semestre.

También se definió el equipo exacto que se necesita para armar la estación con la que se podrá realizar las pruebas de las hipótesis de diseño en las siguientes etapas. Las tareas fueron desde identificar exactamente cuáles equipos se necesitan, como el armado físico más la conexión eléctrica y de datos de cada componente. Por supuesto que las cotizaciones y trámites asociados a las compras formaron parte de esta etapa, así como la preparación del informe del segundo semestre.

Segundo semestre: Desarrollar las sub-rutinas necesarias para poner en práctica las estrategias diseñadas en el objetivo anterior.

En esta etapa se pasa al diseño y programación de un primer CORE de rutinas que pueda interpretar los datos. Es decir, una vez estudiados los formatos en que se encuentran los datos y su posible uso (ya sea directo o por traducción) se procede a diseñar y programar el corazón del sistema.

Con estas rutinas listas se desarrollaron los dos prototipos uno fue usado para discutir los resultados con los integrantes del proyecto al que se le da servicio (eBridge), lo que resultó en un *focus-group* que en realidad estaba planeado para la cuarta etapa. El prototipo fue evaluado tanto desde el punto de vista de la viabilidad tecnológica, como de su eficiencia de uso. ¿Funcionan?,¿Son lo suficientemente intuitivas?, ¿Se ofrece al usuario control suficiente en el ambiente?

El segundo prototipo se usó para el 6to encuentro de investigadores en abril del 2013, en esta ocasión se uso el nuevo sensor Leap Motion en el prototipo y tuvo excelentes resultados.

Tercer semestre: Implementar las sub-rutinas en una aplicación funcional que se pueda correr en forma autónoma

Para este momento se desarrolló una primera aplicación tanto del CORE programado en la segunda etapa, como de la interface desarrollada. La idea fue crear una interface de trabajo en donde un investigador pueda cómodamente evaluar datos. En esta etapa se trata de implementar las sub-rutinas que se desarrollaron en la etapa anterior.

Finamente se realiza la selección del primer grupo de herramientas que se le ofrecerá al usuario del sistema para el control de los datos tridimensionales y bidimensionales y se implementan en un único sistema funcional.

Cuarto semestre: Conexión de todos los elementos

En el plan inicial en esta etapa se hacía el *focus group*, pero eso este se realizó en la segunda etapa con el prototipo y no con la aplicación lo que dio la retroalimentación que se necesitaba.

En esta etapa final fue necesario por supuesto, armar todos los equipos y conectar los elementos físicamente separados como el tdw, el cave y el iPad y por supuesto las acciones administrativas necesarias como este informe.

4. Resultados

En investigaciones de tipo tecnológico, el resultado es el sistema funcionando, el sistema funciona perfectamente con los requerimientos que se establecieron al inicio del proyecto. De este modo el sistema desarrollado actualmente cuenta con los siguientes componentes en plena función:

4.1 Interface

Interface a través de una aplicación que corre en iPad respondiendo a las necesidades del proyecto eBridge. La aplicación puede usarse tanto ella sola como en conjunto con el “cave” y el TDW funcionando en este caso como un control remoto de estos equipos.

