ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

"DESARROLLO DE UN PROTOTIPO DE UN API PARA LA INTERACCIÓN DE UN USUARIO CON APLICACIONES CON CONTENIDO 3D UTILIZANDO KINECT"

INFORME DE PROYECTO DE GRADUACIÓN

Previo a la obtención del título de:

INGENIERO EN CIENCIAS COMPUTACIONALES ESPECIALIZACIÓN SISTEMAS MULTIMEDIA

Presentado por:

Andrés Estuardo Prieto López

GUAYAQUIL – ECUADOR

2013

AGRADECIMIENTO

Agradezco a Dios por ser mi guía, a mi familia por todo su apoyo incondicional. También a todas las personas que una de otra forma me ayudaron, sin ellos no hubiera sido posible llegar hasta este punto de mi carrera.

Andrés Prieto López

DEDICATORIA

Dedico este trabajo a Dios por ser guía de mi vida.

A mi familia por su apoyo incondicional.

TRIBUNAL DE SUSTENTACIÓN

Boris Vintimilla Ph.D.
PRESIDENTE

Sixto García Aguilar Ph.D.
DIRECTOR DEL PROYECTO DE GRADUACIÓN

Daniel Ochoa Donoso Ph.D.
VOCAL PRINCIPAL 1

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de este informe del proyecto de graduación, me corresponde exclusivamente; y el patrimonio intelectual de la misma a la Escuela Superior Politécnica del Litoral"

(Reglamento de exámenes y títulos profesionales de la ESPOL)

_____

Andrés Prieto López.

RESUMEN

En el presente trabajo se desarrolló un conjunto de librerías, que permite una interacción con el computador, haciendo uso de las manos y gestos como forma de entradas para generar comandos.

Para desarrollar este proyecto se utilizo la tecnología del Kinect de Microsoft, este sensor de profundidad permite determinar la distancia de los objetos a la cámara, en particular la posición de la persona y de sus manos.

Se analizaron diferentes tipos de interacción, para la utilización de los gestos más comunes y más utilizados, que ya han sido adoptados por las personas en las interfaces táctiles. Este proyecto se puede utilizar para manipular objetos tridimensionales como modelos arquitectónicos de edificio, autos, muebles, cuerpo humano, órganos, etc.

Al final de la implementación del proyecto se procedió a realizar las pruebas de usabilidad. Para realizar las pruebas del API, se desarrolló una aplicación que utiliza el API para mover un objeto 3D y se realizó un cuestionario en donde se pedía calificar la experiencia, en una escala del 1 al 5 de acuerdo a factores como la comodidad, la facilidad de uso y la exactitud.

El análisis de los resultados demostró que la metodología utilizada fue la correcta, haciendo que la interacción sea natural con este tipo de objetos en 3D.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Desde la salida al mercado de los primeros computadores personales en la década de los 80´s, el teclado y el ratón ha sido el principal medio de comunicación entre el ser humano y la máquina, permitiendo el desplazamiento del cursor en dos dimensiones.

Actualmente la tecnología ha avanzado enormemente y se han desarrollado nuevas aplicaciones con interfaces o entornos en 3 dimensiones (3D), como por ejemplo en realidad virtual, realidad aumentada, navegación en modelos arquitectónicos, etc. donde la utilización del teclado y del ratón tiene sus limitaciones.

Para mejorar la interacción en un entorno 3D utilizaremos nuestras manos y un dispositivo como el Microsoft Kinect que permite sensar profundidad. El objetivo de este proyecto es utilizar esta tecnología para implementar un API donde integramos las librerías del Kinect con las de procesamiento digital de imágenes obteniendo nuevas funciones que nos permita el desarrollo de una aplicación 3D con esta nueva forma de interacción.

CAPÍTULO 1

1. JUSTIFICACIÓN Y ANÁLISIS DEL PROBLEMA

1.1. Análisis del problema

El avance en capacidad de procesamiento de los computadores ha permitido que las aplicaciones vayan evolucionando con el tiempo y puedan tener mayor detalle gráfico. Con estas nuevas formas de visualización en pantallas más grandes, interfaces en ambientes virtuales, etc., utilizar un teclado [9] o un ratón [10] no son los más adecuados (ver Figura 1.1 y Figura 1.2).

Los dispositivos de entrada con que se interactúan también necesitan ser diferentes para adaptarse al entorno nuevo de realidad virtual y entornos 3D. Aunque el teclado y el ratón se han vuelto inalámbricos, es poco útil en ciertas ocasiones como cuando uno se encuentra de pie y trabajando de manera colaborativa en un mismo entorno.

La utilización de un dispositivo que sense el movimiento natural de las personas, como señas y gestos de las manos, es conveniente como medio de comunicación con la aplicación. Con esto se sugiere una nueva interface para interactuar con el computador que es por medio de gestos.

Figura 1.1 Dispositivo de entrada imagen de un teclado Televideo 925 [9]

Figura 1.2 Dispositivo de entrada un ratón de la época de los 90 [10]

1.2. Justificación

Al analizar el problema de cómo poder comunicarse con una aplicación en ambiente virtual mediante una interacción más natural, el seguir utilizando dispositivos que requieren en algunos casos que la persona esté sentado, o que apoye los dispositivos sobre una superficie, no es muy útil en especial en entornos de visualización de tres dimensiones. Para esto hay otros métodos de interacción que se pueden utilizar, como el de un dispositivo que sense el movimiento de las manos para interactuar con las aplicaciones en 3D, implementando un conjunto de funcionalidades que faciliten el desarrollo de gestos para este tipo de aplicaciones, como por ejemplo manipular con las manos modelos 3D en la aplicación y controlar el sistema operativo Microsoft Windows.

1.3. Propuesta de solución

Se utilizó el Microsoft Kinect [11] (ver figura 1.3), porque reduce problemas, como la generación de ruido en la imagen, en ambientes con poca luz y además de poder determinar la posición de los objetos en este caso de la persona sus manos. Podemos apreciar en la figura 1.4 como está constituida las partes del Kinect.

Al determinar la posición de la persona, se puede conocer donde se encuentran las manos, con esto se limita el frustum [12] (ver Figura 1.5) de la proyección del haz de luz infrarroja del Kinect para que capture solo la

sección de las manos; por eso es necesario que sea en un lugar abierto para que no haya obstáculos entre el Kinect y la persona.

Luego de capturar el movimiento de las manos y determinar la posición en el espacio (x,y,z) sabiendo si se mueve hacia arriba, hacia abajo y la profundidad con respecto a la posición del Kinect como referencia, con estos datos se almacena este frame en un buffer [13], de este instante de tiempo para que después de unos x milisegundos vuelva a capturar la siguiente imagen el Kinect y saber la posición de la mano y comparar estas imágenes, con esto se puede determinar que acción tomar o realizar, la técnica utilizada se llama motion detection [35].

Al saber que acción realizar podemos hacer las respectivas funciones en la aplicación para poder mover el objeto 3D.

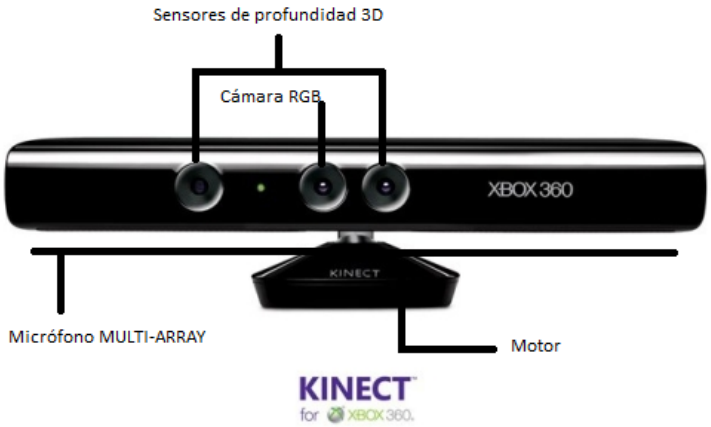


Figura 1.3 Muestra del Microsoft Kinect de la Xbox 360 [16]

Figura 1.4 Partes que conforman el Kinect de la Xbox 360 de Microsoft [17]

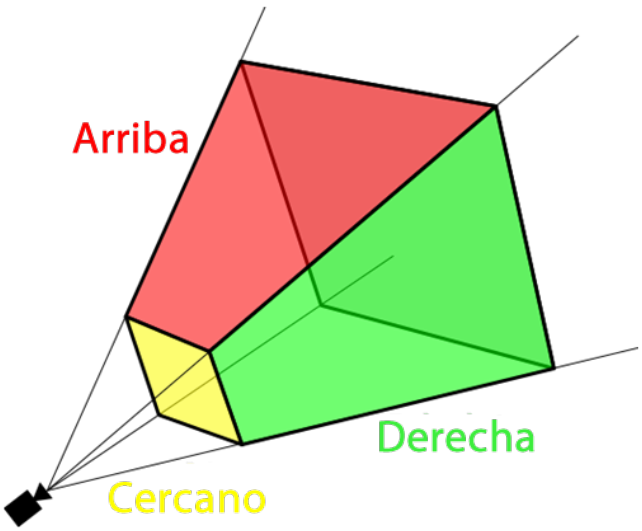


Figura 1.5 Imagen del frustum (sección coloreada) en gráficos por computadora [18]

CAPÍTULO 2

2. REVISIÓN DE LA LITERATURA

2.1. Microsoft Kinect

Luego del lanzamiento al mercado el 4 de Noviembre del 2010, el Kinect que fue creado por Alex Kipman y desarrollado por Microsoft para la consola Xbox 360, no tardaron más que un par de días para que en internet estuvieran disponibles unos controladores que permitieran utilizar el Kinect en el Microsoft Windows. Inmediatamente una comunidad de desarrolladores comenzó a poner a disposición de otros usuarios y desarrolladores, diversas aplicaciones que utilizaban el Kinect sin necesidad de la consola Xbox, posteriormente estuvieron también disponibles controladores para Linux y Mac OS. En un principio Microsoft pensó en tomar medidas contra estos desarrolladores pero luego de analizarlo cuidadosamente decidieron apoyar a la comunidad para que siga

desarrollando y competir con ellos, sacando su propio SDK que sólo funciona en Windows OS.

El kinect consta con una cámara VGA (Video Graphics Array o en español una secuencia de gráficos de video) infrarroja que detecta una malla de puntos que proyecta el kinect y una cámara VGA estándar (ver Figura 1.4) para capturar video a 30 cuadros por segundo con una resolución de 640 por 480 pixeles.

EL profesor británico Andrew Blake, director general de Microsoft Reserch en Cambridge, lideró un equipo de 5 personas quienes desarrollaron el software del kinect, para reconocimiento del cuerpo humano que se usa en los juegos de la Xbox 360 de Microsoft, se tomaron un poco más de 2 años en el trabajo desde el concepto hasta el producto final. Fue premiado en Londres con el MacRobert Award que otorga el ROYAL Academy of Engineering por el trabajo realizado con el software del Kinect en 2011.

Entre las implementaciones que se han realizado utilizando el Kinect, que han servido de guía para realizar este proyecto contamos con:

Reconocimiento de gestos usando Kinect

El la figura 2.1 muestra un sistema que reconoce gestos de números, desarrollado por Harishankar Narayana[25]. Para esta aplicación se utilizaron los controladores del Kinect para Linux y librerías libres libfreenct [5] o OpenNI [20].

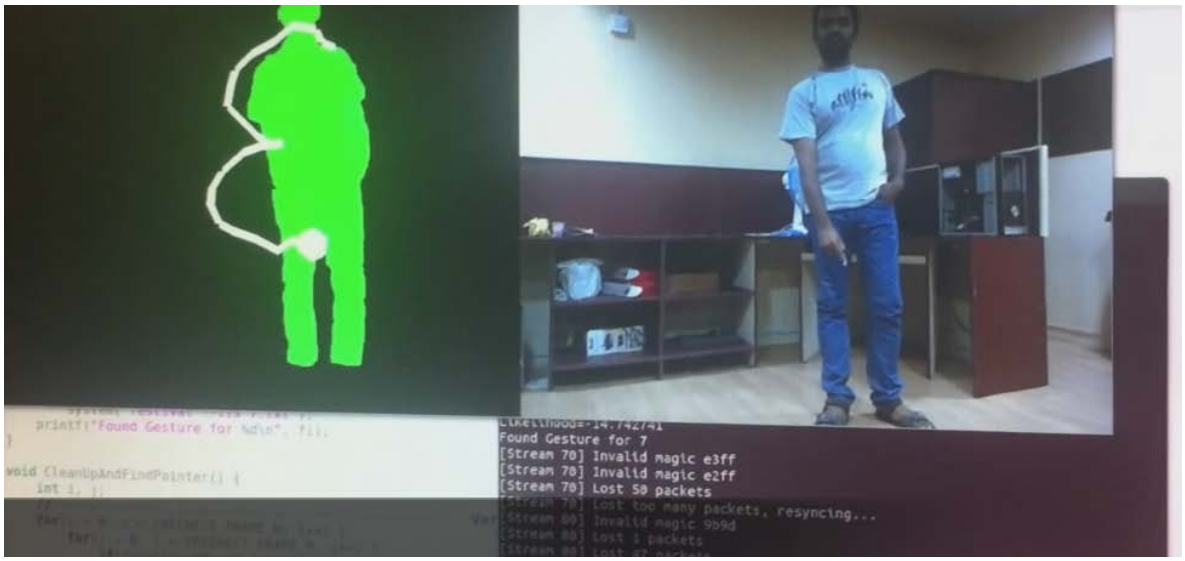


Figura 2.1 Muestra del reconocimiento del número 3 en este proyecto

Control del Windows 7 utilizando el Kinect

La empresa alemana Evoluce [26] que se especializa en tecnología en pantallas multi – táctiles, desarrolló una aplicación donde se puede jugar solitario y mover imágenes en el sistema operativo Windows 7 (ver Figura 2.2), utilizando OpenNi y Windows OS.

Figura 2.2 Muestra de la interacción con imágenes en Windows 7

Kinect + 3D TV = Realidad Virtual

Esta aplicación demuestra la combinación del poder que puede tener el Kinect de Microsoft con un 3D TV (Ver Figura 2.4). El resultado es un sistema responde a la presencia del usuario y crea la ilusión de realidad virtual, moviendo un avión en la pantalla.

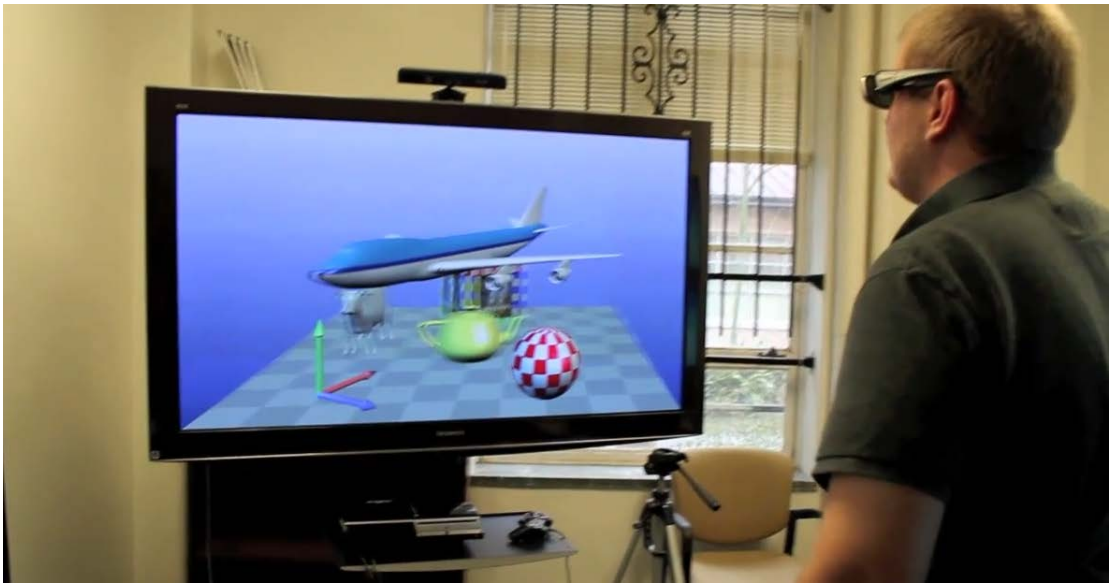


Figura 2.3 Vista del usuario utilizando la aplicación con las gafas estereoscópicas

La aplicación desarrollada por Robert Kooima [27] usa OpenNI con los controladores para Linux. La visualización usa Electro VR, y los gráficos son renderizados usando OpenGL y una tarjeta gráfica Nvidia GTX 470.

World of Warcraft con Microsoft Kinect usando FAAST y OpenNI

Desarrollaron un conjunto de librerías para facilitar el control de juegos y aplicaciones de realidad virtual denominada Flexible Action and Articulated Skeleton Toolkit (FAAST) [28].

Figura 2.4 El jugador utiliza el juego usando su cuerpo

Estas librerías o Kit permiten emular el teclado con una posición del cuerpo o gestos específicos y configurarlo para el juego.

FAAST es un software libre, usa OpenNI framework, desarrollado en la Universidad de Southern Carolina, no está disponible el código fuente.

Kinect + Google Earth!

Este proyecto fue realizado en el Instituto de tecnología de Stockolmo [29], consiste en usar el Kinect de Microsoft para navegar en Google Earth, el usuario puede usar su cuerpo para mover la vista del Google Earth (Ver Figura 2.5). Para este proyecto se utilizó SDK de Kinect y Microsoft Visual Studio Express C# 2010.

Figura 2.5 Muestra del usuario utilizando Google Earth

Para realizar el prototipo de API en nuestro trabajo se revisaron las mejores características de cada proyecto, para desarrollarlas y poder integrar estas funcionalidades.

2.2. Interfaces gráficas de usuario

Las interfaces gráficas de usuario se crearon para que la interacción con el usuario final sea entretenida, fácil de aprender, intuitiva y que sea transparente la programación sin necesidad de ver o escribir códigos.

Este lenguaje visual permite una interacción amigable con computadores (ver Figura 2.6). De esta manera se controla el número de acciones por vista y no se satura con demasiada información en una sola pantalla.

Figura 2.6 Concepto de Interface gráfica de usuario [3]

Las interfaces gráficas de usuario se han adaptado y evolucionado para un nuevo tipo de interacción de ambientes en 3D. Con este nuevo paradigma de interacción se adaptan de mejor manera a la interfaz natural de **usuario** (en inglés *natural user interface* o NUI) en las que se interactúa con un sistema, aplicación, etc. sin utilizar sistemas de mando o dispositivos de entrada (ratón, teclado, etc.) y en su lugar, se hace uso de movimientos gestuales de las manos o el cuerpo es el mismo mando de control, en el caso de pantallas capacitivas multitáctiles la operación o control es por medio de la yemas de los dedos en uno o varios contactos, también se está desarrollando control de sistemas operativos por medio de voz humana y control cercano a la pantalla pero sin tocarla.

