
VNIVERSIDAD NACIONAL
AVTNOMA DE
MEXICO

# UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

## REALIDAD AUMENTADA MEDIANTE GRÁFICOS 3D, SEGUIMIENTO Y LENTES DE REALIDAD VIRTUAL

**TESIS**
**QUE PARA OPTAR POR EL GRADO DE:**
**MAESTRO EN CIENCIAS (COMPUTACIÓN)**

**PRESENTA:**
**DANIELA LEÓN VÁZQUEZ**

**TUTORES PRINCIPALES:**

| | |
|---|---|
| **DR. JORGE ALBERTO MÁRQUEZ FLORES** | **CCADET – UNAM** |
| **DR. ALFONSO GASTÉLUM STROZZI** | **CCADET – UNAM** |

**MIEMBROS DEL COMITÉ TUTOR:**

| | |
|---|---|
| **DR. FERNANDO ARÁMBULA COSÍO** | **CCADET – UNAM** |
| **DR. EDGAR GARDUÑO ÁNGELES** | **IIMAS – UNAM** |
| **DR. MIGUEL ÁNGEL PADILLA CASTAÑEDA** | **CCADET – UNAM** |

**MÉXICO, D. F. ENERO 2016**

# AGRADECIMIENTOS

A mis tutores, el Dr. Jorge Márquez y el Dr. Alfonso Gastélum por sus consejos y enseñanzas a lo largo del camino.

A mi comité tutor y sinodales por sus aportaciones.

A mis profesores de la maestría por sus conocimientos.

Al Posgrado en Ciencias e Ingeniería de la Computación de la UNAM.

Al laboratorio de Análisis de Imágenes y Visualización del CCADET-UNAM y a la Unidad de Investigación y Desarrollo Tecnológico del CCADET en el Hospital General de México por el espacio de trabajo brindado.

Al Dr. Patrice Delmas y la Universidad de Auckland en Nueva Zelanda por permitirme realizar una estancia de investigación en el laboratorio “Intelligent Vision Systems”.

A mi familia por su cariño y apoyo incondicional en todo momento.

A mis amigos por acompañarme y compartir la experiencia.

Al CONACYT por el apoyo económico durante este proceso.

# ÍNDICE

# RESUMEN

En los últimos años la Realidad Aumentada ha sido considerada como un elemento novedoso y revolucionario en el desarrollo de aplicaciones. Esto se debe a su capacidad de enlazar la realidad con información que de otra manera sería inaccesible. Tiene el potencial de presentar nuevas formas de visualización e interacción, permitiendo un mejor desempeño en las actividades realizadas, gracias a los elementos visuales que se integran a la realidad física.

El objetivo del presente trabajo es la elaboración de un sistema de Realidad Aumentada, basado en la integración de diversos elementos de Realidad Virtual, seguimiento y graficación en tres dimensiones.

Se utilizó el casco de Realidad Virtual Oculus Rift como base para la construcción del sistema, añadiendo el sensor de movimiento Leap Motion, así como un par de cámaras Logitech c920 de alta resolución. Además de la implementación física, se desarrolló un programa que permite la alineación de las imágenes obtenidas de distintas fuentes.

Las imágenes obtenidas por las cámaras representan la escena del mundo real, la imagen obtenida por el sensor nos brinda la posición de las manos frente al usuario, y de la escena generada por medio de los gráficos 3D se obtienen los objetos virtuales.

El programa funciona de forma paralela, procesando las imágenes obtenidas de cada una de las fuentes sobre un hilo de ejecución distinto, lo que permite una ejecución en tiempo real. Este programa consigue ajustar todas las imágenes correspondientes a un mismo cuadro de tiempo en la escena, de modo que la imagen resultante muestre todos los elementos desde el mismo punto de vista. De este proceso se obtienen las dos imágenes a mostrar.

La pantalla del casco Oculus Rift permite el despliegue de la escena de manera que cubra por completo el campo de visión del usuario por medio de lentes, los cuales ocasionan distorsión en la imagen percibida. Esta distorsión debe ser corregida aplicando una distorsión de barril a las imágenes resultantes.