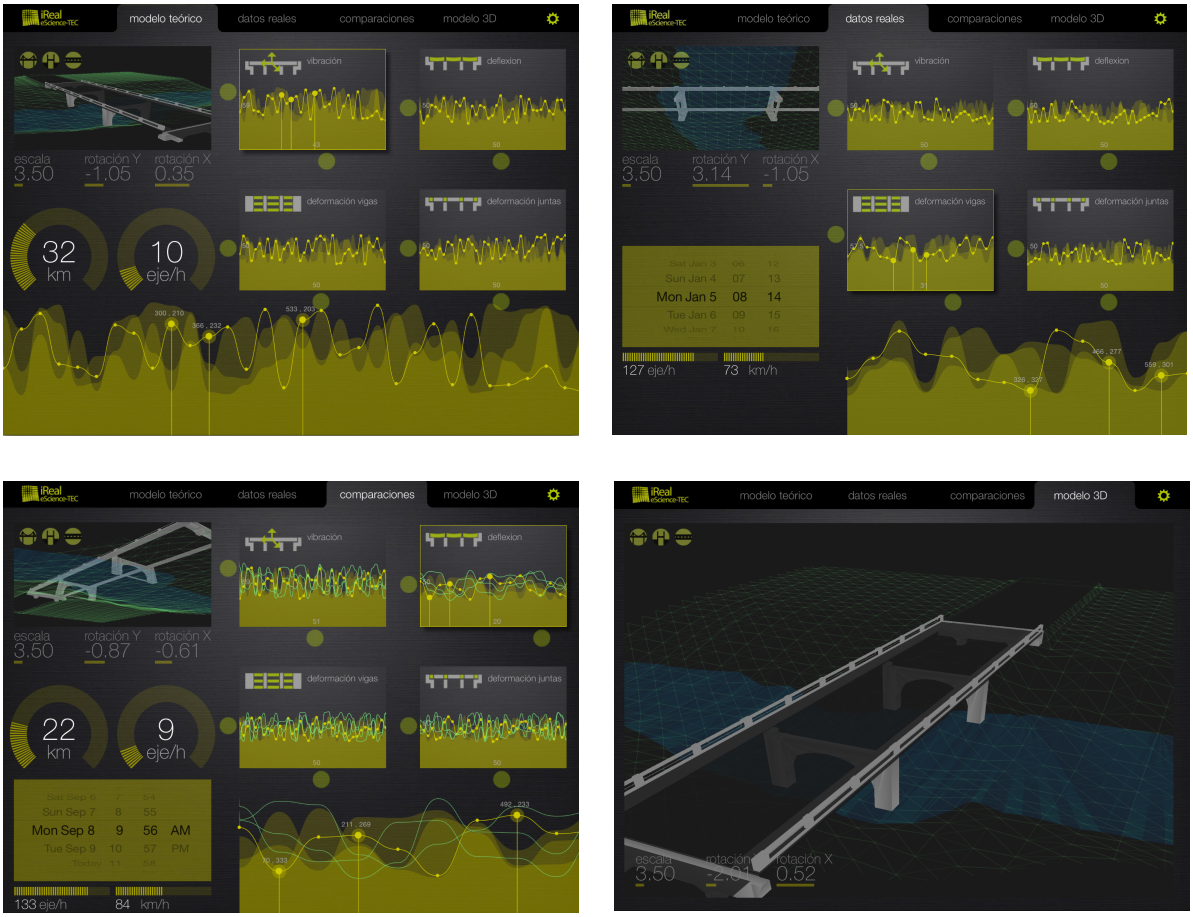


Figura 33. Los cuartos escenarios de la aplicación para iPad. En el primero vemos los datos del modelo teórico, en el segundo se ve cómo se manejan los datos históricos, salidos de los sensores. la cuarta imagen muestra el escenario de comparación entre el escenario teórico y los datos reales y el cuarto escenario muestra la aplicación anterior cuya utilidad es servir de “control remoto” para la navegación en el “cave”

4.2 Comunicación entre la interface móvil y el "*cave*" de visualización.

En este caso se logró la comunicación entre los equipos móviles y el clúster, lo mismo que la comunicación distribuida entre este último y el cave.

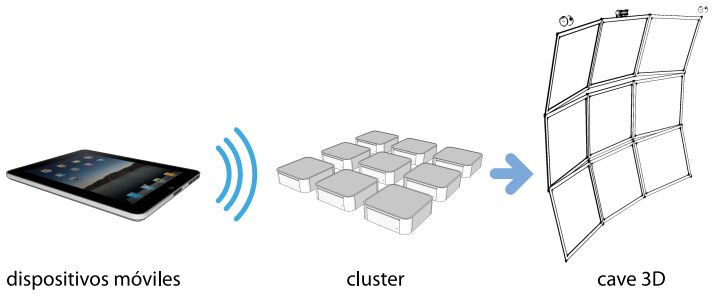


Figura 34. Proceso de comunicación entre la interface y el cave

4.3 Despliegue de datos

Sistema de despliegue de los datos de visualización con una configuración robusta orientada a objetos y que permite el despliegue de contenido 3D Alioscopy.