En la figura 2.7 podemos apreciar lo que se entiende por interfaz natural de usuario:

Figura 2.7 Concepto de interfaz natural de usuario [19]

Las interfaces utilizadas en los proyectos utilizando el Kinect de Microsoft, los podemos ver a continuación:

En la Figura 2.8 observamos que las imágenes que se utilizan en la interface son muy grandes y ocupan casi toda la pantalla y el puntero (“la mano”) tiene un tamaño considerable para poder seleccionar.

Figura 2.8 Muestra una ventana para seleccionar imágenes [26]

En la Figura 2.9 para visualizar un auto en su interior, existe un menú inferior con botones grandes para fácil selección, ya que si fueran muy pequeños se dificultaría la interacción por la poca precisión.

Figura 2.9 Muestra del interior de un auto utilizando las manos [26]

Como podemos apreciar en los ejemplos anteriores, las interfaces gráficas de usuario han cambiado para adaptarse a las interfaces naturales de usuario, con iconos más grandes, respuesta instantánea a las acciones para una retroalimentación y mostrando la información necesaria.

2.3. Librerías

SDK Kinect para Windows beta

Esta fue la versión que se tomó en cuenta cuando se revisaron las librerías disponibles para el Kinect. Los drivers de este SDK fueron diseñados para el Kinect de Xbox 360 para aplicaciones no comerciales. En el cual esta versión tenía una estructura en la forma de programar, en cambio meses más tarde que se libero la versión comercial del SDK del Kinect, el modo de programar era totalmente distinto. Había que pasar todo ese código de la versión beta a la versión comercial (versión release) para que funcione. Como el SDK oficial de Microsoft salió tiempo después, que las librerías creadas por terceros, había mucha más documentación y más ejemplos.

Uno de los puntos negativos es que solo funciona para Windows 7 con Visual Studio 2010, tiene una base de funciones con las que se puede trabajar que es el tracking de reconocimiento del esqueleto de la persona, y no tiene soporte para Linux y MacOS.

Desde que estuvieron disponibles controladores (drivers) para el Kinect de la Xbox 360 para Microsoft Windows, luego para los demás sistemas operativos, la comunidad de desarrolladores a aumentado y han contribuido a la creación otros controladores más estables (tiene pocos bugs o errores en la programación), integrándolo con diferentes lenguajes de programación para que pueda ser usado en diferentes frameworks (usan otros lenguajes de

programación), quiere decir que hay una colección de funciones que pueden ser usadas con otros lenguajes de programación para permitir a un código funcione, junto con otro que no sea compatible (ejemplo Python, javascript, actionScript).

Las librerías disponibles que pueden ser utilizadas para proyectos utilizando el Kinect de Microsoft son:

Libfreenect

Esta librería fue desarrollada por la comunidad As3Kinect [1]. Esta librería incluye los drivers para que Kinect funcione en Windows, Mac OS y Linux.

Del repositorio https://github.com/OpenKinect/libfreenect se descargó el código fuente que se utilizo para este proyecto. Libfreenect es la librería principal para el acceso a la cámara USB del Microsoft Kinect y soporta el acceso a:

- Las imágenes RGB y de profundidad (de colores)
- Motor
- Acelerómetro
- Led indicador de encendido

CL NUI [20]

En Noviembre 6 del 2010, AlexP (su sobrenombre, no se encuentra información concreta de su nombre real) fue el primero en crear unos controladores para el Kinect de Microsoft de la Xbox 360 para usarlo en Windows 7 (ver Figura 2.10), después de esto una gran comunidad continuó la investigación y el desarrollo, para una plataforma estable, pero que no es código abierto, que tiene soporte, ejemplos útiles y documentación.

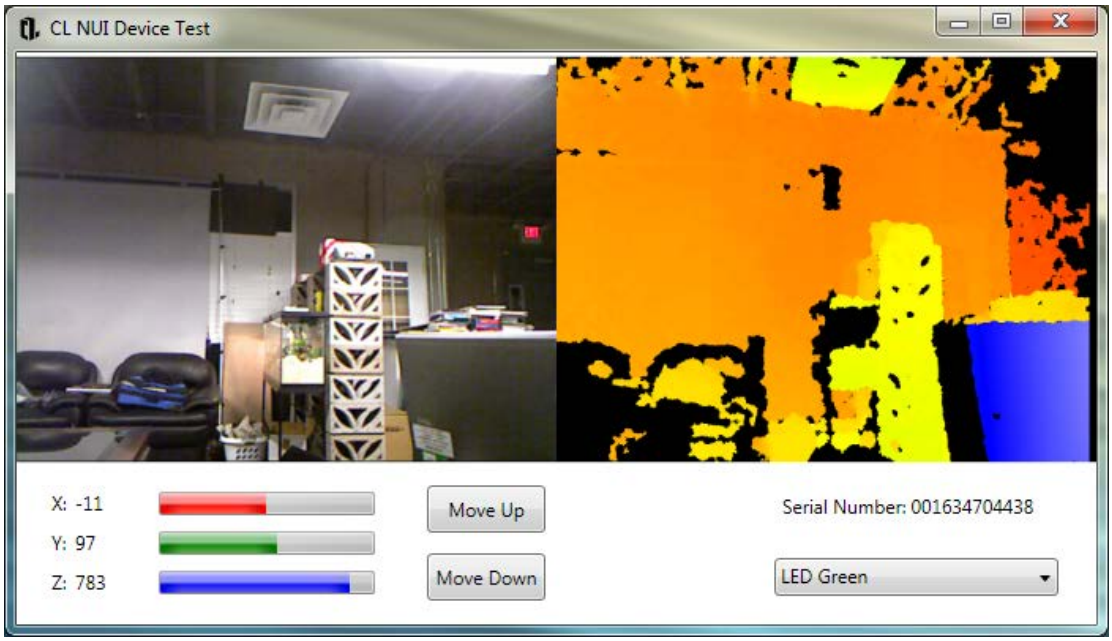


Figura 2.10 Test de los drivers de CL NUI [21]

OpenNI framework

OpenNI una organización sin fines de lucro para promover la compatibilidad de los dispositivos de interacción natural, aplicaciones y middleware [22]. Este framework permite comunicarse con los sensores de audio, video y sensor de profundidad de Kinect, mientras que proporciona una API que sirve de puente entre el hardware del equipo, las funciones de tracking del cuerpo y las aplicaciones e interfaces del sistema operativo. La idea es facilitar el desarrollo de aplicaciones que funcionen con interacción natural, llámese gestos y movimientos corporales (ver Figura 2.11).

Una de las ventajas que tiene OpenNI framework con CL NUI (es software licenciado) es que estos 2 diferentes frameworks utilizan el mismo controlador del kinect por eso son compatibles.

Se los tiene que instalar por separado: Cámara, audio y motor.

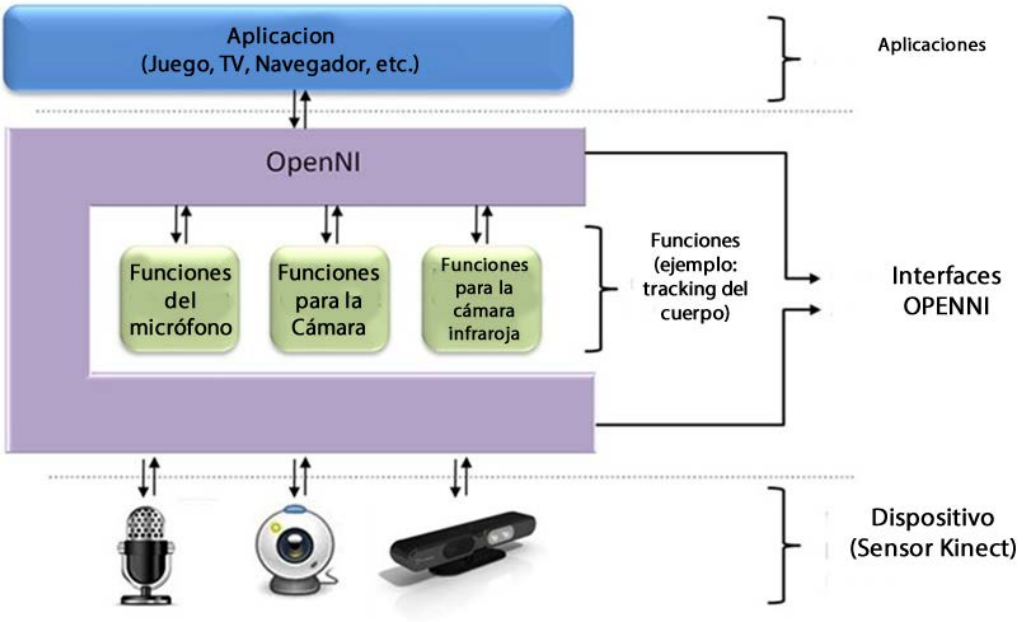


Figura 2.11 Una vista de las capas de OpenNI framework [24]

Frameworks disponibles considerados en el proyecto

Los frameworks disponibles para generar imágenes en 3D utilizados en los proyectos usando el Kinect:

- Openframeworks
- OpenSceneGraph
- QT
- OGRE 3D

OpenFrameworks

Es una herramienta de código abierto desarrollado en C++ diseñado para realizar procesos creativos de una manera simple intuitiva. Esta herramienta fue diseñada para trabajos de uso en general en las que se integran o utilizan librerías para gráficos, audio, tipografías, cargar y guardar imágenes, reproducción y procesamiento de video y otras cosas más.

El código está escrito para ser compatible con muchos lenguajes y sistemas operativos y soporta 4 ambientes de desarrollo integrado (“IDE”) (XCode, Code::Blocks, Visual Studio y Eclipse). Esta API es diseñada para ser fácil de comprender y escribir poco código.

Lo bueno de Openframeworks [6] es la gran variedad de librerías que se les puede añadir, puedes buscar las que están disponibles para poder utilizarlas y extenderlos.

OpenSceneGraph, es una herramienta muy potente que permite cargar los modelos 3D haciendo una simulación en tiempo real de los modelos utilizando las gafas estereoscópicas dinámicas. Tiene un motor gráfico de código abierto usado para el desarrollo de aplicaciones como simulación visual y video juegos. Esta escrita en C++ usando OpenGL y funciona en varios sistemas operativos Microsoft Windows, Mac OS X, Linux y Solaris. Es un poco inestable y tiene problemas con la integración por su incompatibilidad con las diferentes versiones de OpenCV.

QT es una biblioteca multiplataforma para desarrollar aplicaciones con interfaz gráfica de usuario en menos tiempo y crear diseños agradables. Se la quiso utilizar para crear botones con diseño en 3D con transparencias pero la integración con las demás librerías que se estaban utilizando daba problemas. Utilizando Visual Studio Express Edition 2010 no funcionaba correctamente, por lo que requería que se utilice la versión del Visual Studio 2010 Professional.

OGRE 3D es un motor de renderizado especializado en los gráficos 3D escrito en C++ , se pensó en Ogre debido a que puede cargar los objetos 3D con buena calidad, la desventaja es al integrar otras herramientas, problemas de incompatibilidad, referencias de código ambiguo con la librería kinect.

OpenCV

Es una librería de código abierto para visión por computadora [2] donde existen funciones para procesamiento digital de imágenes. Esta librería permite analizar las imágenes recibidas del Kinect en tiempo real, para poder

determinar la posición de las manos de acuerdo a los colores de la cámara infrarroja. Además de analizar los pixeles de la imagen para poder desarrollar los gestos de la aplicación. Esta librería tiene un gran rendimiento al estar implementado en C++ (originalmente fue implementado en C, a partir de las versiones 2.0 todo está en C++ nativo), lo que aporta a este proyecto es de vital importancia, porque con esta librería se analiza la imagen para la detección de las manos. Actualmente está en la versión 2.4, sin embargo se utilizo la versión 2.1 por problemas en la integración y compatibilidad en las funciones.

El factor principal es la integración por lo que se escogieron las siguientes librerías y framework para la implementación del proyecto son: Libfreenect, Openframeworks, OpenCV2.1 desarrolladas en C++ como lenguaje nativo lo que permitiría tener un buen desempeño en rendimiento e integradas en Visual Studio 2010 Express Edition para poder utilizar las funcionalidades nativas de Windows.

Se decidió por **Openframeworks,** principalmente porque existe una comunidad muy grande en constante desarrollo. Está orientado para aplicaciones interactivas, permite mostrar objetos 3D, ofrece muchas facilidades tales como al utilizarlo el código es reducido y fácil de comprender, la integración no es complicada, permite hacer modificaciones en el código así poder centrarse en la parte del proyecto.

De las 4 librerías disponibles que permiten el control del Kinect de Microsoft, se decidió por ***OpenKinect,*** por las siguientes consideraciones:

- El código utilizado es muy sencillo de entender y desarrollar
- Soporta Windows, Linux y Mac OS
- Soporta múltiples ambientes de desarrollo
- Posee módulos para OpenCV, C, python, actionscript, c#, java
- Posee soporte para RGB y profundidad de cámara, motor, acelerómetro y la luz LED de encendido
- Posee soporte para OpenGL

Para el análisis de las imágenes se decidió por OpenCV, porque tiene soporte, buen rendimiento, una comunidad en constante desarrollo, es estable, multiplataforma y de código abierto.

El proyecto OpenKinect [5] es una comunidad abierta que se interesa en hacer uso del Xbox Kinect de Microsoft con el computador, recordemos que fue creado para la consola de video juegos Xbox de Microsoft.

Una de las razones para tomarse el trabajo de escribir librerías externas es de no tener que comprar la Xbox 360 para poder usarlo y que no simplemente sea utilizado para juegos.

Visual Studio 2010 C++ Express Edition

Se utilizo este ambiente de desarrollo para integrar estas librerías utilizando el lenguaje C++ [7]. Las ventajas de utilizar el Visual Studio es utilizar nativamente las librerías para controlar el puntero del ratón y teclado numérico y alfanumérico en Microsoft Windows 7.

CAPÍTULO 3

3. ANÁLISIS Y DISEÑO DEL SISTEMA

3.1. Análisis y Diseño de la aplicación

Luego de revisar las herramientas existentes y todo lo desarrollado hasta el momento para el Kinect en el capitulo anterior, en esta sección se describen los requerimientos funcionales y no funcionales para el desarrollo de nuestra API.

Requerimientos funcionales

Los requerimientos funcionales son aquellos que establecen los servicios que la aplicación debe proporcionar, en especial lo relacionado a las entradas, salidas y excepciones.

Los requerimientos a desarrollar son:

a) Debe ser capaz de detectar con una precisión de 4cm en profundidad los movimientos lentos y suaves de las manos del usuario.

b) Debe permitir al usuario poder controlar los movimientos de rotación del objeto alrededor del eje Y; y el acercamiento y alejamiento de la cámara a través de los movimientos de las manos.
c) Los gestos implementados son mover mano a la derecha o izquierda, gesto de extender y recoger brazos.
d) Debe ser capaz de detectar los gestos de una persona, implementados en el API.
e) La persona que utiliza la aplicación debe estar ubicada entre 100 cm y 200 cm de distancia con respecto al Kinect, esto es por las limitaciones del dispositivo.
f) Debe aceptar la carga de modelos 3D con extensión .3ds (es un formato abierto).
g) Debe permitir la ejecución de comandos de Micrososft Windows usando las librerías nativas de Windows, como mover el puntero del ratón o la acción de presionar una tecla.

Requerimientos no funcionales

Los requerimientos no funcionales son las limitaciones en los servicios o funciones de una aplicación ofrece. Estas limitaciones incluyen aquellas que están relacionadas con el tiempo de respuesta, estándares que se deben cumplir, plataformas de hardware donde debe funcionar, etc.

Para este API, los requerimientos son:

a) El API desarrollado deberá como mínimo funcionar y hacer uso de los recursos de hardware de un computador con procesador de doble núcleo y con una tarjeta gráfica con GPU, para un mejor rendimiento y con un tiempo de respuesta para el análisis de al menos 30 imágenes por segundo.
b) El sistema operativo para que la aplicación funcione deberá ser Windows 7 32/64 bits.
c) No podrá funcionar en máquinas virtuales, porque no detecta el Kinect.
d) Requerirá que se conecte al computador un Kinect de Microsoft para Xbox 360.
e) Seleccionar el plan de energía de Windows 7, la opción de Alto rendimiento.

f) El área desde el Kinect hasta el usuario debe estar despejada, sin obstrucción de algún tipo.

Deberá hacer uso de las siguientes librerías: librfreenect, OpenCV y OpenFrameworks, por lo que deberán ser previamente ser instaladas.

En base a estos requerimientos funcionales y no funcionales y habiendo revisado previamente las librerías existentes que pudieran trabajar con el Kinect de Microsoft, se pudo establecer el diseño de la arquitectura de nuestro API. Esta arquitectura está orientada al control de la aplicación y al uso de comandos (como mover el puntero del ratón o presionar una tecla) de Microsoft Windows mediante la detección de gestos y movimientos de las manos tal como se indica en la Figura 3.1.

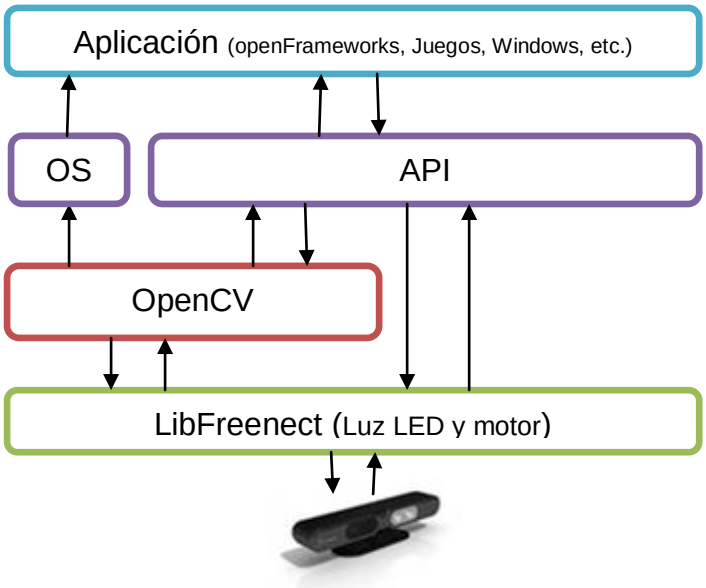


Figura 3.1 Arquitectura del API

La arquitectura de del API está diseñada para utilizar funciones del API, además de poder utilizar las demás librerías directamente y así implementar nuevas funcionalidades al API.