# Capítulo I. INTRODUCCIÓN Y ANTECEDENTES

## **1.1** *Introducción.*

La **Realidad Aumentada** es una técnica que ha sido estudiada desde 1962, cuando se acuñó por primera vez el término tras la invención de un simulador, llamado Sensorama, que hacía uso de imágenes, sonido, vibraciones y olfato (*Burdea, G. C. y Coiffet, P., 2003*). Sin embargo, el desarrollo de nuevas tecnologías ha permitido mejorar las técnicas aplicadas con anterioridad así como la implementación de nuevos sistemas y dispositivos dando al usuario una mayor inmersión e interacción con el ambiente.

En la actualidad existen diferentes dispositivos que nos brindan la experiencia de una Realidad Aumentada, éstos van desde teléfonos inteligentes, tabletas, hasta tecnologías específicamente desarrolladas como lo son Google Glass [1] y los más recientes HoloLens [2] de Microsoft.

El sistema desarrollado para este trabajo de tesis se implementa para poner a prueba diferentes aplicaciones en particular relacionadas al campo de la medicina. Los avances tecnológicos han animado a incluirlos, cada vez más, como parte de las actividades profesionales. Esto permite a los expertos, médicos en el caso de nuestras aplicaciones, disponer de mejores herramientas para desempeñar sus capacidades de una manera más rápida, además de la oportunidad de realizar tareas antes imposibles, riesgosas o en extremo imprecisas.

En los últimos años la Realidad Aumentada ha sido considerada como un elemento novedoso y revolucionario en el desarrollo de aplicaciones. Esto se debe a su capacidad de enlazar la realidad con información que de otra manera sería inaccesible. Tiene el potencial de presentar nuevas formas de visualización e interacción, permitiendo que el desempeño de los profesionales mejore, gracias a los elementos visuales combinados de manera eficiente, en tiempo real y el la posición correcta, con la realidad física.

[1] https://developers.google.com/glass/
[2] http://www.microsoft.com/microsoft-hololens/en-us

### **1.1.1** *Organización de la Tesis.*

Dentro del primer capítulo se explican los antecedentes de la Realidad Aumentada desde diferentes puntos de vista. Se definen las nociones principales, y se describen las diferencias y semejanzas con otros conceptos relacionados. También se exponen los distintos arreglos que utilizan esta tecnología y algunas de las aplicaciones que comparten las mismas ideas que este proyecto.

El segundo capítulo describe los distintos componentes que forman parte del sistema propuesto. Se despliegan las características principales de cada uno y su funcionamiento por sí solo. Además del equipo utilizado para la construcción del equipo se tiene una breve descripción de las bibliotecas utilizadas en los programas desarrollados.

El capítulo tercero plantea las dos principales etapas de la elaboración de este sistema de Realidad Aumentada: la construcción y la programación. En la etapa de construcción se detalla cómo los componentes del sistema fueron montados para generar el dispositivo propuesto, mientras que en la etapa de programación se encuentra descrito cada módulo necesario para el correcto funcionamiento coordinado del sistema.

En el cuarto capítulo se muestran los resultados obtenidos al finalizar el experimento, así como la evaluación de las pruebas de interacción y navegación realizadas.

Para finalizar el trabajo se encuentra la conclusión del mismo, así como trabajo a futuro y los puntos a mejorar de este sistema.

### **1.1.2** *Objetivos*

#### **1.1.2.1** *Objetivo principal*

El objetivo central del presente trabajo consiste en desarrollar e implementar un sistema de Realidad Aumentada basado en la integración de elementos de Realidad Virtual, seguimiento y graficación en tres dimensiones, con el fin de ser utilizado en diversas aplicaciones, particularmente aplicaciones médicas.

#### **1.1.2.2** *Objetivos específicos*

Para poder cumplir con el objetivo principal, se consideraron además los siguientes objetivos específicos:

- Manejo de modelos 3D (acceso, preparación, procesamiento requerido y visualización).
- Acoplamiento de las entradas (cuadros de video y gráficos) provenientes de distintas fuentes (cámaras, sensor de seguimiento y computadora).
- Integración de los elementos en tiempo real.
- Integración mecánica del sistema sobre el casco de Realidad Virtual Oculus Rift.
- Pruebas: validación y evaluación de navegación e interacción en el ambiente.