Figura 35. Sistema de visualización inmersiva

4.4 Administración del clúster

Sistema eficiente a nivel de software encargado de la administración del clúster y el contenido distribuido en el "cave".

Figura 36. Sistema de visualización inmersiva

El sistema ya fue presentado ante ante la comunidad y se detalló sobre las potencialidades y posibilidades de uso, especialmente para otros proyectos dentro y fuera del TEC.

5. Discusión y conclusiones

Por la naturaleza del trabajo de I+D, las discusiones de las acciones se han venido mostrando a través de todo el informe, así como las conclusiones que se tomaron para solventarlas.

En el sentido más general se podría agregar:

- Este tipo de ambientes de simulación a gran escala, puede ser de mucha utilidad para una gran variedad de investigaciones tanto a nivel interno como externo. Muchas de las cuales serían mucho más difícil de hacer (a veces imposible) sino se cuenta con la capacidad de cálculo y visualización del tipo que se desarrollaron en este proyecto.
- Un gran número de fenómenos son más fáciles de analizar y comprender cuando se pueden mostrar en simulaciones a escala natural, contando con un nivel de detalle que permita deducir el comportamiento del sistema. Con ello se puede, no solo dar soporte a los investigadores, sino también a la docencia de la ciencia y la ingeniería.
- En el proyecto iReal 2.0 estamos motivados por compartir estos conocimientos y facilidades con la comunidad científica y ponerlos al servicio del TEC y del país.

6. Actividades de divulgación

6.1 Almuerzo académico

A nivel interno, se participó en un Almuerzo Académico organizado por la VIE, en el cual se realizó una presentación del road map del grupo de investigación de iReal y de las posibilidades de visualización que ofrece el laboratorio para proyectos internos.

6.2 Colaboración con la CCSS

Por las característica del laboratorio y su estado actual, fue utilizado para recibir una comitiva de personeros de la **CCSS**, y en este actividad se aprovecho para mostrar las potenciales del laboratorio y del grupo de investigación.

Figura 37. Presentación ante delegación de profesionales de la CCSS

6.3 Visita de Mike McGregor

En septiembre del 2013 se tuvo la visita de Ph.D. Mike H. MacGregor del Department of Computing Science de la Universidad de Alberta Canadá, el cual realizó una evaluación del programa eScience y una valoración de los proyectos que lo conforman incluyendo al nuestro.

Dentro de sus principales aportes está la solicitud de que debíamos enfocar lo objetivos del grupo, de allí se realizaron algunas sesiones de discusión y se terminó con el Road Map del grupo iReal.

Las siguientes son algunas de las imágenes de su visita:

Figura 38. Visita de Ph.D. Mike H. MacGregor

6.4 Visita del embajador de Costa Rica en Alemania

El 28 de julio del 2014 se recibió en el laboratorio de eScience al señor embajador de Costa Rica en Alemania, Sr. José Joaquín Chaverri Sievert. Le fue presentado el programa de investigación eScience así como este proyecto, al mismo tiempo se le ofreció al embajador una lista de posibles socios de investigaciones en Alemania con el fin de que formaran parte de su agenda como embajador en Berlin.

6.5 Visita del director regional DAAD

El 26 de agosto del 2014 se recibió la visita del Director del Centro de Información para Centroamérica del Servicio Alemán de Intercambio Académico (DAAD), Michael Eschweiler, M.A.
La idea de la visita fue demostrar el trabajo que se está haciendo y explorar nuevos modos de cooperación con la República Federal Alemana. Se encontraron varios nuevos mecanismos a través de los cuales se puede buscar financiamiento en ese organismo.