La librería LibFreenect permite conectarse al computador con el Kinect, teniendo acceso a las cámara infrarroja, que es controlada por funciones de OpenCV para procesar las imágenes y determinar los gestos.

El motor se lo controla con el API con un ángulo entre -31 a 31 grados y la luz LED para que se encienda o apague.

Librerías del sistema operativo nativo de Windows nos permite controlar el puntero del ratón y realizar acciones como la de presionar una tecla.

La aplicación utiliza las funciones del API y las del sistema operativo nativo para poder realizar los gestos, movimiento del puntero o la acción de presionar una tecla.

3.2. Alcance del diseño

Con los requerimientos funcionales y no funcionales identificados establecemos el alcance de nuestro diseño del API:

a) Seguir la arquitectura de la Figura 3.1.
b) Cargar objetos en formato .3ds en la aplicación.
c) Rotar el objeto en 3D, de izquierda a derecha o viceversa utilizando cualquier mano.
d) Acercamiento o alejamiento del objeto en 3D, separando las manos para acercar y juntándolas para alejar el objeto.
e) Controlar aplicaciones de Windows.
f) Utilizar las manos para controlar la posición del puntero.
g) Crear nuevos gestos para los comandos de Windows.
h) Generar el frustum, que es el área donde se va a detectar la posición de las manos, calculando la ubicación de la persona.

3.3. Diseño de pruebas

Para determinar el grado de precisión, sensibilidad y control que se puede tener con diferentes usuarios, se diseñaron un conjunto de pruebas de usabilidad de la aplicación.

Las pruebas consisten en determinar la experiencia del usuario a 3 diferentes distancias con respecto al Kinect (110 cm, 150 cm y 190 cm).

La altura a la que se trabajará con el Kinect es de 70 cm con respecto al piso.

Las pruebas constan de 3 partes:

1. Un cuestionario para determinar qué gestos realizaría el usuario para mover y realizar un acercamiento o alejamiento de un objeto 3D, pero sin ninguna previa instrucción.
2. Un conjunto de pruebas practicas para analizar la precisión, sensibilidad y control de los gestos diseñados y utilizados en la aplicación.
3. Un cuestionario al final de las pruebas practicas, para recoger datos sobre el modo de trabajar con la aplicación y poder determinar si los gestos utilizados fueron los adecuados, si los recordaría con facilidad y si el usuario se sintió bien (bajo tensión, satisfecho, molesto, impaciente, frustrado, bien) al terminar de usar la aplicación.

Las pruebas las realiza un encuestador, que anotará las observaciones y llenará el cuestionario para determinar que gesto realizaría (1ra. parte) y el cuestionario final (3ra. parte) de las pruebas prácticas. Se le entregará al usuario solo la hoja con las pruebas practicas (2da. Parte) para que las realice.

El cuestionario inicial consistió de tres preguntas para el usuario. Cada pregunta evaluaba específicamente qué gestos naturales se le ocurriría o tuviera en mente para realizar una acción determinada (ver Figura 3.2):

El encuestador le dice al usuario que realice el movimiento que respondió en la encuesta para anotar las observaciones.

Cuestionario para determinar los gestos que realizaría el usuario

a) ¿Cómo rotaría el objeto 3D que se encuentra en la pantalla?

	Derecha	Izquierda
Usa una mano	_____	_____
Usa ambas manos	_____	_____

Observación: _____

b) ¿Cómo haría más grande o acercar el objeto 3D?

	Derecha	Izquierda
Usa una mano	_____	_____
Usa ambas manos	_____	_____

Observación: _____

c) ¿Cómo alejaría o haría más pequeño el objeto 3D?

	Derecha	Izquierda
Usa una mano	_____	_____
Usa ambas manos	_____	_____

Observación: _____

Figura 3.2 Primera prueba de usabilidad

La prueba práctica (ver Figura 3.3) que realiza el usuario nos permite determinar las funcionalidades de la aplicación.

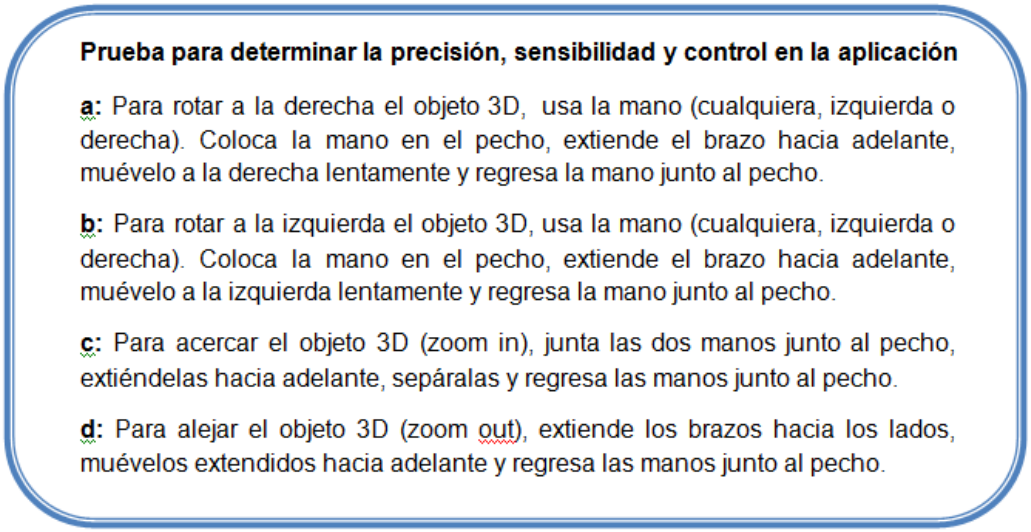


Figura 3.3 Prueba práctica del usuario

Para efectos prácticos utilizaremos la denominación de las letras a, b, c y d cuando se nombren las pruebas.

El cuestionario sobre las pruebas prácticas (ver Figura 3.4), tiene como objetivo la valoración cualitativa de la precisión, sensibilidad y control de la aplicación a 3 diferentes distancias 110 cm, 150 cm y 190 cm. Con estas pruebas de usabilidad se desea establecer cómo es la experiencia del usuario y si los gestos utilizados son los más adecuados.

a) **Precisión:** Si la aplicación responde realizando lo que realmente el usuario desea hacer.
b) **Sensibilidad:** Si se mueve muy despacio o rápido el manejo con las manos, si para mover el objeto es mejor mover mucho la mano o poco.
c) **Control:** Si los gestos utilizados son los adecuados, y comparados con los que describió en la primera encuesta, si fueron más adecuados o no.

Para esta evaluación se realizó una escala de respuestas de 1 al 5, donde:

- 1 es pésimo
- 2 malo
- 3 bueno
- 4 muy bueno
- 5 excelente

Se va a determinar la precisión, sensibilidad y control de cada acción realizada en el test práctico. Cada acción implementada por el API que puede realizar el usuario se la ha nombrado con una letra, para poder ser evaluada en el cuestionario (ver Figura 3.4) como se muestra a continuación:

a: Para rotar a la derecha el objeto 3D, usa la mano (cualquiera, izquierda o derecha). Coloca la mano en el pecho, extiende el brazo hacia adelante, muévelo a la derecha lentamente y regresa la mano junto al pecho.

b: Para rotar a la izquierda el objeto 3D, usa la mano (cualquiera, izquierda o derecha). Coloca la mano en el pecho, extiende el brazo hacia adelante, muévelo a la izquierda lentamente y regresa la mano junto al pecho.

c: Para acercar el objeto 3D (zoom in), junta las dos manos junto al pecho, extiéndelas hacia adelante, sepáralas y regresa las manos junto al pecho.

d: Para alejar el objeto 3D (zoom out), extiende los brazos hacia los lados, muévelos extendidos hacia adelante y regresa las manos junto al pecho.

Para la evaluación usando el cuestionario sobre las pruebas prácticas realizadas por el usuario (ver Figura 3.4), el encuestador coloca la letra a, b, c o d en los casilleros del 1 al 5, para la precisión, sensibilidad y control, para cada distancia correspondiente, con esto se evalúa independientemente cada acción del usuario.

A continuación se muestran las preguntas y el formato que se utilizó en este cuestionario:

Cuestionario sobre las pruebas prácticas realizadas por el usuario

Precisión	1	2	3	4	5
110 cm	_____	_____	_____	_____	_____
150 cm	_____	_____	_____	_____	_____
190 cm	_____	_____	_____	_____	_____

Sensibilidad	1	2	3	4	5
110 cm	_____	_____	_____	_____	_____
150 cm	_____	_____	_____	_____	_____
190 cm	_____	_____	_____	_____	_____

Control	1	2	3	4	5
110 cm	_____	_____	_____	_____	_____
150 cm	_____	_____	_____	_____	_____
190 cm	_____	_____	_____	_____	_____

	SI	NO
Recordaría los gestos utilizados	___	___

¿Cómo se siente al terminar de usar la aplicación?

Bajo tensión, molesto, satisfecho, impaciente, frustrado, bien

Figura 3.4 Cuestionario sobre las pruebas prácticas realizadas por el usuario

Las pruebas se las realizarán a 10 personas, entre niños, adultos, hombres y mujeres, que nos servirán para sacar una muestra de la funcionalidad de la aplicación. Se realizarán pruebas individualmente para saber el grado de precisión, sensibilidad y control de los gestos utilizados. Además determinar si los gestos fueron los correctos.

Los usuarios van a estar separados en grupos de menores de 18 años (niños y jóvenes) y mayores de 18 años (adultos).

Usuarios	Hombres	Mujeres
Niños/Jóvenes < 18 años	-	-
Adultos >= 18 años	-	-

Tabla 3.1 Edades de las personas a quien se realizarán las pruebas

CAPÍTULO 4

4. DESARROLLO Y ANÁLISIS DE RESULTADOS

4.1. Implementación

Para realizar las pruebas del API también se procedió a implementar una aplicación utilizando el Kinect y donde se llamaba a las funciones del API. Para el efecto se utilizaron: libfreenect, OpenFrameworks, OpenCV2.1, estas librerías están integradas en el Visual Studio 2010 C++.

La instalación detallada se encuentra en la sección de **ANEXOS** al final del documento.

Cada función del API llama o utiliza las librerías de: procesamiento digital de imágenes, control de las funcionalidades del Kinect de Microsoft, visualización de modelos 3D y control de acciones de Microsoft Windows por medio de código.

En nuestra implementación todas las funciones de la API comienzan con las letras “kl” como prefijo haciendo referencia a las iniciales en inglés de Kinect library.

Las funcionalidades de la API implementadas se las divide en 4 grupos, tienen una relación con el grupo de librerías opensource utilizadas.

Utilizando o llamando a funciones de la librería Libfreenect

1. **bool klGestoAcercar();**

Detecta los movimientos de las manos cuando se las une - Retorna Valor verdadero o falso si se detecta o no el gesto. Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

2. **bool klGestoAlejar();**

Detecta los movimientos de las manos cuando se las separa - Retorna Valor verdadero o falso si se detecta o no el gesto. Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

3. **bool klGestoClic();**

Detecta el movimiento de la mano cuando se la estira hacia adelante - Retorna Valor verdadero o falso si se detecta o no el gesto. Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

4. **void klKinectRango(int** rangoMin**, int** rangoMayor**);**

Función de inicialización del API para configurar manualmente el rango del frustum donde se quiere detectar las manos. El rango de los valores de entrada está entre 60 cm y 200 cm. Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

5. **void klKinect();**

Función de inicialización del API para configurar automáticamente la posición de la persona. Esta función siempre va antes de las demás funciones del API. Esta función se la coloca en el main de la aplicación.

6. **void klKinectDatos();**

Función que encera las variables globales que utilizan las funciones, siempre hay que colocar esta función al final de todos los procesos que realiza el API. Esta función se la coloca en el main al final de los procesos realizados por el API.

7. **struct manoPosicion klKinectXYZManoDerecha();**

Devuelve el valor de las coordenadas XYZ de la mano derecha - la estructura manoPosicion. Tiene 3 valores tipo **float** en la estructura. Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

8. **struct manoPosicion klKinectXYZManoIzquierda();**

Devuelve el valor de las coordenadas XYZ de la mano izquierda - la estructura manoPosicion. Tiene 3 valores tipo **float** en la estructura. Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

Utilizando o llamando a funciones de las librerías Libfreenect y Windows

1. **void klGestoPunteroMover();**

Función que permite mover el puntero del ratón con cualquiera de las dos manos (izquierda o derecha). Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

Utilizando o llamando a funciones de las librerías OpenFrameworks y Libfreenect

1. **void klModelo3DAcercar();**

Función que acerca la posición de la cámara del modelo 3D haciendo el efecto de zoo in en OpenFrameworks. Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

2. **void klModelo3DAlejar();**

Función que aleja la posición de la cámara del modelo 3D haciendo el efecto de zoom out en OpenFrameworks. Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

3. **void klModelo3DCargar(string nombreModelo, float scale = 1.0);**

Esta función carga un Modelo 3D con extensión .3ds. Los datos de entrada de esta función son uno de tipo **string** que es la ruta donde se encuentra el modelo3D en el computador y uno de tipo **float** que es el valor de la escala del modelo 3D, siendo 1 el tamaño original. Esta función se la coloca en el setup de la aplicación.

4. **void klModelo3DDibujar();**

Esta función dibuja el modelo en la pantalla, cargado con la función void klModelo3DCargar(**string** nombreModelo, **float** scale=1.0). Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

5. **void klModelo3DRotarEjeY();**

Función que hace rotar el modelo en el eje Y, al mover cualquier mano de izquierda a derecha o viceversa. Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

Utilizando o llamando a funciones de la librería de Windows

1. **void klClicIzquierdo();**

Función que permite hacer clic Izquierdo como si fuera el botón del ratón. Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

2. **void klClicDobleIzquierdo();**

Función que permite hacer doble clic Izquierdo como si fuera el botón del ratón. Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

3. **void klClicDerecho();**

Función que permite hacer clic Derecho como si fuera el botón del ratón. Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

4. **void klClicIzquierdoPresionado();**

Función que permite realizar la acción de mantener presionado el botón izquierdo del ratón. Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

5. **void klClicIzquierdoSoltar();**

Función que permite realizar la acción de soltar el botón izquierdo del ratón. Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

Dos funciones implementadas para utilizarlas en google maps para Acercar y Alejar el mapa, son un ejemplo de lo que se puede hacer con el API, utilizando las funciones para reconocer los gestos de abrir o juntar las manos y las funciones de Windows para la acción de presionar teclas, se juntan para tener las siguientes funciones:

6. **void klGoogleEarthAcercar();**

Función de ZOOM IN en la aplicación de Google Earth. Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

7. **void klGoogleEarthAlejar();**

Función de ZOOM OUT en la aplicación de Google Earth. Esta función se la coloca en el main de la aplicación, después de la inicialización del API.

Para la demostración de integración del API en un programa, se desarrolló una aplicación que permite visualizar y manipular un modelo 3D en pantalla con extensión .3ds (ver Figura 4.1).

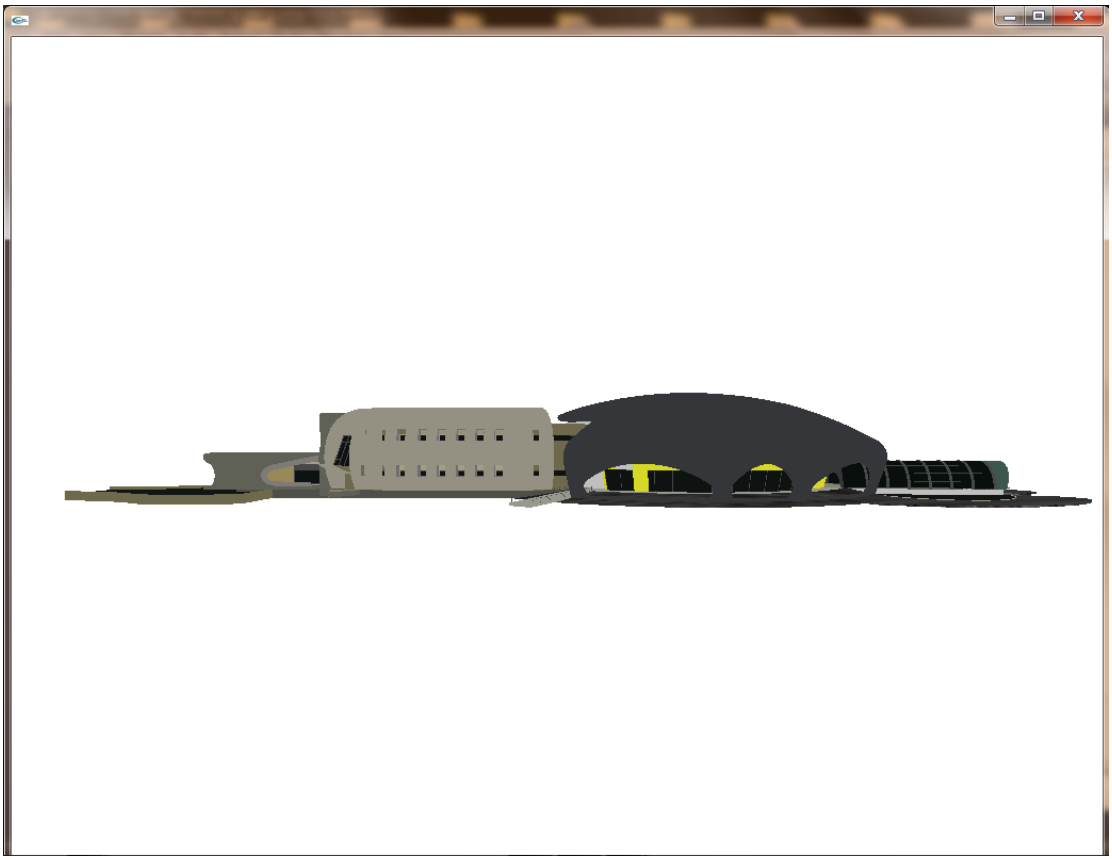


Figura 4.1 Pantalla inicial de la aplicación

Los círculos de color verde en la aplicación (ver Figura 4.2), aparecen cuando se detectan las manos y sirven para retroalimentación del usuario que la está utilizando.