### **1.1.3** *Aplicaciones*

La Realidad Aumentada nos brinda un gran número de posibilidades de interacción, esto hace posible su integración sobre las diferentes áreas de conocimiento como son la educación, la arquitectura, el entretenimiento, el arte y la medicina, así como en cualquier ámbito en el que sea posible el uso de un dispositivo inteligente además de la interacción entre imágenes del mundo real y aquellas generadas de los mundos virtuales.

Estas aplicaciones pueden ir desde algo muy elaborado como un manual de reparación donde, a través de los lentes, se observa cómo aparecen las instrucciones, se identifican las piezas a manipular, etc., hasta algo sencillo como desplegar un objeto virtual en la pantalla de la cámara de un teléfono inteligente. Además, se tiene lo que se puede desarrollar con un nivel de dificultad intermedio a estos dos ejemplos.

A continuación se describen tres de las principales áreas a las que el desarrollo de este proyecto va dirigido:

#### **1.1.3.1** *Entrenamiento*

Cuando se requieren realizar prácticas de ciertas actividades limitadas por altos costos o situaciones difíciles de recrear, las aplicaciones de Realidad Aumentada pueden ser utilizadas para mejorar el entrenamiento en estos escenarios. Con el desarrollo de las nuevas tecnologías ahora es posible mostrar ambientes virtuales mucho más realistas donde el usuario se sumerge de formas que antes no eran posibles.

Dependiendo de las habilidades de los usuarios y lo que quieran realizar, cada uno toma diferentes decisiones y rutas de aprendizaje sin presión de tiempo ni consecuencias reales (por ejemplo: lesiones o daños a un paciente) en caso de que se cometan errores durante la práctica.

Esta tecnología permite que los usuarios tengan un entrenamiento activo, tanto físico como psicológico. El propósito de estas aplicaciones es darle al usuario la oportunidad de abordar la solución de un problema tomando distintos puntos de vista, preparándolo para tener un mejor desempeño al momento de aplicar los conocimientos, adquiridos en los entrenamientos, en situaciones de la vida real.

#### **1.1.3.2** *Rehabilitación*

Los ejercicios de rehabilitación, o fisioterapia, son practicados actualmente por un profesional de una forma manual y metódica (**Figura 1**). Es el terapeuta quién tiene el criterio de evaluar la ejecución del paciente, es decir, si está realizando el ejercicio de manera adecuada. Al ser una evaluación subjetiva, estos métodos pueden mejorarse, complementando el trabajo del profesional, con el uso de aplicaciones de Realidad Aumentada, permitiendo, además, que el paciente realice ciertos ejercicios desde casa.

***Figura 1**. Fisioterapia de rodilla y hombro realizada por un profesional* [3]

Existen distintas formas en que se pueden aplicar estos métodos, una de ellas *(Burke, J. et al., 2010)* nos muestra cómo la Realidad Aumentada permite al usuario utilizar objetos reales para interactuar con ambientes generados por computadora. Con marcadores adheridos a estos objetos para poder seguir su posición y orientación, el sistema presenta distintas tareas a realizar sobre el ambiente virtual (**Figura 2**).