6.6 Visita de la Ministra de Ciencia y Tecnología

En octubre 16 del 2014 se tuvo la visita de la Ministra de Ciencia y Tecnología M.Sc Gisela Kopper Arguedas, esto como parte de la visita del presidente y algunos de sus colaboradores a la Institución. A pesar de la poca disponibilidad en sus agendas, se planteó como visita única al lab de eScience. Allí M.Sc Luis Alexander Calvo, director de CIC en ese momento y coordinador del grupo eScience les dio un panorama rápido sobre la labor del CIC y los proyectos realizados en eScience. Como caso particular se hizo una explicación detallada de las capacidades del laboratorio y del proyecto iReal 2.0.

Figura 39. Visita de la Ministra de Ciencia y Tecnología

6.7 Visita de Universidad de San Buenaventura en Cali Colombia

El 3 de noviembre del 2014 se tuvo la visita la Master Rosmery Dussán Aguirre, directora del centro de desarrollo en Moda de la Universidad de San Buenaventura en Cali Colombia.

La idea de la visita fue mostrar lo que se está haciendo en iReal y buscar posible puntos de intersección y colaboración entre nuestros trabajos.

6.8 Artículos

Hernandez-Castro, F. Monge-Fallas, J. 2014. **Dibujando 2D en iOS**. Tecnología en Marcha. Volumen 27, No. Especial 2014. Editorial Tecnológica de Costa Rica.

Hernandez-Castro, F. Monge, J. 2013. **iReal: Posibilidades**. Investiga.TEC, Volumen 16, Número 16. Febrero 2013. Instituto Tecnológico de Costa Rica.

Hernandez-Castro, F. Monge, J. 2013. **Real: Visualización de información**. Investiga.TEC, Volumen 15, Número 15. Enero 2013. Instituto Tecnológico de Costa Rica.

6.9. Participación en el 6to encuentro de investigadores

Como actividad complementaria, el proyecto expuso en el 6to encuentro de investigadores el mes de abril del 2014. Para esta ocasión se desarrolló un stand en el cual se enfatizó en algunas interfaces intangibles las cuales son exploradas en el proyecto con el objetivo de implementarlas posteriormente en el cave. De este modo se utilizó un sensor del tipo Leap Motion para controlar el segundo prototipo de la aplicación iReal 2.0.

Las siguientes son algunas imágenes durante el desarrollo del evento:

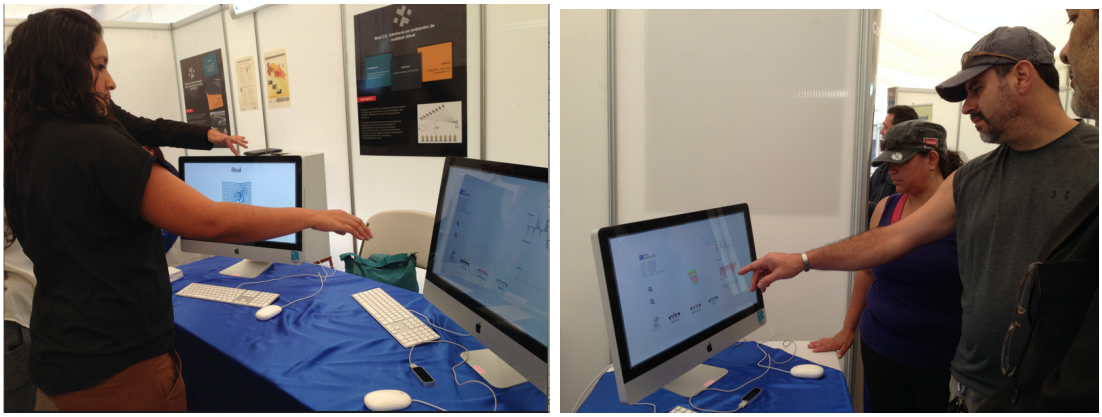


Figura 40. Participación en el ato Encuentro de Investigación

6.10. Descarga de la aplicación iReal

Un resultado de divulgación muy interesante es la cantidad de descargas que se han dado. En estos tres años de estar disponible la aplicación se ha descargado más de 7,700 veces, sin duda, un interesante nuevo modo de divulgar proyectos de esta naturaleza.