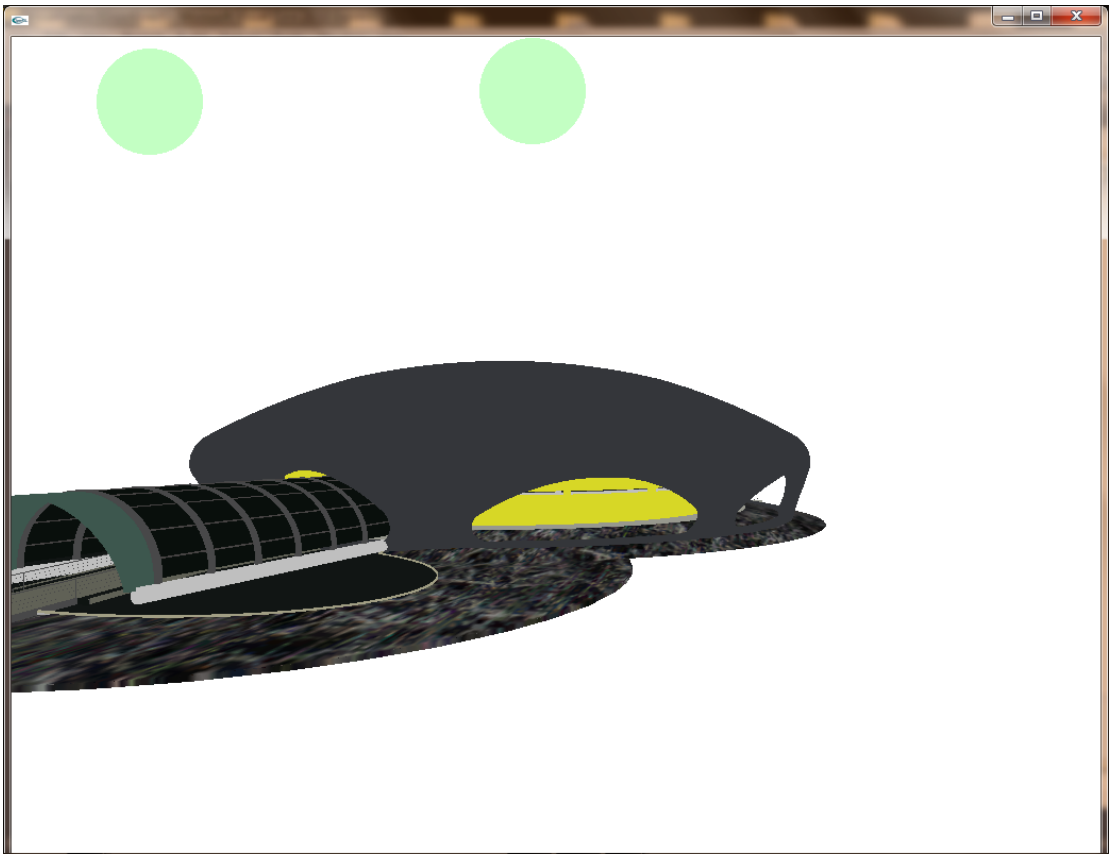


Figura 4.2 Vista de la pantalla cuando se detectan las 2 manos

Para la implementación de la aplicación se utilizaron las siguientes siete funciones:

1. **void klModelo3DCargar(string** nombreModelo**, float** scale = 1.0**);**
   Esta función se la ubica en la configuración (setup) de la aplicación.

   Las siguientes funciones se las ubica en el archivo principal de la aplicación (main), en el caso de OpenFrameworks es en la parte de dibujar de la aplicación (draw) (ver Figura 4.3):

```
#include "testApp.h"


//--------------------------------------------------------------
void testApp::setup(){
klModelo3DCargar("squirrel/cti.3ds", 5);
}

//--------------------------------------------------------------
void testApp::update(){
}

//--------------------------------------------------------------
void testApp::draw(){

//1.-CALIBRACION Y RECOLECCION DE DATOS

klKinect();

// 2.- DETECCIÓN DE GESTOS Y EJECUCIÓN DE ACCIONES EN LA APLICACIÓN
klModelo3DAcercar();
klModelo3DAlejar();
klModelo3DRotarEjeY();
klModelo3DDibujar();

//3.-ENCERAR VARIABLES GLOBALES QUE SE UTILIZAN EN EL API
klKinectDatos();

}
```

Figura 4.3 Código de la aplicación utilizando el API

2. **void klKinect();**

La función **klKinect** se la utiliza para inicializar el API, calibra automáticamente la posición de la persona y guarda la posición de las manos para que se ejecuten correctamente las demás funciones de nuestro API.

3. **void klModelo3DAcercar();**

La función **klModelo3DAcercar** (zoom in) modifica los valores de la cámara para visualizar más cerca el objeto 3D

4. **void klModelo3DAlejar();**

La función **klModelo3DAlejar** (zoom out) modifica los valores de la cámara para visualizar más lejos el objeto 3D

5. **void klModelo3DRotarEjeY();**

La función **klModelo3DRotarEjeY** modifica los valores de la cámara para visualizar en un diferente ángulo la posición del objeto con respecto al eje Y.

6. **void klModelo3DDibujar();**

La función **klModelo3DDIbujar** se la tiene que ubicar luego de las acciones que se realicen en la aplicación, esta función toma los valores que modifica al modelo 3D para dibujarlos en cada imagen nueva que se muestra en pantalla.

7. **void klKinectDatos();**

La función **klKinectDatos** va al final de todas las funciones que se ejecuten del API, nos permite encerar los valores de las variables utilizadas en la librería, no utilizarla haría que nuestra aplicación no funcione correctamente.

Para este ejemplo de la utilización del API se incluye la librería en el proyecto de OpenFrameworks de esta manera (ver Figura 4.4):

```
#pragma once

//Librerías del Kinect API
#include "klApi.h"


  //========================================================================

class testApp : public ofBaseApp{

    public:

        void setup();
        void update();
        void draw();
        void exit();

        void keyPressed(int key);
        void keyReleased(int key);
        void mouseMoved(int x, int y );
        void mouseDragged(int x, int y, int button);
        void mousePressed(int x, int y, int button);
        void mouseReleased(int x, int y, int button);
        void windowResized(int w, int h);
        void dragEvent(ofDragInfo dragInfo);
        void gotMessage(ofMessage msg);

};
```

Figura 4.4 Agregar librería klAPI a un proyecto de OpenFrameworks

Se agrega la librería en el archivo .h de la aplicación y el código de la librería (klAPI.cpp y klAPI.h) ubicarlos en la carpeta del proyecto ver Figura 4.5:

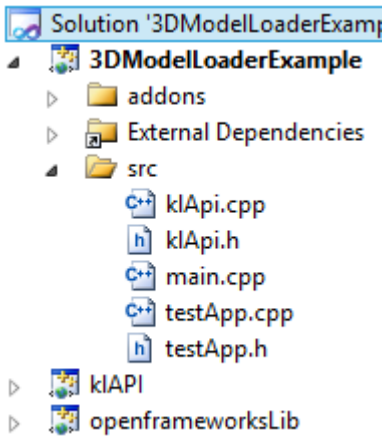


Figura 4.5 Librería del API ubicado en la carpeta del proyecto de OpenFrameworks

La metodología que se utiliza se la divide en 2 partes:

1.- Calibración y recolección de datos

2.- Detección de gestos y ejecución de acciones en la aplicación

Calibración y recolección de datos del kinect

En la calibración se detecta la posición de la persona donde se encuentra de pie y se determina el frustum que se va a utilizar para sensar las manos. La recolección de datos, guarda la posición de la mano izquierda y derecha, es decir, se determina la posición X, Y, y Z de las manos con respecto al kinect.

Detección de gestos y ejecución de acciones en la aplicación

En esta etapa se procesan los datos obtenidos del kinect para determinar el gesto y luego ejecutar acciones en la aplicación.

La implementación e investigación

Para determinar la profundidad de los objetos, se utilizó un rango de color en RGB que va desde 0 al 255 como se muestra en la Figura 4.6, siendo lo mas blanco lo más cerca y lo mas rojo lo más lejos. Se trabajó con la librería libfreenect que permite determinar la distancia en profundidad por medio de rangos de precisión, usando la función:

IplImage *GlViewColor(IplImage *depth) determina en este caso los rangos de precisión, hay 5 casos cada uno de ellos determina una distancia determinada, por ejemplo:

En este estudio la precisión es el error que hay para determinar la profundidad de los objetos del kinect en cada caso.

Caso 0: De 49 cm hasta 50 cm – precisión de 5 mm

Caso 1: De 50 cm hasta 60 cm – precisión de 1 cm

Caso 2: De 60 cm hasta 100 cm – precisión de 1 cm

Caso 3: De 100 cm hasta 500 cm

- De 100 cm a 150 cm – precisión de 2 cm
- De 150 cm a 190 cm – precisión de 4 cm
- De 190 cm a 240 cm – precisión de 5 cm
- De 240 cm a 330 cm – precisión de 9 cm
- De 330 cm a 430 cm – precisión de 18 cm
- De 430 cm a 500 cm – precisión de 20 cm

Caso 4: De 500 cm hasta 1000 cm – precisión es mayor a 20 cm y aumenta a mayor distancia.

Figura 4.6 Muestra del rango de color en Rojo en el Kinect

Cada caso tiene su rango de precisión, para la utilización de este proyecto se utilizó como precisión de profundidad de 4 cm, dejando lineal el rango de precisión entre 60 cm y 200 cm. En la Figura 4.7 se muestra el cambio de color de los casos, una vista en perspectiva desde un costado, además se aprecia el tono y el color que se utiliza para determinar la profundidad de los objetos.

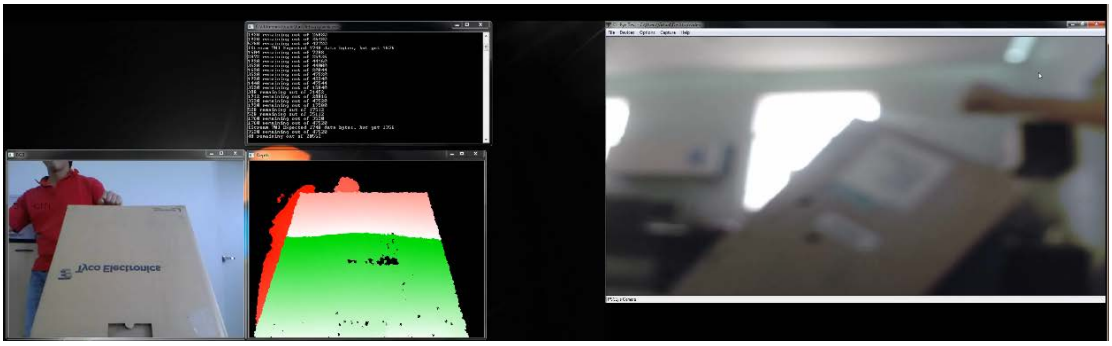


Figura 4.7 del rango de color de diferentes casos uno rojo y otro verde

En la figura 4.8 podemos observar que la diferencia de profundidad se puede determinar a simple vista por las diferentes tonalidades del color y los diferentes colores según el caso.

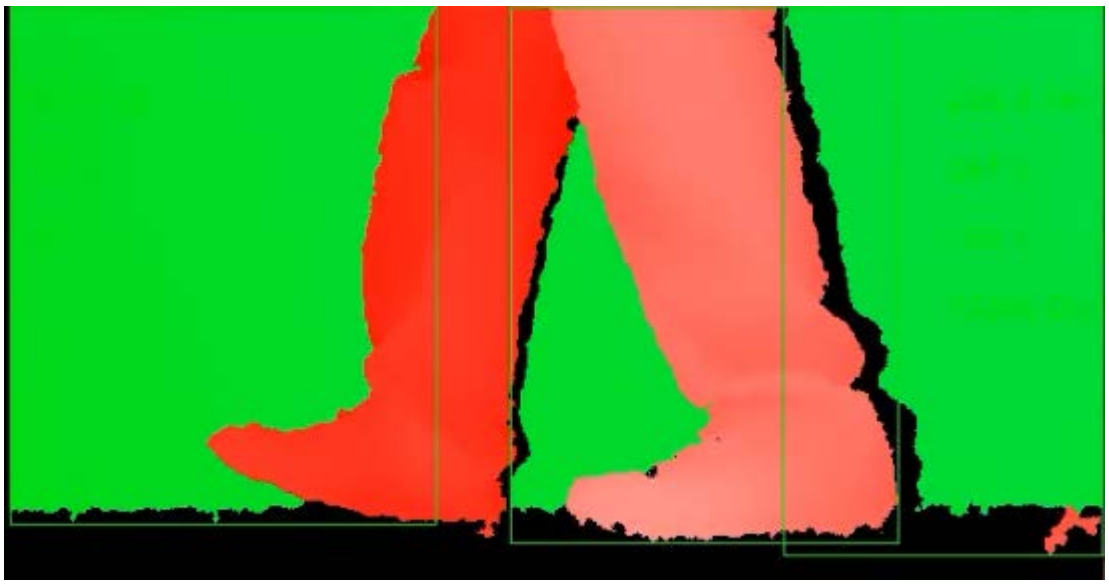


Figura 4.8 El pie izquierdo está más cerca, el pie derecho atrás y la pared

En la figura 4.9 podemos ver lo que ve una cámara infrarroja que es la nube de puntos.

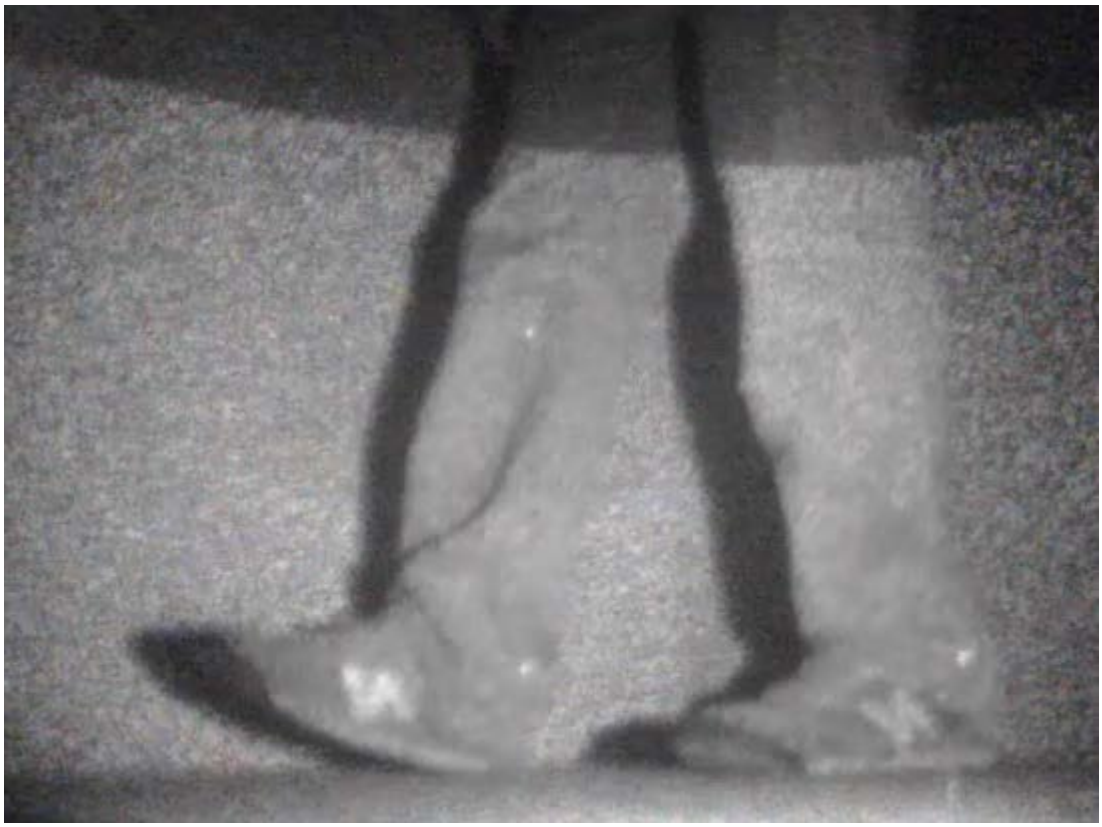


Figura 4.9 Vista desde una cámara infrarroja

La nube de puntos o la malla que proyecta el Kinect se la visualiza bien en la Figura 4.10 además hay unos puntos rojos en donde se encuentran unos puntos más intensos que son para utilizarlos como referencia.

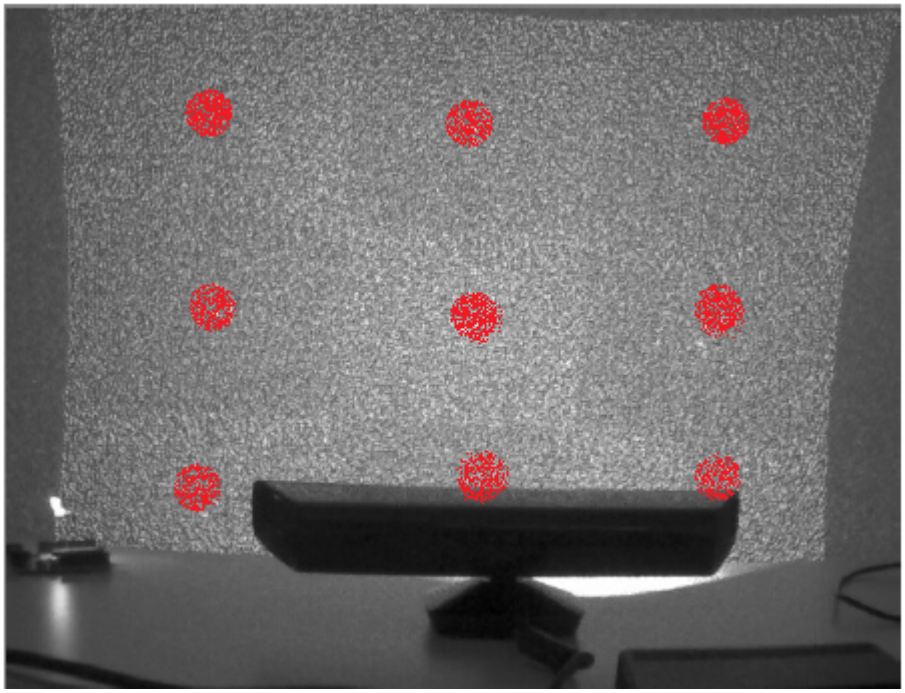


Figura 4.10 Malla de puntos que proyecta el Kinect vista con una cámara infrarroja

Los puntos visualizados desde más cerca usando una cámara infrarroja (ver Figura 4.11), para tener una referencia del diámetro de los puntos que se separan a mayor distancia se los visualiza a lado de unos dedos.