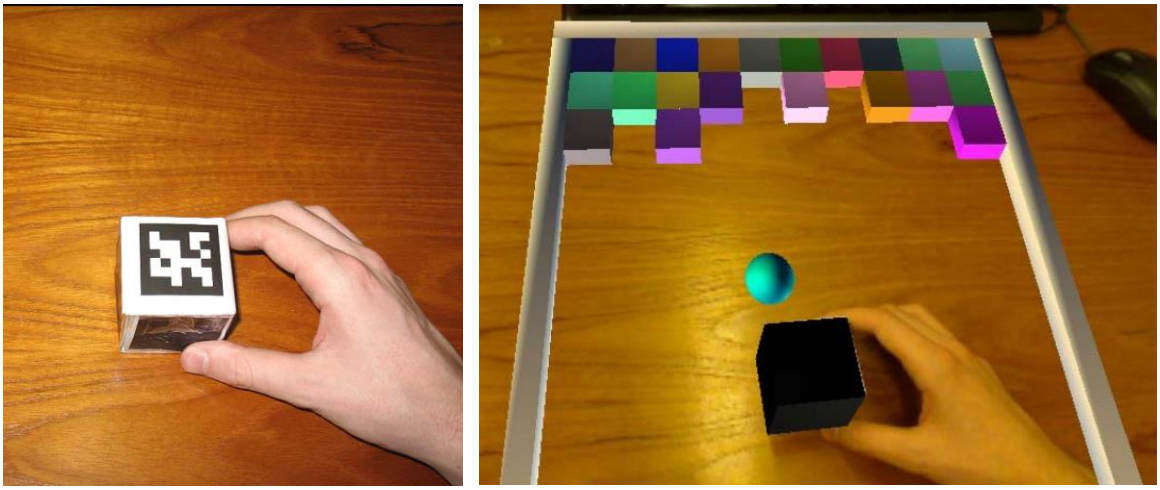

***Figura 2.** "Brick'a'Break" (Burke, J. et al., 2010). Escena del mundo real, el usuario sostiene un cubo con un marcador adherido (izquierda). La escena del mundo real ha sido aumentada con la escena virtual, creando un juego para romper bloques (derecha).*

[3] http://www.buenaforma.org/2011/05/07/rehabilitacion-deportiva-movilidad-articular-kaltenborn/

Otra forma en la que estas aplicaciones pueden ser de utilidad en la rehabilitación es utilizando dispositivos que reconozcan los movimientos del cuerpo humano, como Microsoft Kinect[4], monitoreando los movimientos que realiza el paciente. Al ser un sistema accesible, no es necesario ir a los centros de rehabilitación para realizar las terapias. Estas aplicaciones muestran los objetivos a realizar por medio de actividades cotidianas, así como retroalimentación en tiempo real sobre la ejecución de los ejercicios.

***Figura 3.*** *Aplicaciones de rehabilitación utilizando Microsoft Kinect. Rehabilitación por medio de videojuegos[5] (arriba). Retroalimentación de los ejercicios a realizar[6] (abajo).*

[4] https://dev.windows.com/en-us/kinect
[5] http://spanishcompanies-medica.com/virtualware/16.html
[6] http://iadeu.crearjoomla.com/fisioterapia/253-kinect.html

#### **1.1.3.3** *Aplicaciones en la medicina*

Poco a poco la Realidad Aumentada ha dejado de ser sólo un medio de entretenimiento y se incorpora a propósitos más prácticos y con un mayor aporte al proporcionar nuevas formas de visualización de los elementos y facilitando el desempeño en las tareas. Un ejemplo de ello lo vemos en el campo de la medicina.

Esta tecnología se abre camino rápidamente entre los profesionales de la medicina. Las características novedosas de la Realidad Aumentada permiten que sea implementada como una herramienta útil y práctica en las actividades que deben realizar los médicos, obteniendo un mejor desempeño en el diagnóstico y tratamiento de sus pacientes. Algunos ejemplos de estas aplicaciones se presentan más adelante, dentro de esta misma sección.

Teniendo en cuenta lo anterior se puede considerar que los sistemas de Realidad Aumentada cumplen el papel de simplificar en cierta medida algunos de los procesos quirúrgicos más complejos. Apoyándose en estudios preoperatorios, por ejemplo: mediante imágenes de Resonancia Magnética Nuclear, es posible recopilar información anatómica y funcional del interior del paciente de una manera no invasiva.

Actualmente se cuenta con la capacidad de superponer en tiempo real, sobre una superficie del cuerpo, la reconstrucción en tres dimensiones de las estructuras internas de la persona, como si la superficie real fuera transparente (**Figura 4**). De esta manera, se pueden obtener resultados con una mayor garantía de seguridad para el paciente *(Navab, N. et al., 2012)*.