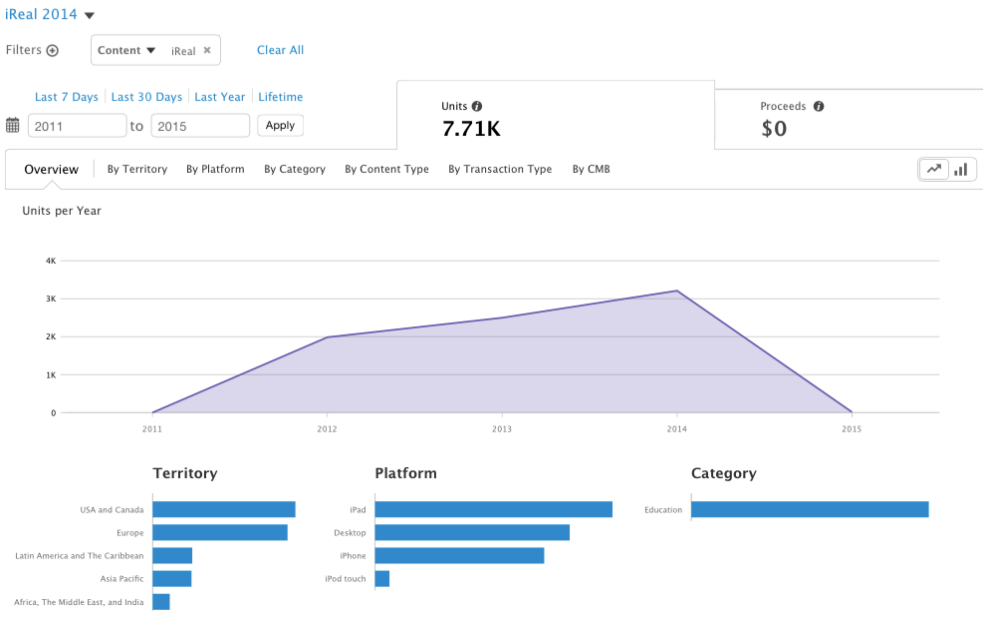


Figura 41. Reporte del App Store mostrando la distribución de las más de 7,700 descargar de la aplicación iReal en los 4 años que ha estado a disposición.

7. Limitaciones o problemas encontrados

Los problemas principales en el proyecto hasta ahora son dos:

El primero está relacionado con el mantenimiento del equipo adquirido en iReal 1.0. Habían algunos componentes del "*cave*" que se habían dañado, y tanto el tiempo de especificación, como la compra del mismo, han retrasado la configuración completa del sistema.

El segundo problemas está relacionado, por segunda vez, con el cambio de investigador en el área de computación. Dado que la administración y configuración del "*cave*" no es materia de dominio general, ha llevado un tiempo considerable para que el investigador se contextualice en la problemática. Además el dominio de OpenGL es fundamental y actualmente no hay en el TEC profesiones con una experticia en este campo, por que lo que nuestro investigador ha tenido que auto-capacitarse al respecto.

Sin embargo, las dificultades fueron superadas y finalmente se ha podido generar el contenido Alioscopy y la coordinación de los diferentes equipos para visualizar los datos en cuestión.

8. Bibliografía

Brown, M., Leigh, J., Renambot, L. , Nam, S., Jagodic, R., Hur, H., Long, L. & Jeong, B. **SAGE: Scalable Adaptive Graphics Environment Persistent Visualization and Collaboration Services for Global Cyberinfrastructure**. Presentado en Pragma 18. San Diego, 2010.