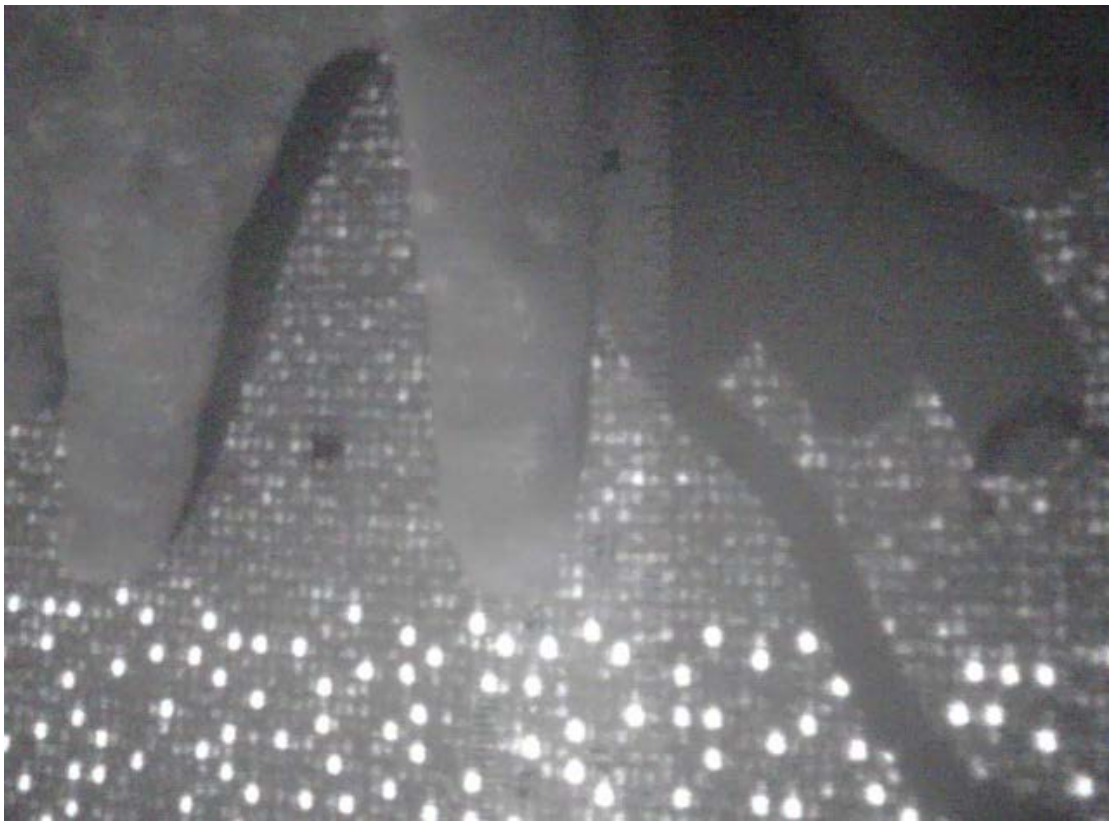


Figura 4.11 Vista de cerca de los puntos que proyecta el Kinect

Una demostración de los puntos que a menor distancia los puntos están más cerca, podemos ver en las figuras 4.12, 4.13, 4.14 y 4.15.

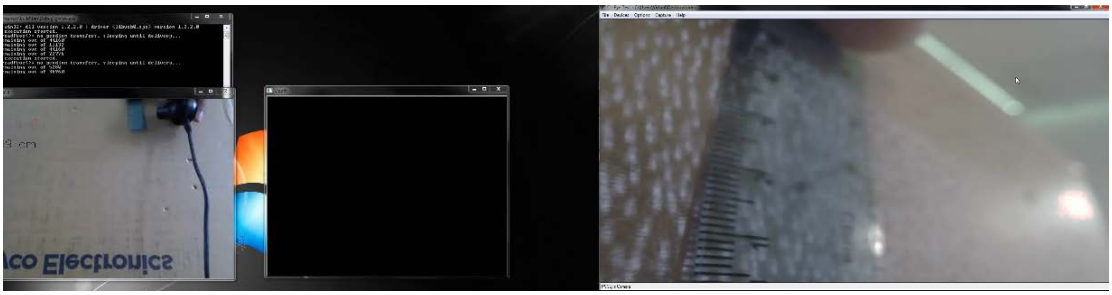


Figura 4.12 Vista de los puntos a una distancia de 45 cm

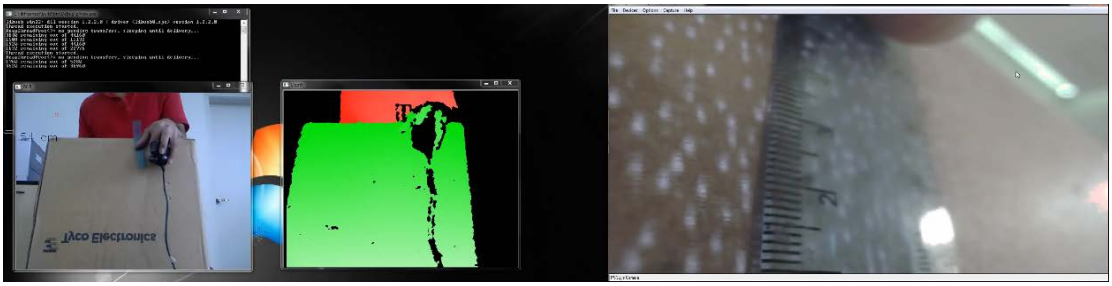


Figura 4.13 Vista de los puntos a una distancia mayor a 54 cm

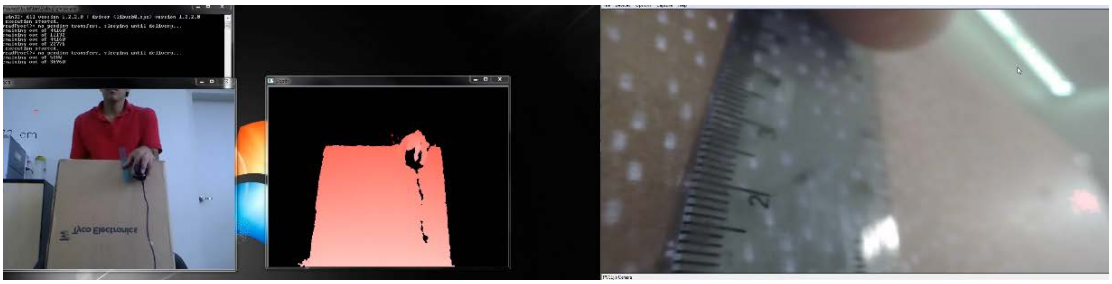


Figura 4.14 Vista de los puntos a una distancia de 90 cm

Figura 4.15 Vista de los puntos a una distancia de 120 cm

Como vemos en las imágenes los puntos se alejan entre sí a mayor distancia y están más juntos a menor distancia por eso hay una distancia mínima que es de 50 cm. Menor a esta distancia de 50 cm los puntos están muy pegados, están uno encima de otro y se cruzan no dejando determinar bien los puntos proyectados como lo vemos en la figura 4.16.

Figura 4.16 Vista de los puntos a una distancia menor a 40 cm

El haz de luz que proyecta la luz infrarroja desde el Kinect se la puede visualizar en la figura 4.17 y tener una idea de la misma (el Kinect está en el piso).

Figura 4.17 Haz de luz que proyecta el Kinect vista desde una cámara infrarroja

Para determinar la profundidad que determina el Kinect y la precisión se utilizo una regla de 2 metros de largo dividida en milímetros y centímetros, colocada en el piso midiendo el haz de luz, lo que marcaba en la regla y la

precisión que tenia al cambiar de posición de acuerdo a los casos. Con estas pruebas realizadas se creó una tabla para determinar la distancia que mide en profundidad pero para el proyecto se la estandarizó a 4 cm de precisión en el rango que se estableció, de 60 cm a 200 cm.

El procedimiento que se realiza es utilizar la cámara depth del Kinect de Microsoft para detectar el blob o la mancha que se reconoce dando el rango de posición de la persona establecido entre 100 cm y 200 cm respecto a la distancia del Kinect, con ésto se determina el blob más cercano y más grande de mayor área de pixeles, se asume que el objeto más grande es la persona, cuando ya está determinada la posición se determina el frustum que es el rango a partir de la posición de la persona. De acuerdo a las pruebas realizadas se determinó que sería de 50 cm la longitud.

Utilizando las librerías de OpenCV y las funciones para determinar los blobs o las manchas de la imagen, la imagen que procesamos es la de colores, ver figura 4.22, aquí vemos las manos que están siendo captadas por la cámara y esta imagen es la que se procesa.

Lo que se hace es determinar el número de blobs que tiene la imagen, para ésto hacemos nuestra imagen binaria, (que sea blanco y negro), utilizando las siguientes funciones (Figura 4.18):

```
1  im_gray = cvCreateImage(cvGetSize(frameDeLaimagen),IPL_DEPTH_8U,1);
2  cvCvtColor(frameDeLaimagen,im_gray,CV_RGB2GRAY);
3
4  im_bw = cvCreateImage(cvGetSize(im_gray),IPL_DEPTH_8U,1);
5  cvThreshold(im_gray, im_bw, 0, 255, CV_THRESH_BINARY | CV_THRESH_OTSU);
```

Figura 4.18 Funciones para transformar la imagen a binaria

Dilatamos un poco la imagen para tener mayor seguridad del blob (Figura 4.19):

```
1  cvDilate( im_bw, im_bw, NULL, 30);
```

Figura 4.19 Función para hacer más visible la mancha

Después de tener binarizada la imagen procedemos a contar el número de blobs que hay en la imagen (Figura 4.20):

```
1  contours = extract_and_filter_CC(im_bw, &mem, square);
```

Figura 4.20 Función para contar en número de manchas

Utilizamos un algoritmo que recorre los pixeles y determina los blancos y los vecinos, para cada contorno encontrado dibujamos una cajita encerrando el blob (Figura 4.21):

```
1  int square(CvSeq *contour)
2  {
3      CvRect box;
4      double R;
5      box = cvBoundingRect(contour, 0);
6      R = 1.0*box.width / box.height;
7      if (R > 0.6 && R < 1.4) {
8          return 1;
9      }
10     return 1;//se pone 1 para que coja cualquier tipo de rectángulo
11 }
```

Figura 4.21 Función para dibujar un cuadrilátero al encontrar las manchas

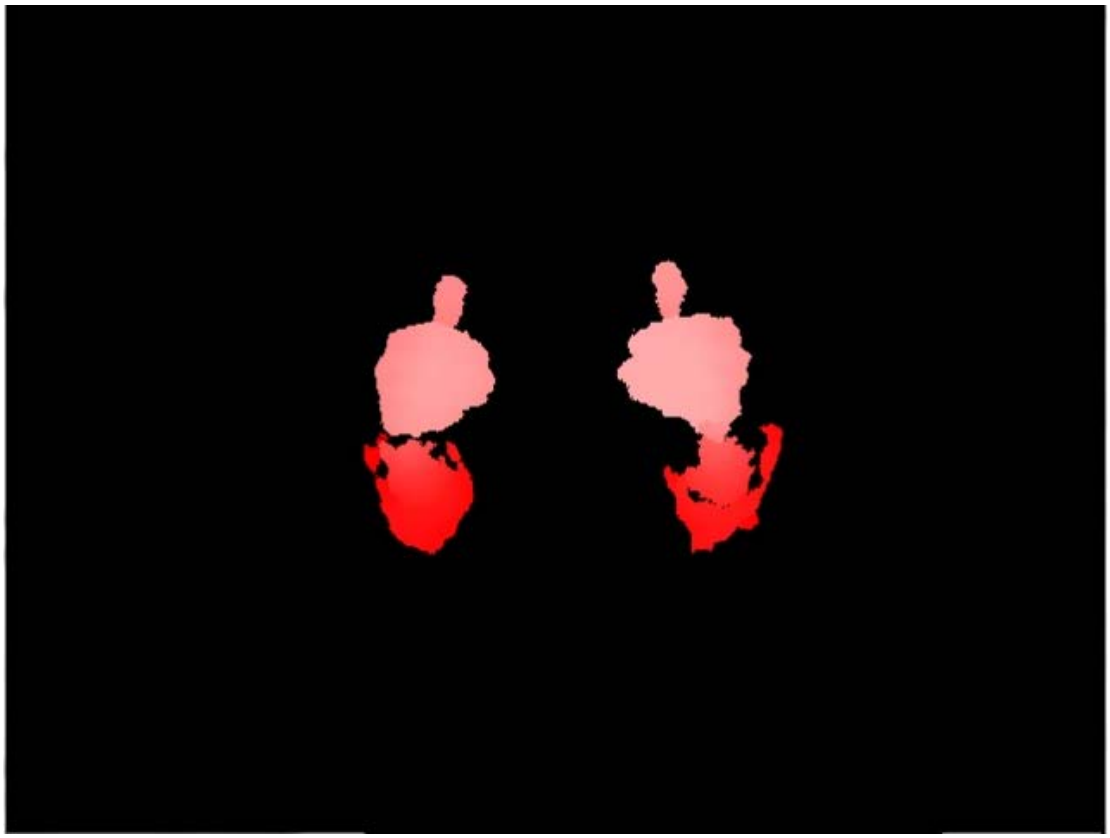


Figura 4.22 Imagen de colores que se va a procesar

Ya que determinamos el número de blobs recorremos este arreglo para saber cuál de estos blobs nos sirven (Figura 4.23):

```
1  if(boundingRect.width*boundingRect.height>1000)
2  //Si el área del cuadrado es mayor a 16 pixeles lo dibujo, sino no.
```

Figura 4.23 Verificamos que el área de la mancha sea mayor a 1000 pixeles

Con esto descartamos pequeñas manchas o blobs que puede haber en la imagen, luego sacamos una copia del recuadro o blobs seleccionados para poder analizarla y determinar el punto más cercano con esto podemos saber la distancia de la mano, como tenemos la imagen binaria, utilizamos la imagen original de colores tomando la posición de X y Y que la obtuvimos con el recorte y al dibujar el cuadrado sobre la imagen binaria (Figura 4.24):

```
//Recorro la imagen, cada pixel para saber el pixel mas cercano.
            for (imgy = 0; imgy < subimagen->height; imgy++) {

                for (imgx = 0; imgx < subimagen->width; imgx++) {

                    ptrsubimagen = cvPtr2D(subimagen,imgy,imgx,NULL);

                        bluez=ptrsubimagen[0];//blue
                        greenz=ptrsubimagen[1];//green
                        redz=ptrsubimagen[2];//red
                        if(bluez==0){ptrDistanciaZ=cvPtr2D(frame1,0,0,NULL); }
                        if(redz==0||redz==255){
                            if(redz==255&&greenz==0)if(bluez>pixcaso2){
                            pixcaso2=bluez; ptrfinal1Blue=ptrsubimagen[0];ptrfinal1Green=ptrsubimagen[1];ptrfinal1Red=ptrsubimagen[2];}

                            if(redz==0&&greenz==255&&bluez>0&&bluez<145)if(bluez<=pixcaso3){
                            pixcaso3=bluez;ptrfinal2Blue=ptrsubimagen[0];ptrfinal2Green=ptrsubimagen[1];ptrfinal2Red=ptrsubimagen[2];}


                        }


                }

            }
```

Figura 4.24 Recorremos la imagen para determinar pixel más cercano

Una vez determinada la posición más cercana de la mano debemos determinar si es la mano derecha o la mano izquierda, para ésto la primera vez asumimos que es la mano derecha si es que no hay más blobs encontrados, si hay mas blobs lo que hacemos es comparar la posición del pixel más cercano en X, cuál está mas a la derecha y cuál está mas a la izquierda.

De acuerdo al valor del color en RGB que obtengamos del pixel más cercano de la mano determinamos su posición, con la tabla de distancia determinada en el rango de 60 cm a 200 cm. Ver figura 4.25

```
if((redz==255)&&((255-lb)>=0) && ((255-lb)<=21)  ) { profundidadz=4+case2;}
if((redz==255)&&((255-lb)>21) && ((255-lb)<=47)  ) { profundidadz=8+case2;}
if((redz==255)&&((255-lb)>47) && ((255-lb)<=73)  ) { profundidadz=12+case2;}
if((redz==255)&&((255-lb)>73) && ((255-lb)<=99)  ) { profundidadz=16+case2;}
if((redz==255)&&((255-lb)>99) && ((255-lb)<=125)  ) { profundidadz=20+case2;}
if((redz==255)&&((255-lb)>125) && ((255-lb)<=151)  ) { profundidadz=24+case2;}
if((redz==255)&&((255-lb)>151) && ((255-lb)<=177)  ) { profundidadz=28+case2;}
if((redz==255)&&((255-lb)>177) && ((255-lb)<=203)  ) { profundidadz=32+case2;}
if((redz==255)&&((255-lb)>203) && ((255-lb)<=229)  ) { profundidadz=36+case2;}
if((redz==255)&&((255-lb)>229) && ((255-lb)<=255)  ) { profundidadz=40+case2;}

if((redz==0)&&((lb)>=0) && ((lb)<=8)  ) { profundidadz=4+case3;}
if((redz==0)&&((lb)>8) && ((lb)<=16)  ) { profundidadz=8+case3;}
if((redz==0)&&((lb)>16) && ((lb)<=24)  ) { profundidadz=12+case3;}
if((redz==0)&&((lb)>24) && ((lb)<=32)  ) { profundidadz=16+case3;}
if((redz==0)&&((lb)>32) && ((lb)<=40)  ) { profundidadz=20+case3;}
if((redz==0)&&((lb)>40) && ((lb)<=48)  ) { profundidadz=24+case3;}
if((redz==0)&&((lb)>48) && ((lb)<=56)  ) { profundidadz=28+case3;}
if((redz==0)&&((lb)>56) && ((lb)<=64)  ) { profundidadz=32+case3;}
if((redz==0)&&((lb)>64) && ((lb)<=72)  ) { profundidadz=36+case3;}
if((redz==0)&&((lb)>72) && ((lb)<=80)  ) { profundidadz=40+case3;}
if((redz==0)&&((lb)>80) && ((lb)<=88)  ) { profundidadz=44+case3;}
if((redz==0)&&((lb)>88) && ((lb)<=96)  ) { profundidadz=48+case3;}
if((redz==0)&&((lb)>96) && ((lb)<=104)  ) { profundidadz=52+case3;}

if((redz==0)&&((lb)>104) && ((lb)<=108)  ) { profundidadz=56+case3;}
if((redz==0)&&((lb)>108) && ((lb)<=112)  ) { profundidadz=60+case3;}
if((redz==0)&&((lb)>112) && ((lb)<=116)  ) { profundidadz=64+case3;}
if((redz==0)&&((lb)>116) && ((lb)<=120)  ) { profundidadz=68+case3;}
if((redz==0)&&((lb)>120) && ((lb)<=124)  ) { profundidadz=72+case3;}
if((redz==0)&&((lb)>124) && ((lb)<=128)  ) { profundidadz=76+case3;}
if((redz==0)&&((lb)>128) && ((lb)<=132)  ) { profundidadz=80+case3;}
if((redz==0)&&((lb)>132) && ((lb)<=136)  ) { profundidadz=84+case3;}
if((redz==0)&&((lb)>136) && ((lb)<=140)  ) { profundidadz=88+case3;}
if((redz==0)&&((lb)>140) && ((lb)<=144)  ) { profundidadz=92+case3;}

if((redz==0)&&((lb)>144) && ((lb)<=148)  ) { profundidadz=96+case3;}
if((redz==0)&&((lb)>148) && ((lb)<=152)  ) { profundidadz=100+case3;}
if((redz==0)&&((lb)>152) && ((lb)<=156)  ) { profundidadz=104+case3;}
if((redz==0)&&((lb)>156) && ((lb)<=160)  ) { profundidadz=108+case3;}
if((redz==0)&&((lb)>160) && ((lb)<=164)  ) { profundidadz=112+case3;}
if((redz==0)&&((lb)>164) && ((lb)<=168)  ) { profundidadz=116+case3;}
if((redz==0)&&((lb)>168) && ((lb)<=172)  ) { profundidadz=120+case3;}
if((redz==0)&&((lb)>172) && ((lb)<=176)  ) { profundidadz=124+case3;}
if((redz==0)&&((lb)>176) && ((lb)<=180)  ) { profundidadz=128+case3;}
```

Figura 4.25 Tabla de colores con la distancia determinada de acuerdo al caso

Con estos datos determinados de las manos sabemos la posición de X, Y y Z de la mano derecha y de la mano izquierda, utilizando diferentes algoritmos (Figura 4.26 y Figura 4.27) podemos realizar funciones para determinar qué gesto se está efectuando.