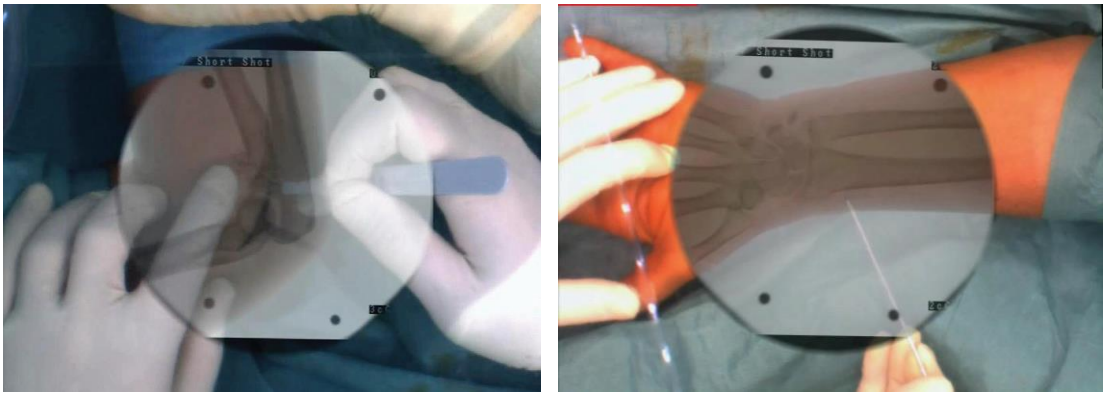

***Figura 4****. Ejemplos clínicos donde una imagen de Rayos X se superpone sobre el paciente durante la cirugía. La imagen final es presentada al médico en una pantalla.*

Además, teniendo las plataformas adecuadas, los médicos pueden brindar un apoyo no sólo durante la cirugía, sino también en la atención pre y postoperatoria. Algunos de los dispositivos de uso general sobre los que es posible aplicar la Realidad Aumentada pueden ser utilizados en tareas médicas sin necesidad de ser aparatos especializados.

Podemos encontrar beneficios no sólo en el tratamiento directo de los pacientes, sino enfocados al área de investigación de nuevos procedimientos no invasivos para la caracterización de diversas enfermedades. La Realidad Aumentada nos trae beneficios dentro de los que sobresalen el análisis de imágenes médicas, simulación de sistemas y procedimientos, entrenamiento, y visualización interactiva y concurrente con información gráfica y textual de diversas fuentes.

Retomando un poco las aplicaciones sobre entrenamiento, la medicina es un campo donde los estudiantes adquieren experiencia sobre la marcha. Los jóvenes médicos pueden hacer uso de esta tecnología de manera que sus prácticas se realizan en un ambiente virtual controlado. Los apoyos visuales brindados permiten explorar las diferentes condiciones que se pueden presentar durante el procedimiento sin poner en riesgo a un paciente.

## **1.2** *Antecedentes*

Retomando el objetivo general del proyecto, la coordinación de distintos elementos, reales y virtuales, para la conformación de un solo sistema, es importante conocer los principios básicos. Paul Milgram y Fumio Kishino introducen en 1994 el concepto de un "Continuo Real – Virtual" o *"Reality – Virtuality Continuum"* (**Figura 5**) *(Kishino, F. y Milgram, P., 1994)* en donde se describe una escala continua que va desde la Realidad (mundo real) hasta la Realidad Virtual (mundo virtual).

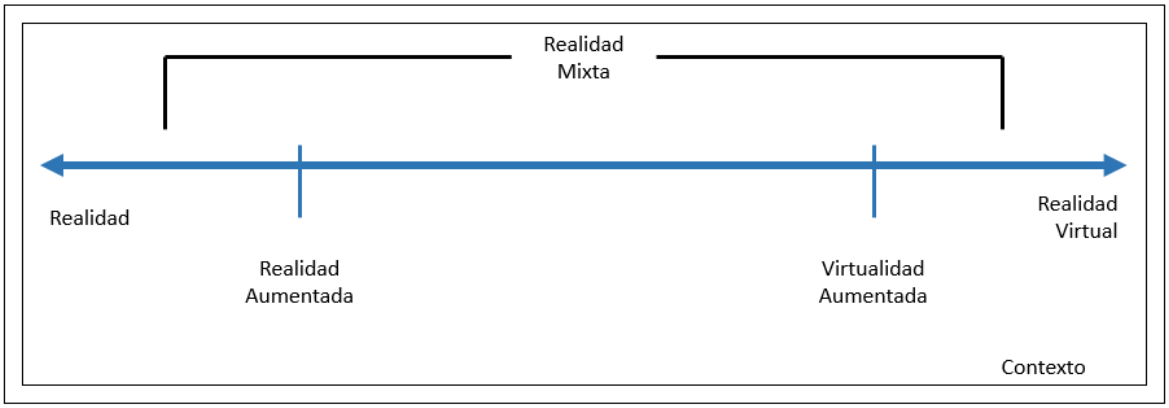


***Figura 5**. "Continuo Real – Virtual", presentado por Paul Milgram y Fumio Kishino (1994) en donde se aprecian los principales puntos entre el mundo real y el virtual.*

En el diagrama se puede observar que los ambientes reales y los virtuales se encuentran en extremos opuestos, estos son los límites del Continuo, dejando en el centro todas las posibilidades de combinar los dos conceptos *(Kishino, F. y Milgram, P., 1994)*.