Chikama, M., Kadobayashi, R., Fukunaga, K. & Shimojo, S. (2010) **Browsing Digital Heritage on an Interactive Tiled Display Wall (I-TDW) with Smartphone Interface.** Presentado en Pragma 18. San Diego, 2010.

DeFanti,T., Dawe, G., Sandin, G., Schulze, J., Otto, P. ,Girado, J., Kuester, F., Smarr, L. & Rao, R. (2009). **The StarCAVE, a third-generation CAVE and virtual reality OptIPortal**. FGCS, The International Journal of Grid Computing: Theory, Methods & Applications. 2009. Pages 169-178

DeFantia, T., Leighb,. Renambotb, L., Jeongb, B. Verlob, A. and others. (2009). **The OptIPortal, a scalable visualization, storage, and computing interface device for the OptiPuter.** FGCS, The International Journal of Grid Computing: Theory, Methods & Applications. 2009.

Fujiwara, Y., Ichikawa, K., Date, S. & Takemura, H. (2010). **A Control Mechanism of Multiple Visualization Applications on SAGE-enabled TDW**. Presentado en Pragma 18. San Diego, 2010.

O'Hagan, R. & Zelinsky, A. (2000). Visual Gesture Interfaces for Virtual Environments.Co-operative Research Centre for Advanced Computational Systems The Australian National University. 2000

Lau, C., Levesque, M., Chien, S.,Date, S. & Haga, J. (2010). **ViewDock TDW: Rapid Visualization of Virtual Screening Results.** Presentado en Pragma 18. San Diego, 2010.

Meng, H., Pears, N. & Bailey, C. (2007). A Human Action Recognition System for Embedded Computer Vision Application. Publicado en el 2007 en la IEEE.

Nirnimesh, Harish, P. & Narayanan, J. (2010). **Garuda: A Scalable, Tiled Display Wall Using Commodity PCs**. Center for Visual Information Technology International Institute of Information Technology Hyderabad, 500032 INDIA. Presentado en Pragma 18. San Diego, 2010.

Nan, K., Dong, K., Zheng, Y., Li, W., Tilak, S. & Simas, T. **Collaboration Environment for e-Science: Duckling and Applications.** Presentado en Pragma 18. San Diego, 2010.

Sy, R. (2010). **Visualization and Collaboration: Tiled Display Wall System**. Osaka University. Presentado en Pragma 18. San Diego, 2010.

Núñez, S., Barrantes, G., Malavassi, E & Brenes, J. (2010). **NG-TEPHRA: Enabling Large-Scale Volcanic Hazard Simulations in the PRAGMA Grid Environment**. PRAGMA WORKSHOP. SAN DIEGO, CA, USA. MARCH 3-4 2010

Kooima, R.,Prudhomme, A.,Schulze, J., Sandin, D., DeFanti, T., (2010). **A Multi-viewer Tiled Autostereoscopic Virtual Reality Display.** Center for Computation & Technology, Louisiana State University (LSU), California Institute for Telecommunications and Information Technology, University of California San Diego (UCSD), Electronic Visualization Laboratory, University of Illinois at Chicago (UIC)

Seligmann., D.(2007). **Computer Vision and Art**. Artful Media, Published by the IEEE Computer Society, April-June 2007.

Apple Inc. (2011) **Drawing and Printing Guide for iOS: Graphics & Animation: 2D Drawing**. Cupertino, CA.

Hernandez-Castro, F. Monge, J. (2011). **Flujo de control en iOS.**Tecnología en Marcha. Volumen 25, No. 5. Editorial Tecnológica de Costa Rica.

Kooima, R., Peterka, T., Girado, J., GE, J., Sandin, D., y Defanti, T. 2007. **A GPU Sub-pixel Algorithm for Autostereoscopic Virtual Reality.** In *IEEE Virtual Reality Conference, 2007. VR'07*, 131–137.