Con la posición de las dos manos en X y Y podemos determinar la distancia entre dos puntos, de esta manera podemos determinar si se están alejando

las manos o las estamos acercando cuando hacemos Zoom out o Zoom in respectivamente (Figura 4.26):

```
//Saco la distancia entre los dos puntos para hacer ZOOM
   distanciap=sqrt(((mxDer-mxIzq)*(mxDer-mxIzq))+((myDer-myIzq)*(myDer-myIzq)));
```

Figura 4.26 Función para determinar la distancia entre dos puntos

```
//Movimiento de acuerdo a los valores de x tomados del Kinect
//Funcion para mover en X
if(mxDer==0||mxIzq!=0){mxDerB2=mxDerB2;}
     else if((mxDer-1)>mxDerB){
     if((10>=mxDer-mxDerB)>=1)mxDerB2=mxDerB2+2;
     if(20>=(mxDer-mxDerB)>=11)mxDerB2=mxDerB2+5;}
     else if(0<mxDer&&(mxDer+1)<mxDerB){
     if(10>=(mxDerB-mxDer)>=1)mxDerB2=mxDerB2-2;
     if(20>=(mxDerB-mxDer)>=11)mxDerB2=mxDerB2-5;}
ofRotate(mxDerB2,0,1,0);
```

Figura 4.27 Función para determinar si el punto se mueve a la izquierda o derecha

Este es un ejemplo para una función de gestos, que puede ser utilizado en el computador. Todos los datos se guardan en variables globales, para luego ser usados en las funciones implementadas de los gestos, lo que se utiliza es la posición de las manos de los puntos más cercanos, con esto se pueden implementar y determinar un gesto.

Se utilizan funciones del Windows API [31] para interactuar con las aplicaciones del sistema operativo, utilizando funciones como *mouse_event function* (Windows) [32], para controlar el ratón.

Se utiliza la función GenerateKey para interactuar con Windows OS, el valor que se asigna está en hexadecimal (Figura 4.28).

```
GenerateKey(# hexadecimal,Valor verdadero o falso);
```

Figura 4.28 Función para simular acción de presionar una tecla

Las constantes que se utilizan para simular el teclado están en la sección de Keyboard Input Notifications [34] del Windows API.

4.2. Pruebas

Las pruebas se las realizaron a 10 personas (ver Tabla 4.2), entre niños, adultos, hombres y mujeres. Para recolectar los datos se utilizaron las pruebas de usabilidad descritas en el diseño de pruebas, en el capítulo 3.3.

Se realizaron las pruebas individualmente para saber el grado de precisión, sensibilidad y control de los gestos utilizados. Además determinar si los gestos fueron los correctos.

Usuarios	Hombres	Mujeres
Niños < 19 años	2	2
Adultos > 18 años	4	2

Tabla 4.1 Edades de las personas a quien se realizaron las pruebas

Las 4 pruebas se las realiza a cada persona, primero a 110 cm, luego a 150 cm y al final a 190 cm de distancia del kinect. Al término de las pruebas se llena la encuesta para saber que le pareció la experiencia de su uso.

Se realizó la encuesta a 10 personas entre niños, niñas, hombres y mujeres, para determinar las diferentes reacciones en el uso de la aplicación.

4.3. Resultados

Las pruebas de usabilidad realizadas a personas de diferentes edades y sexo nos dan los siguientes resultados que se pueden apreciar en las Figuras 4.20, 4.21, 4.22 y 4.23. Se muestran los gráficos para las 4 diferentes acciones: para rotar el objeto a la derecha, para rotar el objeto a la izquierda, para acercar el objeto (zoom in) y alejar el objeto (zoom out), como se los detalla a continuación:

A: Para rotar a la derecha el objeto 3D, usa la mano (cualquiera, izquierda o derecha). Coloca la mano en el pecho, extiende el brazo hacia adelante, muévelo a la derecha lentamente y regresa la mano junto al pecho.

B: Para rotar a la izquierda el objeto 3D, usa la mano (cualquiera, izquierda o derecha). Coloca la mano en el pecho, extiende el brazo hacia adelante, muévelo a la izquierda lentamente y regresa la mano junto al pecho.

C: Para acercar el objeto 3D (zoom in), junta las dos manos junto al pecho, extiéndelas hacia adelante, sepáralas y regresa las manos junto al pecho.

D: Para alejar el objeto 3D (zoom out), extiende los brazos hacia los lados, muévelos extendidos hacia adelante y regresa las manos junto al pecho.

Para la precisión (ver figura 4.29) se puede apreciar que las personas que utilizaron la aplicación les parecía la mejor experiencia a una distancia de 190 cm desde la posición del kinect para las 4 encuestas.

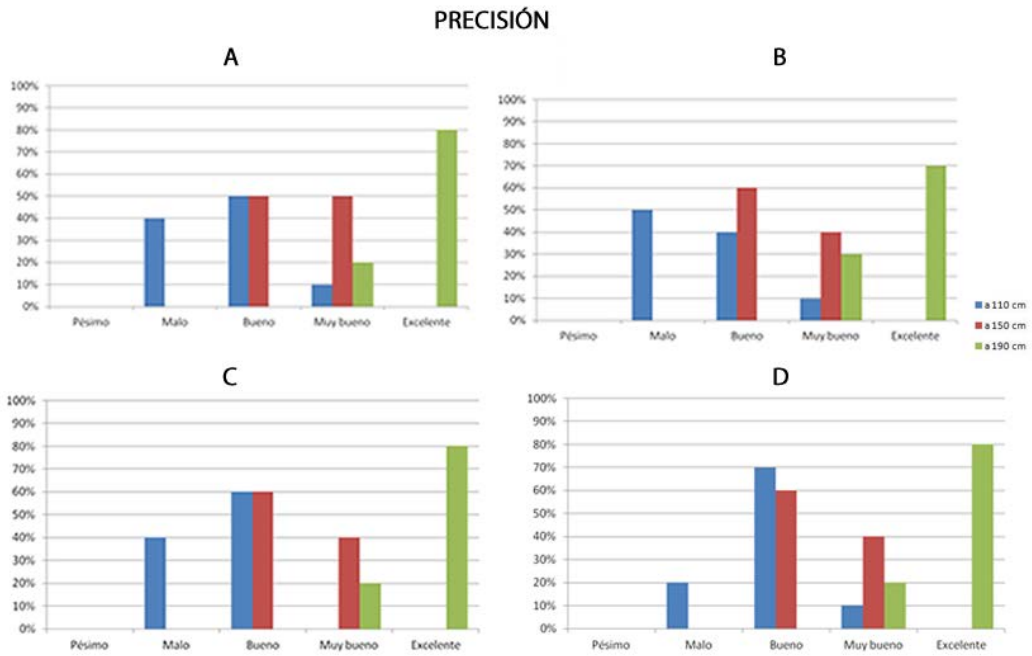


Figura 4.29 Datos obtenidos de los usuarios sobre la precisión de la aplicación

En los gráficos de la figura 4.30 observamos la sensibilidad de la aplicación, se puede apreciar que la sensibilidad a una distancia cerca de 110 cm es muy sensible y a las personas no les gustaba mucho eso, excepto cuando había que alejar el objeto les parecía bueno que se alejara rápidamente.

La sensibilidad a una distancia de 150 cm las personas en general les pareció buena para todas las acciones. En las pruebas para la distancia

mayor, rotar y acercar el objeto les parecía muy bueno, excelente, pero cuando tenían que alejar, les parecía mejor a una distancia más cerca porque se alejaba más rápido.

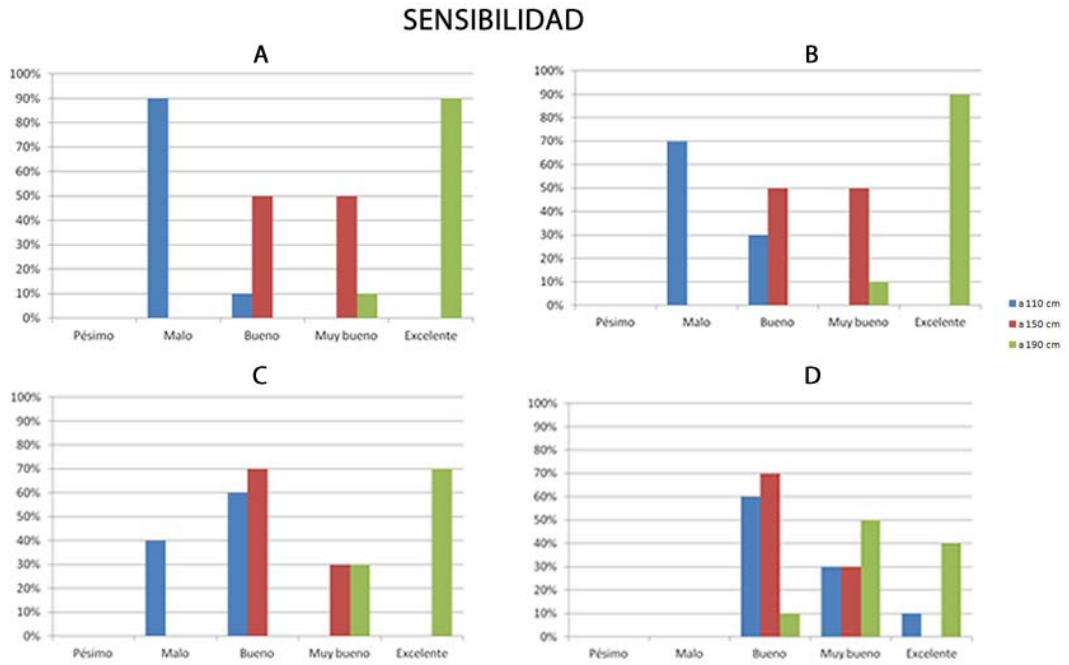


Figura 4.30 Datos de la sensibilidad de la aplicación

El control en todas las pruebas estuvo bien, donde hay mayor contundencia es a mayor distancia a 190 cm, donde el control de los gestos es excelente según los encuestados. El control a una menor distancia estaba bien pero a mayor distancia es un poco mas estable y agradable el manejo.

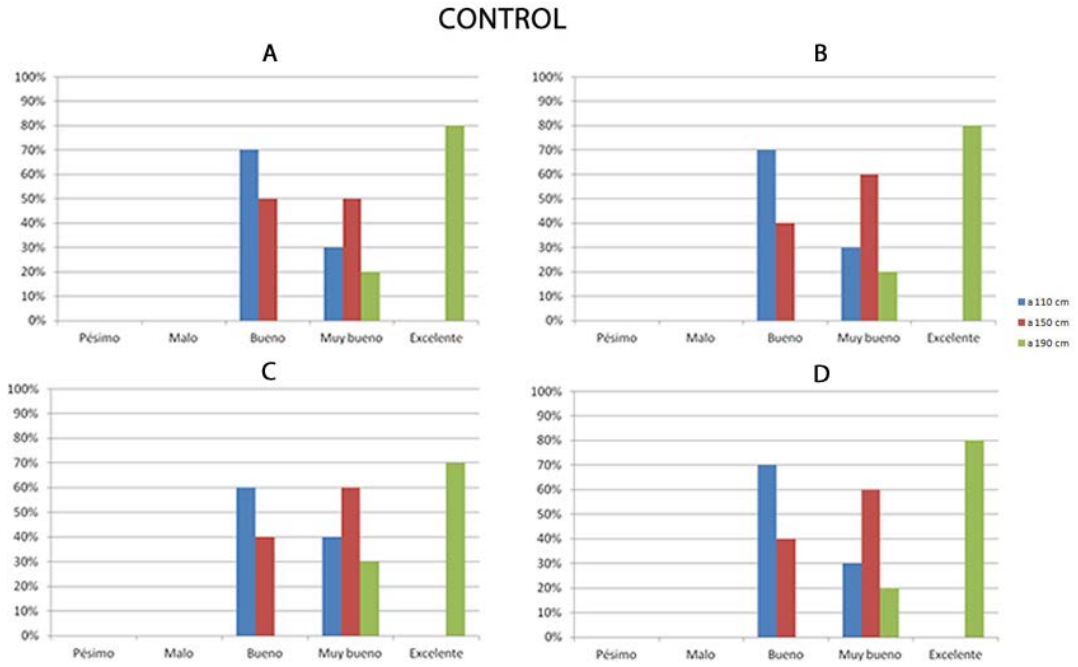


Figura 4.31 Datos de los encuestados sobre el control de la aplicación

El total de las personas que utilizaron la aplicación recordarían los gestos utilizados, porque están semejados a los utilizados en las tabletas que utilizan tecnología táctil y ya tienen un previo conocimiento de la utilización, de igual manera a persona que no han utilizado una tableta también se les hacia fácil recordar.

Con los datos obtenidos de los usuarios observamos que la mayor precisión, sensibilidad óptima y control en el uso de la aplicación está a la distancia de 190 cm, en las gráficas se observa que la mayor cantidad de personas se siente mejor al utilizar la aplicación a esta distancia, y esto se debe a que a mayor distancia la mano recorre menos pixeles y se puede rotar el objeto con mayor precisión y la sensibilidad es mejor, porque al estar muy cerca la sensibilidad es muy alta y al mover un poco la mano se recorren muchos pixeles y eso hace que rote mas el objeto 3D.

Los gestos utilizados en la aplicación serían recordados por todos los encuestados.

Al revisar los resultados nos dimos cuenta que el 90% de las personas se encuentras bien o satisfechas al usar la aplicación y un 10% de las personas frustradas.

CONCLUSIONES

1. Con los datos obtenidos de las pruebas de usabilidad, podemos sacar como conclusiones que la mejor precisión, sensibilidad y control de la aplicación se obtiene dentro de un rango de 180 a 190 cm con respecto al Kinect.

2. Los gestos implementados fueron los adecuados para la aplicación porque todos los que realizaron las pruebas estuvieron de acuerdo que recordarían fácilmente los movimientos realizados si volvieran a interactuar con la aplicación.

3. Este API es la base para que puedan integrarse nuevos algoritmos de reconocimiento de gestos para la interacción con el computador, la aplicación ejemplo fue un objeto en 3D que se lo controla con gestos en una PC con ambiente Microsoft Windows, pero que también se puede utilizar el API en Linux y Mac OS.

RECOMENDACIONES

1. Para futuros proyectos se pueden implementar nuevos algoritmos, que utilicen técnicas de reconocimiento de imágenes para nuevos gestos.

2. También se podría utilizar un método que incluya el reconocimiento en la imagen con la cámara RGB para determinar la posición de los objetos, mediante la comparación de la posición del pixel en la imagen, con la imagen de la cámara de profundidad (Depth camera).

3. El código implementado del API se lo puede utilizar en Linux y Mac OS, hacer las respectivas instalaciones de OpenCV, OpenFramworks y Libfreenect.

4. Hacer las implementaciones para controlar el puntero y presionar teclas utilizando el IDE nativo para la integración en Linux o Mac OS.

ANEXOS

ANEXO 1: DETALLE DE LA INSTALACIÓN E INTEGRACIÓN DE LAS LIBRERÍAS

La integración se realizó en Visual Studio 2010 C++ Express, usando como base un proyecto de Openframeworks (Versión: of_v0071_vs2010_release), el addon (el addon es un ejemplo) 3DModelLoaderExample que carga modelos 3D en formato .3DS, que se encuentra en el directorio:

C:\of_v0071_vs2010_release\examples\addons\3DModelLoaderExample, después de descomprimir el archivo **of_v0071_vs2010_release**, luego hay que compilar los ejemplos.

Para compilar los ejemplos se abre una consola de MS-DOS en modo Administrador y se ubican en el directorio:

C:\of_v0071_vs2010_release\scripts\vs2010, luego se ejecutan estos archivos:

1.- setupCommandLine.bat

2.- compileAllExamples.bat

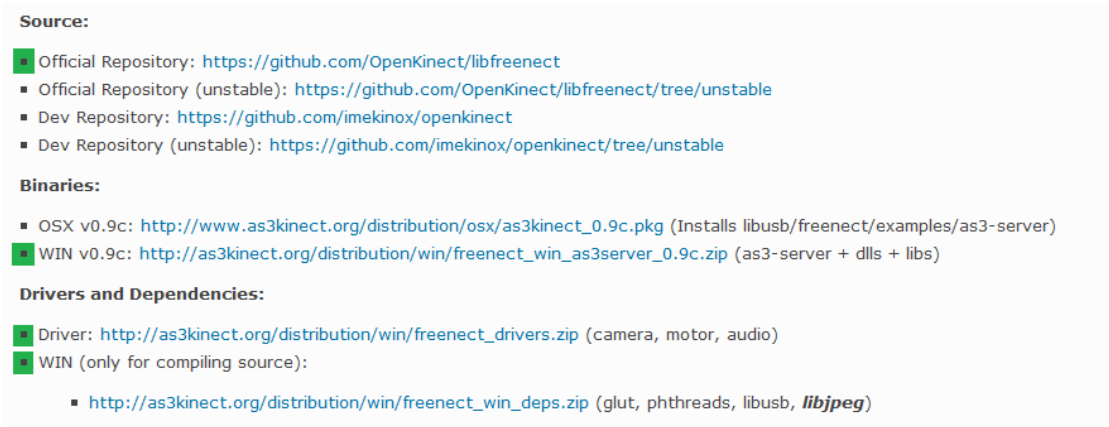
En este orden deben ser ejecutados, tardará unos minutos, dependiendo del procesador del computador

Ahora ya podemos abrir nuestro addon compilado que se encuentra en este directorio:

C:\of_v0071_vs2010_release\examples\addons\3DModelLoaderExample el archivo 3DModelLoaderExample.sln y lo ejecutamos, con esto ya podemos visualizar los modelos 3D.