Empezando por sus definiciones, el extremo izquierdo muestra la **Realidad** tradicional, esto se refiere a ambientes que son construidos meramente por elementos reales. Los elementos pueden o no ser observados a través de algún medio, es decir, puede referirse a la percepción de un entorno real en el cual nos encontramos físicamente o a la percepción de un entorno real a través de una imagen o su reproducción en otro momento (por ejemplo, tomada por una cámara y proyectada en una pantalla o sobre lentes de Realidad Virtual). Existe una inmersión e interacción en tiempo real por parte del usuario.

Por el contrario, en el extremo derecho se encuentra la **Realidad Virtual** que representa un ambiente en el cual el usuario u observador está totalmente inmerso, conformado puramente por elementos ficticios o virtuales, usualmente sintetizados por computadora, con los que se puede interactuar en tiempo real.

Todo el espacio entre los extremos explicados anteriormente se considera como una **Realidad Mixta**, esto es una serie de diversos ambientes en el cual elementos del Mundo Real y Virtual coexisten (se superponen y combinan) sin ninguna distinción. Dentro de esta noción destacan los siguientes ambientes particulares:

a) **Virtualidad Aumentada:** se refiere a los casos cuando un escenario creado puramente con elementos virtuales se complementa con algún elemento real.
b) **Realidad Aumentada:** es el caso inverso a la Virtualidad Aumentada, es decir, cuando a cualquier ambiente considerado como real se le añaden elementos virtuales, generados por computadora.

La interacción le permite al usuario seleccionar información, cambiarla, moverla, enfocarse en un subconjunto de la misma o descartarla. Sin embargo, todas estas acciones y reacciones que tenga el sistema toman significado a través de un contexto. Dependiendo de la actividad que se quiera realizar, es el entorno en el que se debe desarrollar el sistema, por ejemplo: la visualización de ejercicios militares se debe llevar a cabo en el exterior y de una manera precisa, a diferencia de un experto estudiando el cambio climático que puede acceder a la información desde su laboratorio *(Roberts, J. C. et al., 2014)*.

Este contexto se ha ido mejorando con el paso de los años y la creación de nuevos dispositivos móviles. Antes de la llegada de estos dispositivos, la mayoría de las tareas estaban ligadas a un lugar en particular: para revisar el correo electrónico se debía estar frente a la computadora, o estar presente en una sala de conferencias para tener una reunión. La constante innovación en estos dispositivos nos permite una mayor interacción con el mundo, desde el real hasta el virtual, así como una manera más fácil y natural para realizar las tareas que antes eran más complejas.

Un sistema de Realidad Aumentada logra complementar el mundo real con ayuda de elementos virtuales (generados por computadora) aparentando existir en el mismo espacio. Lo anterior no se limita sólo al sentido de la vista, estos elementos virtuales pueden hacer referencia a cualquiera de los sentidos mientras permita una coordinación e interacción en tiempo real con nuestro entorno *(Azuma, R. et al., 2001)*. Sin embargo este proyecto toma un enfoque puramente visual, aunque puede ser complementado con retroalimentación táctil y sonora en caso de aplicaciones de entrenamiento y rehabilitación.

Enfocándonos en la parte gráfica de la Realidad Aumentada tenemos el concepto de visualización, la cual nos permite comunicar y percibir información a través de representaciones visuales. Gracias a la visualización los usuarios cuentan con elementos que les permiten entender e interactuar con la información con mayor facilidad e intuición. Ejemplos claros de esto son las gráficas de datos, donde con una imagen podemos analizar grandes cantidades de información.

Como se mencionó anteriormente, el medio visual no es la única forma de comunicar la información, pues existe una variedad de estímulos sensoriales que permiten, si no sustituir, complementar la retroalimentación visual. Por ejemplo: una señal auditiva indicando que se completó una tarea o acción.

Los dispositivos comerciales especializados en el área ya implementan estos estímulos sensoriales con diferentes modalidades de entrada y salida de información. Tenemos cascos de Realidad Virtual y Aumentada como lo son Oculus Rift[7], Google Glass y Microsoft HoloLens para el despliegue de la información, así como sensores kinestésicos, por ejemplo: Leap Motion[8] y Microsoft Kinect, permitiendo al usuario interactuar con el sistema.

Los precios de los dispositivos se vuelven cada vez más accesibles para el público, lo que provoca la exigencia de aplicaciones que cumplan sus expectativas, por lo que la visualización y todos los elementos que implica un sistema de Realidad Aumentada deben adaptarse a las nuevas capacidades de estos dispositivos para obtener el mayor provecho.

[7] https://www.oculus.com/en-us/
[8] https://www.leapmotion.com/

Los componentes principales del proceso de visualización se pueden describir con un flujo de información *(Roberts, J. C. et al., 2014)*. Como se puede apreciar en el diagrama (**Figura 6**) este flujo de información representa la entrada de información, la cual puede estar modificada, aumentada o filtrada, para después ser desplegada. El usuario hace uso de sus capacidades de razonamiento, reconocimiento y los sentidos para poder interactuar con esta información mostrada. Las interfaces son las que permiten que el usuario modifique, añada o elimine la información con la que está interactuando a través del ambiente. Por lo general, esta interacción se da dentro de un contexto que relaciona al usuario con lo que está percibiendo, permitiendo una mayor inmersión así como un mejor entendimiento.

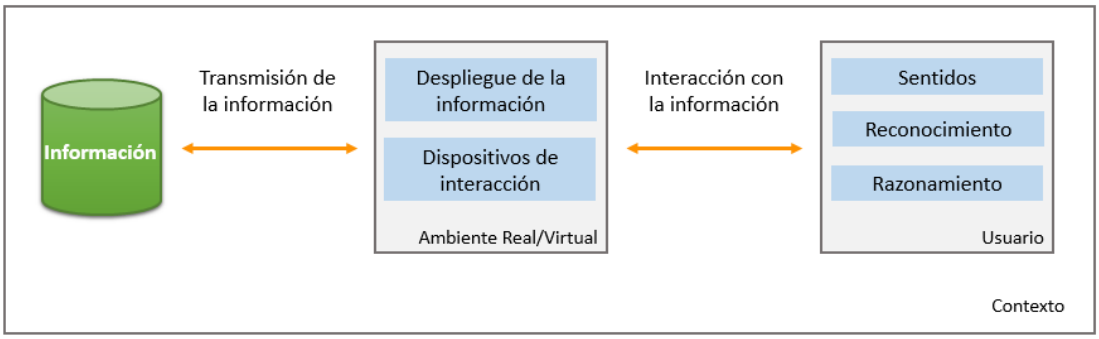


***Figura 6**. Flujo de la información hacia los elementos de despliegue, así como las capacidades del usuario utilizadas para reconocer e interactuar con el ambiente.*

### **1.2.1** *Métodos de visualización*

Los modelos u objetos virtuales son desplegados desde el mismo punto de vista desde el cual se toman las imágenes del mundo real a través de las cámaras. Esto es debido a que deben coincidir rasgos de los objetos virtuales con el objeto real equivalente, deben también estar alineados y usualmente mantener la misma escala. Existen diferentes métodos de visualización de Realidad Aumentada que realizan el registro o alineación de estas dos componentes, sin embargo, todos mantienen una metodología común. En general, esta metodología consta de dos etapas *(Carmigniani, J. et al., 2010)*: seguimiento y reconstrucción / reconocimiento.

En la primera etapa se buscan puntos de interés en las imágenes obtenidas por las cámaras, los puntos pueden ser características de la escena o marcadores, ya sea fiduciarios o puntos artificiales que destaquen en la escena. Estas características pueden ser obtenidas utilizando distintos algoritmos de procesamiento de imágenes como lo es detección de bordes. Las técnicas de seguimiento también se pueden dividir en dos grupos:

a) Basados en características: estos métodos consisten en encontrar la relación entre las características de la imagen en dos dimensiones y sus coordenadas en el espacio tridimensional. Por ejemplo, utilizar mapas de profundidad para estimar la posición de los objetos *(Lee, B. y Chun, J., 2010)*.
b) Basados en modelos: se utilizan modelos tridimensionales, como los modelos CAD, de los objetos que se están analizando para encontrar sus coordenadas espaciales correspondientes. Se puede tomar un modelo tridimensional creado a mano, representado paramétricamente por objetos complejos, como líneas, esferas y cilindros *(Comport, A. I. et al., 2003)*. Otro ejemplo es construir modelos texturizados, donde al combinar la información de la textura con la de los bordes del modelo, permite al sistema de seguimiento encontrar los bordes del objeto real con mayor facilidad *(Reitmayr, G. y Drummond, T.W., 2006)*.