Integración de libfreenect al proyecto de Openframeworks

Primero nos descargamos la librería Libfreenect del repositorio oficial: http://www.as3Kinect.org/download/ , el código fuente, los drivers y las dependencias (Ver ANEXO Figura 1):



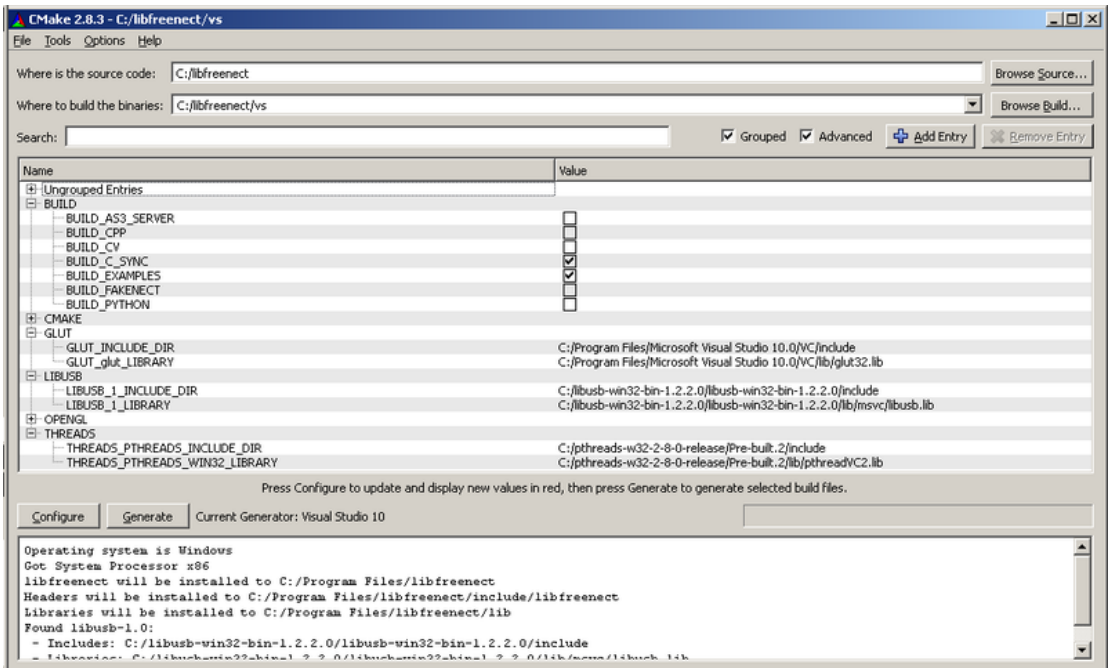
ANEXO Figura 1 Archivos para descargar

Procedemos la configuración de la librería para compilarla

Dependencias

Para cada dependencia requerida, primero descargamos e instalamos las librerías (.dll) en el sistema y luego identificamos los path a las librerías (.lib or .a) y header (.h) que se usaran para configurar en el Cmake:

Dependencia	Instalación	Libreria/includes usadas con Cmake (ANEXO Figura 2)
libusb-win32 – Descargar el ultimo libusb-win32-bin-x.x.x.x.zip **libusb-win32 version 1.2.5.0 o mas reciente.**	Tener cuidado en instalarlos en las carpetas correspondientes	El /lib e /include contienen las librerias y header requeridos para compilar el proyecto: • Para configurar el MS Visual Studio 2010, usar /lib/msvc/libusb.lib • Seleccionar el path de libusb header /include como libusb include path en cmake (LIBUSB_1_INCLUDE_DIR)
pthreads-win32 – descargar y extraer el ultimo pthreads-w32-x-x-x-release.exe	Buscar la carpeta /lib y copiar los .dll a la carpeta /windows o /windows/system32	El /lib e /include contienen las librerias y header que se requieren para compilar el proyecto: • Para configurar el MS Visual Studio 2010, usar /lib/pthreadVC2.lib como pthread library path en cmake (THREADS_PTHREADS_WIN32_LIBRARY) • Instalar el correspondiente .dll a /windows/system32 phtreadVC2.dll • Seleccionar el path pthread header /include como pthread include path en cmake (THREADS_PTHREADS_INCLUDE_DIR)
Glut – descargar y extraer el último glut-x.x.x-bin.zip	Buscar el glut32.dll y copiarlo en /windows/system32	• Con MSVC, copiamos glut.h a /include/GL y glut32.lib a /lib en (/Program Files/Microsoft Visual Studio 10.0/VC/) – Crear la carpeta GL y poner ahí el glut.h. En cmake-gui agregar el /include (NO ../include/GL) y el path (../lib/glut32.lib) en cmake (GLUT_INCLUDE_DIR and GLUT_LIBRARY respectively)



ANEXO Figura 2 Configuración de Cmake para compilar librería Libfreenect

Seguir estos pasos para configurar libfreenect y compilar:

1. Descargar Cmake (Cross-Platform Make) asegurar de tener instalado el compilador (Visual Studio 2010)
2. Iniciar Cmake-GUI y seleccionar /libfreenect como “source”, seleccionar la carpeta de salida, seleccionar "advanced" y "grouped" checkboxes para visualizar las demás variables y categorías entonces presionar en el boton "Configure"
3. Seleccionar el compilador C que en este caso el Visual Studio 2010
4. Seleccionar las operaciones que desee BUILD, tomando en cuenta las siguientes consideraciones:

    a) Por el momento solo seleccionar EXAMPLES y C_SYNC en BUILD.

5. Las dependencias no resueltas se mostraran en rojo en el CMake GUI. Proporciona los path que hacen falta y presionas de nuevo el botón de "Configure":

a) *_LIBRARY es una variable que necesita un archivo .lib no una carpeta
b) INCLUDE es una carpeta que tiene que estar referenciada al respectivo include

6. Cuando todos los errores han sido resueltos, hacemos clic en el boton "Generate" para crear los makefiles para el Visual Studio 2010.

Compilando en Visual Studio 2010

Ahora el proyecto está configurado, abrir libfreenect.sln y compilamos con el Visual Studio. Luego de esto revisamos los archivos compilados en las carpetas /bin y /lib

- Las librerías freenect están en /lib están compiladas. Las tenemos para usarlas e integrarlas en el proyecto de Openframeworks:
  - Luego debemos copia los .dll que se encuentran en la carpeta /bin donde se encuentra el proyecto de Openframeworks el compilado del proyecto.
- Si hay algunos items que no se compilaron bien , simplemente clic derecho y volvemos a compilar presionando "Build" or "Rebuild"
- Si hay problemas al compilar revisa la carpeta /libfreenect/platform/windows para más información.

Instalación del driver del Kinect para la Xbox 360 de Microsoft

Hay dos partes en la libreria libfreenect – el de bajo nivel libusb-based que es el controlador del dispositivo y el código libfreenect, la librería que se comunica con el controlador. Hay que instalar el controlador una sola vez.

Windows 7: paso a paso

- Conectar el Kinect. Windows muestra un mensaje que el dispositivo conectado no tiene el controlador correcto (el LED del Kinect no se muestra encendido). Si Windows muestra una pantalla para buscar un controlador, seleccionar cancelar.

- Abrir el administrador de tareas: **Inicio** >> **Panel de control** >> **Seguridad y Sistemas** >> **Sistema** >> **Administrador de dispositivo**

- Hay un dispositivo "*Xbox NUI Motor*" debería haber uno con el icono de precaución (probablemente en "**Otros dispositivos**") "!". Clic derecho sobre él, y seleccionar "**Actualizar el controlador...**", luego clic en "**Buscar software del controlador en el equipo**".

- "**Buscar**" y seleccionar la carpeta donde esta "*XBox_NUI_Motor.inf*" localizada en (/platform/windows/inf dentro de la carpeta de libfreenect). Clic "**Siguiente**" y si Windows muestra un non-certified driver , clic en aceptar, para que lo instale.

- Después de esto, el LED del Kinect debería encenderse y apagarse constantemente. Ahora veremos dos dispositivos mas en el administrador de dispositivos: "*Xbox NUI Camera*" and "*Xbox NUI Audio*". Repetir las instrucciones para estos dos dispositivos.

El Kinect ahora tiene los controladores para poder ser usados con la librería libfreenect.

Prueba

Para hacer una prueba rápida ejecutamos el programa /bin/glview.exe, que se encuentra en la carpeta del archivo descargado:
freenect_win_as3server_0.9c.zip

Con esta configuración tenemos lista la librería libfreenect para poder utilizar el Kinect en Windows.

OpenCV

Descargamos las librerías de OpenCV 2.1, para poder incluirlas en nuestro proyecto, descargamos las versión OpenCV-2.1.0-win32-vs2008, esta versión es la que utilizamos en este proyecto.

Integración de las librerías en Visual Studio 2010

Para realizar la integración utilizamos como proyecto base 3DModelLoaderExample que es un addon de Openframeworks para cargar modelos 3D con extensión .3DS

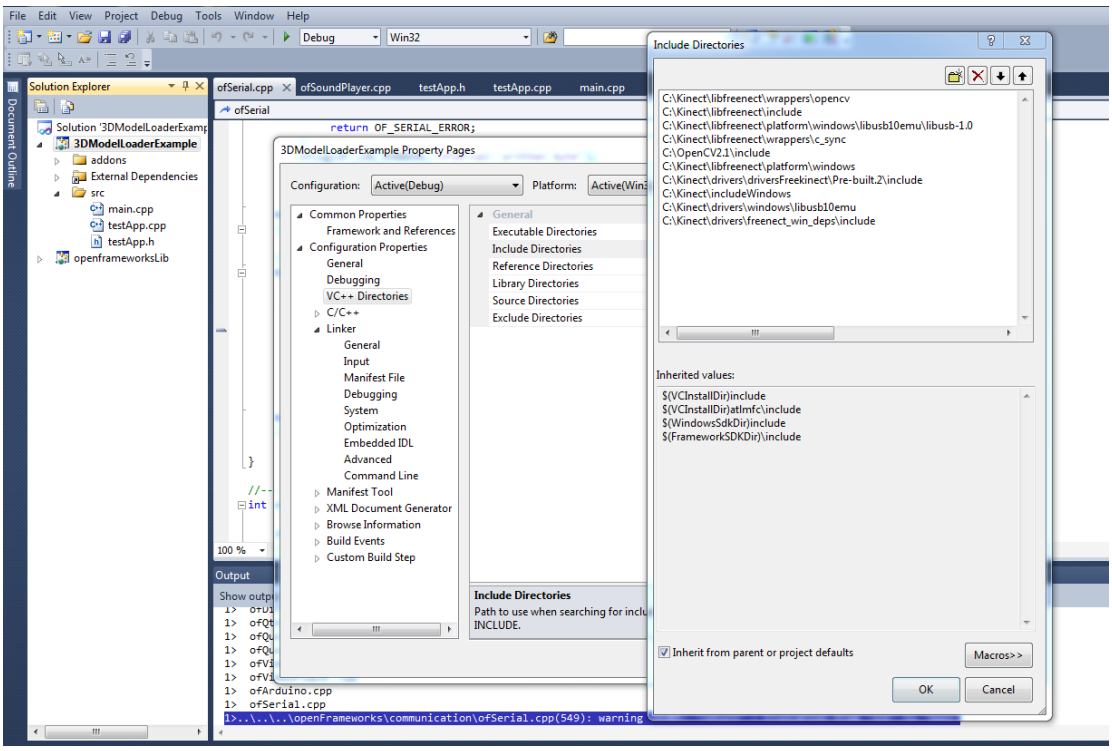
Abrimos el proyecto 3DModelLoaderExample.sln que se encuentra en el directorio:

C:\of_v0071_vs2010_release\examples\addons\3DModelLoaderExample

(Openframeworks se lo instaló en la carpeta raíz del sistema).

Ahora agregamos las librerías en el proyecto de Visual Studio 2010:

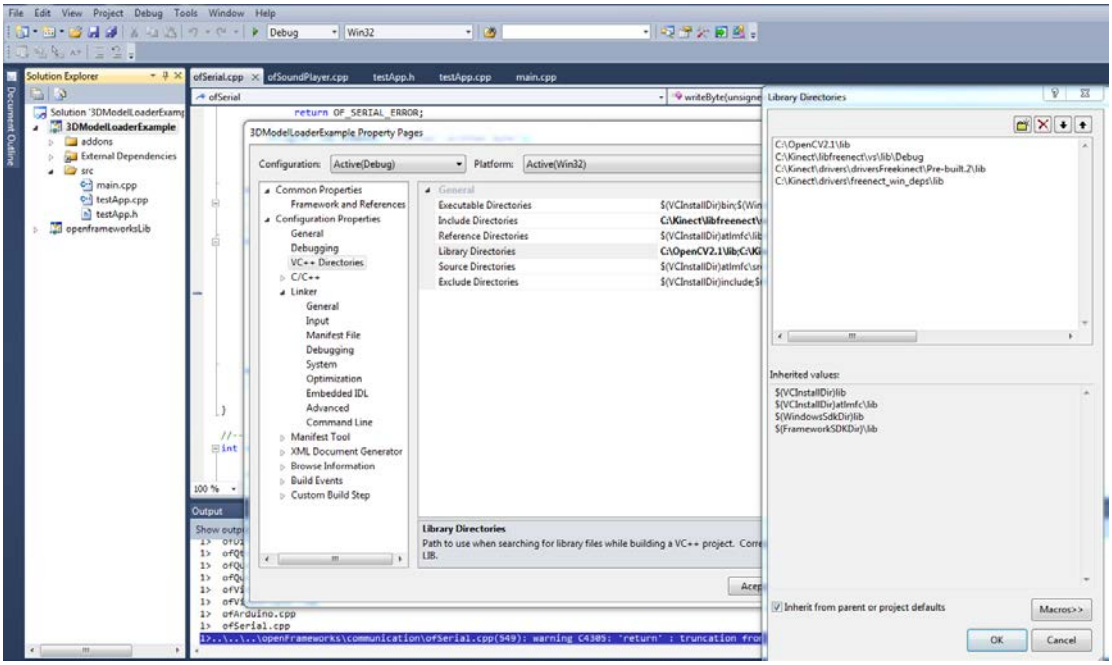
1. Hacemos clic derecho sobre el proyecto 3DModelLoaderExample, luego seleccionamos Referencias >> Propiedades de Configuración >> Directorios de CV++ >> Directorios de archivos de inclusión (Ver ANEXO Figura 3), está en modo Debug



ANEXO Figura 3 Ruta donde se agregan los .h de las librerías en el Visual Studio

Se está agregando las rutas de las librerías de libfreenct y las de OpenCV en el proyecto luego de esto hacemos clic en el botón Ok y luego Aplicar.

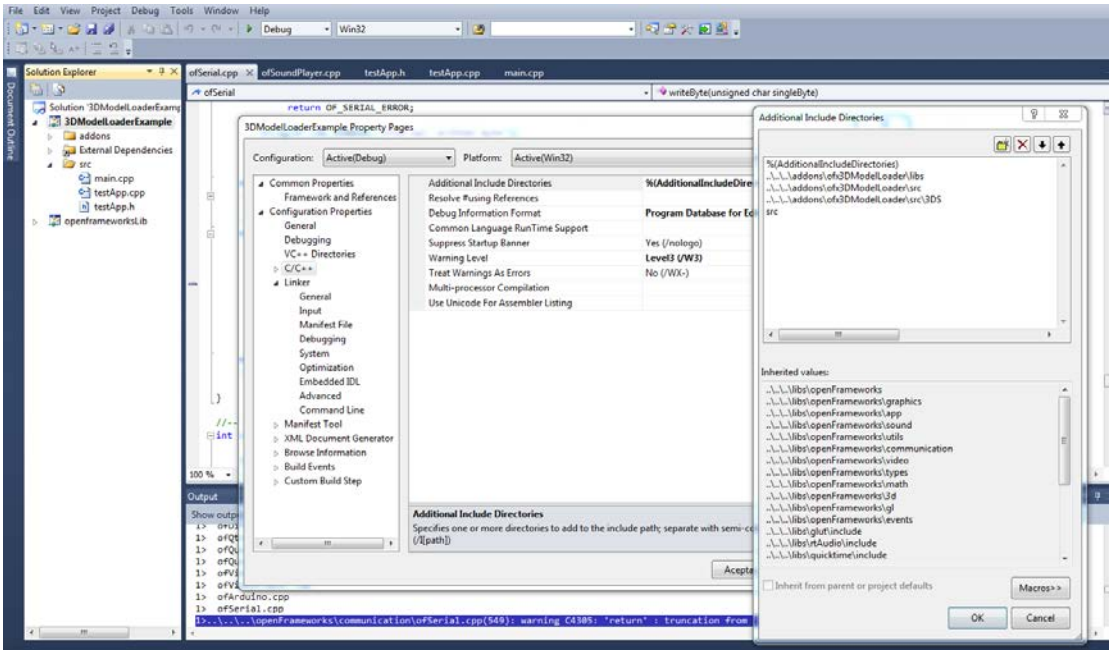
2. Hacemos el mismo procedimiento, clic derecho sobre el proyecto 3DModelLoaderExample, luego seleccionamos Referencias >> Propiedades de Configuración >> Directorios de CV++ >> Directorios de archivos de bibliotecas (Ver ANEXO Figura 4)



ANEXO Figura 4 Ruta donde se encuentran los .lib de las librerías en el proyecto

Se está agregando las rutas donde se encuentran los .lib de las librerías libfreenct y OpenCV , hacemos clic en Ok y luego en Aplicar.

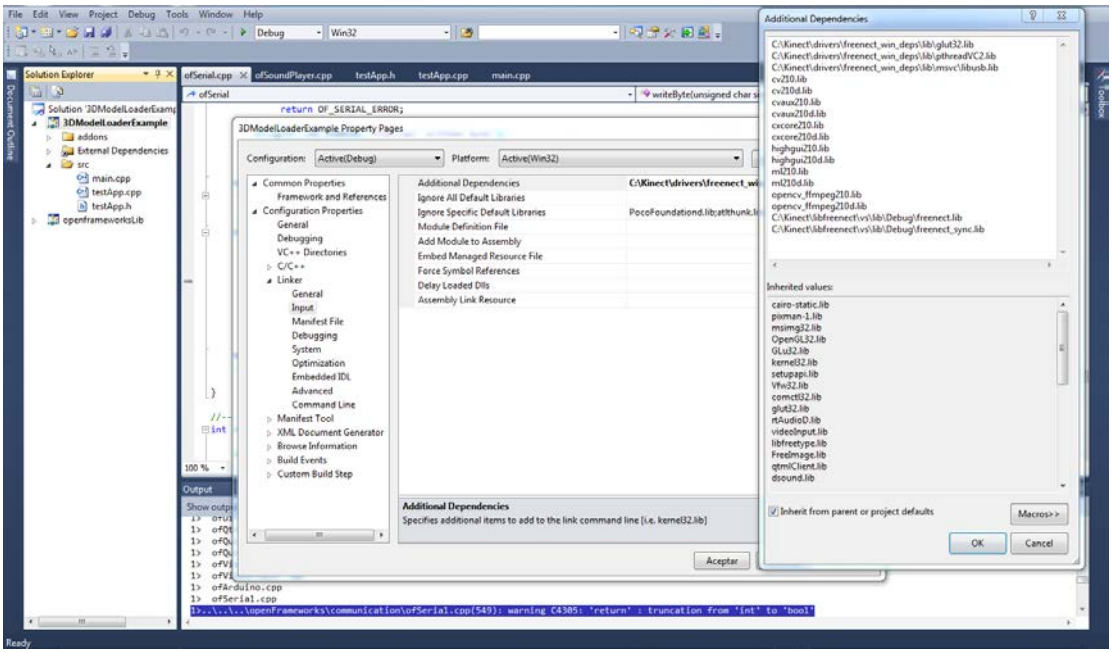
3. Revisamos que en la ruta de directorios adicionales estén correctamente , realizamos el mismo procedimiento clic derecho sobre el proyecto 3DModelLoaderExample, luego seleccionamos Referencias >> Propiedades de Configuración >> C/C++ >> Directorios de inclusión adicionales (Ver ANEXO Figura 5)



ANEXO Figura 5 Ruta donde se encuentran los directorios adicionales al proyecto

Si está todo bien hacemos clic en Ok, sino agregamos los directorios que se muestran en ANEXO Figura 5, luego hacemos clic en Ok y Aplicar.

4. Hacemos clic derecho sobre el proyecto 3DModelLoaderExample, luego seleccionamos Referencias >> Propiedades de Configuración >> Vinculador >> Entrada >> Dependencias adicionales (Ver ANEXO Figura 6)



ANEXO Figura 6 Ruta donde agregamos los nombres de las librerías de libfreenect y OpenCV

Lista de las librerías que hay que agregar:

C:\Kinect\drivers\freenect_win_deps\lib\glut32.lib

C:\Kinect\drivers\freenect_win_deps\lib\pthreadVC2.lib

C:\Kinect\drivers\freenect_win_deps\lib\msvc\libusb.lib

cv210.lib

cv210d.lib

cvaux210.lib

cvaux210d.lib

cxcore210.lib

cxcore210d.lib

cxcore210d.lib

highgui210.lib

highgui210d.lib

ml210.lib

ml210d.lib

opencv_ffmpeg210.lib

opencv_ffmpeg210d.lib

C:\Kinect\libfreenect\vs\lib\Debug\freenect.lib

C:\Kinect\libfreenect\vs\lib\Debug\freenect_sync.lib

Después de agregar estas librerías le hacemos clic en Ok y luego Aplicar.

Hay que tomar en cuenta de que hay un archivo .h que se encuentra en la carpeta C:\Kinect\includeWindows **winable.h** es un archivo de cabecera que fue modificado y para que funcione el proyecto debe ser este específicamente.

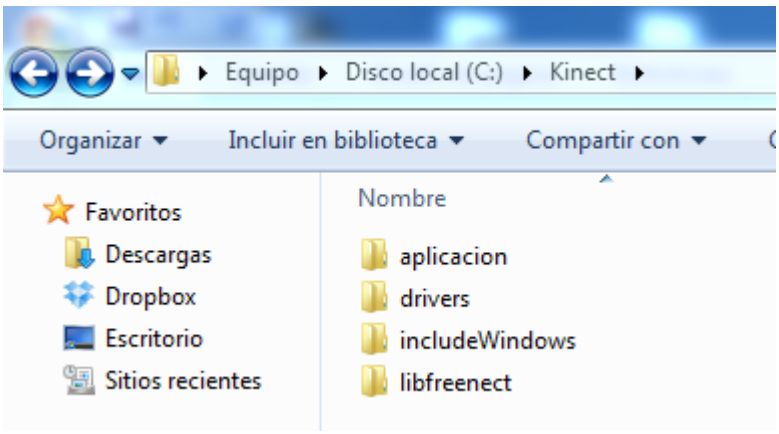
Con estos pasos realizados correctamente ya tenemos configurado nuestro proyecto para poder compilar.

Todos los archivos, librerías (.lib), archivos de cabecera (.h) se encuentran en una carpeta con nombre Kinect, que para tener la misma referencia se la ubicará en la raíz del disco en la unidad C: (Ver ANEXO Figura 7)

Las carpetas que se incluyen son las siguientes:

- Aplicación
- Drivers
- includeWindows
- libfreenect

En la carpeta aplicación se encuentra el proyecto, en la carpeta drivers los controladores y dependencias del Kinect, en includeWindows se encuentra el archivo de cabecera que fue modificado para que funcione una librería y comandos para controlar las teclas y el puntero en Windows, y en libfreenect la librería como tal, donde se encuentra el código fuente, los archivos de cabecera, los wrapper y librerías compiladas. El OpenCV se encuentra en el directorio raíz de la unidad C: en C:\OpenCV2.1, que se la descarga de internet pero que también está incluida en otra carpeta aparte el instalador.



ANEXO Figura 7 Carpetas que se incluyen en el proyecto

Siguiendo estos pasos queda configurado el proyecto.

El código del proyecto se encuentra en la carpeta: C:\of_v0071_vs2010_release\examples\addons\3DModelLoaderExample\src en el cual se encuentran tres archivos: main.cpp, testApp.cpp y testApp.h, en el main.cpp y testApp.h se encuentran la parte de configuración y las llamadas a los archivos de cabecera de las librerías y en el testApp.cpp se encuentra el código de la implementación y las funciones que se implementaron y se utilizan.

ANEXO 2: TABLA DE LAS CONSTANTES

Nombre simbólico	Valor (hexadecimal)	Equivalente al teclado o ratón
VK_LBUTTON	01	Left mouse button
VK_RBUTTON	02	Right mouse button
VK_CANCEL	03	Control-break processing
VK_MBUTTON	04	Middle mouse button (three-button mouse)
VK_BACK	08	BACKSPACE key
VK_TAB	09	TAB key
VK_CLEAR	0C	CLEAR key
VK_RETURN	0D	ENTER key
VK_SHIFT	10	SHIFT key
VK_CONTROL	11	CTRL key
VK_MENU	12	ALT key
VK_PAUSE	13	PAUSE key
VK_CAPITAL	14	CAPS LOCK key
VK_ESCAPE	1B	ESC key
VK_SPACE	20	SPACEBAR
VK_PRIOR	21	PAGE UP key
VK_NEXT	22	PAGE DOWN key
VK_END	23	END key
VK_HOME	24	HOME key
VK_LEFT	25	LEFT ARROW key
VK_UP	26	UP ARROW key
VK_RIGHT	27	RIGHT ARROW key
VK_DOWN	28	DOWN ARROW key
VK_SELECT	29	SELECT key
VK_PRINT	2A	PRINT key
VK_EXECUTE	2B	EXECUTE key
VK_SNAPSHOT	2C	PRINT SCREEN key
VK_INSERT	2D	INS key
VK_DELETE	2E	DEL key
VK_HELP	2F	HELP key
	30	0 key
	31	1 key
	32	2 key
	33	3 key
	34	4 key
	35	5 key
	36	6 key
	37	7 key
	38	8 key

ANEXO Tabla 1 Valor hexadecimal para los comandos en Windows OS [33]

Nombre simbólico	Valor (hexadecimal)	Equivalente al teclado o ratón
	39	9 key
	41	A key
	42	B key
	43	C key
	44	D key
	45	E key
	46	F key
	47	G key
	48	H key
	49	I key
	4A	J key
	4B	K key
	4C	L key
	4D	M key
	4E	N key
	4F	O key
	50	P key
	51	Q key
	52	R key
	53	S key
	54	T key
	55	U key
	56	V key
	57	W key
	58	X key
	59	Y key
	5A	Z key
VK_NUMPAD0	60	Numeric keypad 0 key
VK_NUMPAD1	61	Numeric keypad 1 key
VK_NUMPAD2	62	Numeric keypad 2 key
VK_NUMPAD3	63	Numeric keypad 3 key
VK_NUMPAD4	64	Numeric keypad 4 key
VK_NUMPAD5	65	Numeric keypad 5 key
VK_NUMPAD6	66	Numeric keypad 6 key
VK_NUMPAD7	67	Numeric keypad 7 key
VK_NUMPAD8	68	Numeric keypad 8 key
VK_NUMPAD9	69	Numeric keypad 9 key

ANEXO Tabla 2 Valor hexadecimanl para los comandos en Windows OS [33]

Nombre simbólico	Valor (hexadecimal)	Equivalente al teclado o ratón
VK_SEPARATOR	6C	Separator key
VK_SUBTRACT	6D	Subtract key
VK_DECIMAL	6E	Decimal key
VK_DIVIDE	6F	Divide key
VK_F1	70	F1 key
VK_F2	71	F2 key
VK_F3	72	F3 key
VK_F4	73	F4 key
VK_F5	74	F5 key
VK_F6	75	F6 key
VK_F7	76	F7 key
VK_F8	77	F8 key
VK_F9	78	F9 key
VK_F10	79	F10 key
VK_F11	7A	F11 key
VK_F12	7B	F12 key
VK_F13	7C	F13 key
VK_F14	7D	F14 key
VK_F15	7E	F15 key
VK_F16	7F	F16 key
VK_F17	80H	F17 key
VK_F18	81H	F18 key
VK_F19	82H	F19 key
VK_F20	83H	F20 key
VK_F21	84H	F21 key
VK_F22	85H	F22 key
VK_F23	86H	F23 key
VK_F24	87H	F24 key
VK_NUMLOCK	90	NUM LOCK key
VK_SCROLL	91	SCROLL LOCK key
VK_LSHIFT	A0	Left SHIFT key
VK_RSHIFT	A1	Right SHIFT key
VK_LCONTROL	A2	Left CONTROL key
VK_RCONTROL	A3	Right CONTROL key
VK_LMENU	A4	Left MENU key
VK_RMENU	A5	Right MENU key
VK_PLAY	FA	Play key
VK_ZOOM	FB	Zoom key

ANEXO Tabla 3 Valor hexadecimal para los comandos en Windows OS [33]

BIBLIOGRAFÍA

[1] **Comunidad as3Kinect,** librería de controladores para Kinect de la Xbox 360, Marzo 15 2012 [Web]. Disponible en: http://www.as3Kinect.org/.

[2] **Bradski Gary,** OpenCVWiki computer vision library, Abril 20 2012 [Web]. Disponible en: http://opencv.willowgarage.com/wiki/.

[3] **Neil Schwarz,** Estudio de usabilidad con Kinect, Diciembre 17 2010 [Web]. Disponible en: http://www.cxpartners.co.uk/cxblog/Kinect_gestural_interfaces_-_a_usability_study/.

[4] **Jakob Nielsen**, Kinect Gestural UI: First Impressions, Diciembre 27, 2010 [Web]. Disponible en: http://www.useit.com/alertbox/Kinect-gesture-ux.html.

[5] **Comunidad OpenKinect,** librerías de código abierto para Kinect, Abril 7 2012. Disponible en: http://openKinect.org/wiki/Main_Page

[6] **Lieberman Zach, Watson Theodore, and Castro Arturo** ("el núcleo del API"), con ayuda de la comunidad **OpenFrameworks,** API de código abierto C++ para procesos creativos y fáciles uso, Junio 8 2012. Disponible en: http://www.openframeworks.cc/

[7] **Microsoft,** Visual Studio 2010 C++ Express Edition**,** se utiliza este IDE de Microsoft para integrar las librerías utilizadas, Junio 12 2012. Disponible en: http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express

[8] **Burguera John,** Interface de usuario Natural**,** Experiencia de usuarios para Kinect, Julio 7 2011. Disponible en: http://www.naturallyinterfacing.com/2011/07/disenando-para-Kinect-nuestro-test-de.htmlv

[9] **Qurren,** Imagen ratón, Un ratón de computador con una rueda en medio de los botones, Abril 11 2006. Disponible en: http://commons.wikimedia.org/wiki/File:Wheel_mouse.JPG?uselang=es

[10] **Wtshymanski,** imagen teclado, Teclado de un computador, un Televideo 925, Enero 3 2006. Disponible en: http://commons.wikimedia.org/wiki/File:Televideo925Terminal.jpg?uselang=es

[11] **Microsoft,** Kinect dispositivo de entrada para la Xbox 360, Noviembre 4 2010. Disponible en: http://www.xbox.com/en-US/KINECT

[12] **Wikipedia,** Frustum definición en gráficos por computadora y en geometría, Junio 30 2012. Disponible en: http://es.wikipedia.org/wiki/Frustum

[13] **Wikipedia,** Frame definición para este estudio, Mayo 14 2012. Disponible en: http://es.wikipedia.org/wiki/Frame

[14] **Murillo Alejandro,** Diferencia entre Kinect para Windows y Kinect para Xbox 360**,** muestra las diferentes características entre estos dispositivos, Marzo 1 2012. Disponible en: http://blog.Kinectfordevelopers.com/2012/03/01/diferencias-entre-Kinect-xbox-360-y-Kinect-for-windows/

[15] **Gold Standard Group,** OpenGL (Open Graphics Library), es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D, Febrero 15 2012. Disponible en: http://www.opengl.org/

[16] **Bosque Jesús,** Imagen Kinect for Xbox 360, Partes que conforman el Kinect, Noviembre 15 2010. Disponible en: http://xbox360.ongames.com/noticia_ya-hay-mas-de-un-millon-de-Kinect-vendidos-en-el-mundo_Kinect

[17] **Ifixit,** Imagen Kinect desarmado, Partes internas y externas del Kinect, Junio 3 2012. Disponible en: http://www.ifixit.com/

[18] **MithrandirMage,** imagen Frustum**,** Representación del frustum en gráficos por computadora, Junio 18 2012. Disponible en: http://en.wikipedia.org/wiki/Viewing_frustum

[19] **Nexobit Axel,** Imagen NUI, Representación Interfaz natural de usuario, Enero 6 2010. Disponible en: http://nexobit.com/2010/01/06/windows-8-distinto-a-todo-lo-conocido/

[20] **Code Laboratories,** CL NUI**,** Librería para usar el Kinect de Microsoft en Windows 7, Diciembre 8 2010. Disponible en: http://codelaboratories.com/kb/nui

[21] **Code Laboratories,** Imagen CL NUI**,** Imagen del test tomada del website CL NUI Device Test, Diciembre 7 2010. Disponible en: http://codelaboratories.com/kb/nui

[22] **Wikipedia,** Middleware, Significado definición de wikipedia, Septiembre 12 2012. Disponible en: http://es.wikipedia.org/wiki/Middleware

[23] **Organización OpenNI,** Framework de OpenNI para sensar movimientos de 3 dimensiones, Marzo 13 2012. Disponible en: http://openni.org/About.aspx

[24] **Organización OpenNI,** Imagen Representacion de OpenNI, Imagen tomada de la página de OpenNI, Marzo 13 2012. Disponible en: http://openni.org/Documentation/ProgrammerGuide.html

[25] **Harishankar Narayanan**, Reconocimiento de gestos**,** desarrollador de este proyecto, Julio 9 2012. Disponible en: http://www.youtube.com/user/harishankarn

[26] **Evoluce GmbH,** Empresa alemana dedicada a tecnología multi – touch e interfaces naturales de usuario, Septiembre 7 2012. Disponible en: http://www.evoluce.com

[27] **Kooima Robert**, Kinect + 3D TV**,** insvestigador y creador de una aplicación realidad virtual inmersiva a bajo costo, Febrero 26 2011. Disponible en : http://csc.lsu.edu/~kooima/installations.html

[28] **Universidad Southtern Carolina**, FAAST**,** desarrolla un conjunto de librerías para el uso en realidad virtual, Diciembre 25 2010. Disponible en: http://projects.ict.usc.edu/mxr/work/ y el ejemplo de la aplicación en : http://www.youtube.com/watch?feature=player_embedded&v=62wj8eJ0FHw

[29] **Royal Institute of Technology,** Kinect + Google Earth**,** Proyecto de curso de otoño, Stockholm, Diciembre 11 2011. Disponible en: http://www.youtube.com/watch?v=5vYnwiCkk-g

[30] **Brother Joel,** Kinmote, Aplicación que Controla juegos y aplicaciones con el Kinect para Xbox de Microsoft, Junio 8 2012. Disponible en: http://www.kinemote.net/

[31] **Microsoft,** Windows API, Los comandos para controlar Windows son tomados del API, Junio 30 2012. Disponible en : http://msdn.microsoft.com/en-us/library/windows/desktop/ff657751(v=vs.85).aspx

[32] **Microsoft,** Mouse Event Function, Funciones del Win API para controlar el ratón, Julio 3 2012. Disponible en: http://msdn.microsoft.com/en-us/library/windows/desktop/ms646260(v=vs.85).aspx

[33] **Microsoft,** Virtual Key codes (Constantes), Constantes simbólicas que se utilizan para controlar aplicaciones y ventanas, Julio 8 2012. Disponible en: http://delphi.about.com/od/objectpascalide/l/blvkc.htm

[34] **Microsoft,** Constantes Win API, Constantes que se pueden utilizar con la función virtual key codes, Julio 9 2012. Disponible en: http://msdn.microsoft.com/en-us/library/windows/desktop/dd375731(v=vs.85).aspx ó http://msdn.microsoft.com/en-us/library/windows/desktop/ff468861(v=vs.85).aspx

[35] **Esqueda Elizondo José Jaime y Palafox Maestre Luis Enrique,** Fundamentos de procesamiento de imágenes, Motion Detection Libro de Procesamiento digital de imágenes, Universidad Autónoma de Baja California, Edición 2005. Disponible en: http://books.google.com.ec/books?id=h4Gj8GuwPVkC&lpg=PA1&pg=PA1#v=onepage&q&f=false