Ambas técnicas buscan el mismo objetivo, encontrar la correspondencia entre las coordenadas de la imagen del punto o característica y las coordenadas en las que se encuentra en el mundo real. Habiendo encontrado esta relación es posible determinar la posición de la cámara proyectando las coordenadas espaciales del punto o característica a sus coordenadas observadas en la imagen, y minimizando las distancias obtenidas a sus coordenadas correspondientes (**Figura 7**).

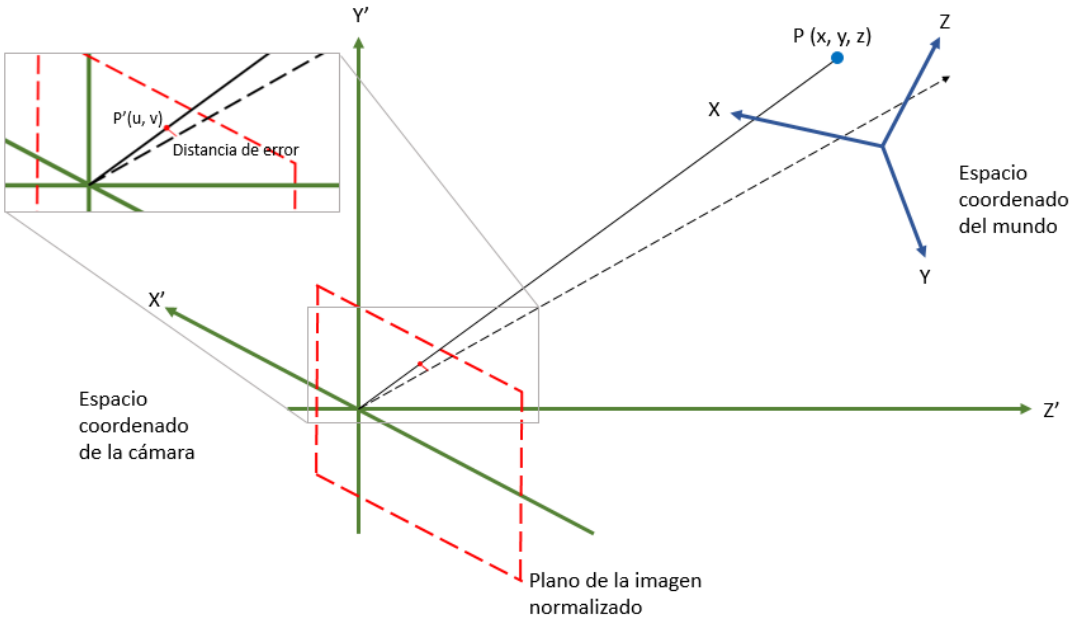


***Figura** 7. Proyección del espacio coordenado del mundo real al espacio coordenado de la cámara o imagen.*

Para la segunda etapa se utiliza la información obtenida en la primera, permitiendo reconstruir un sistema coordenado del mundo real. Esto permite conocer en qué posición del mundo real se encuentran los objetos que se están viendo a través de las imágenes obtenidas por las cámaras, y por lo tanto, conocer la posición sobre la imagen en donde debería colocarse el objeto virtual y con qué perspectiva para que parezca coexistir en el mundo real.

Para realizar el proceso de proyectar una escena en tres dimensiones a una imagen 2D se requiere revisar el modelo de la Cámara de *Pinhole* presentado a continuación, pues representa el modelo más simple de captura de imágenes por medio de una cámara. Este modelo plantea la relación matemática entre las coordenadas de un punto 3D en el espacio y su proyección en un plano utilizando una cámara ideal, siendo esta una cámara sencilla sin ningún tipo de lente, sólo con un pequeño orificio que permite el paso de la luz.

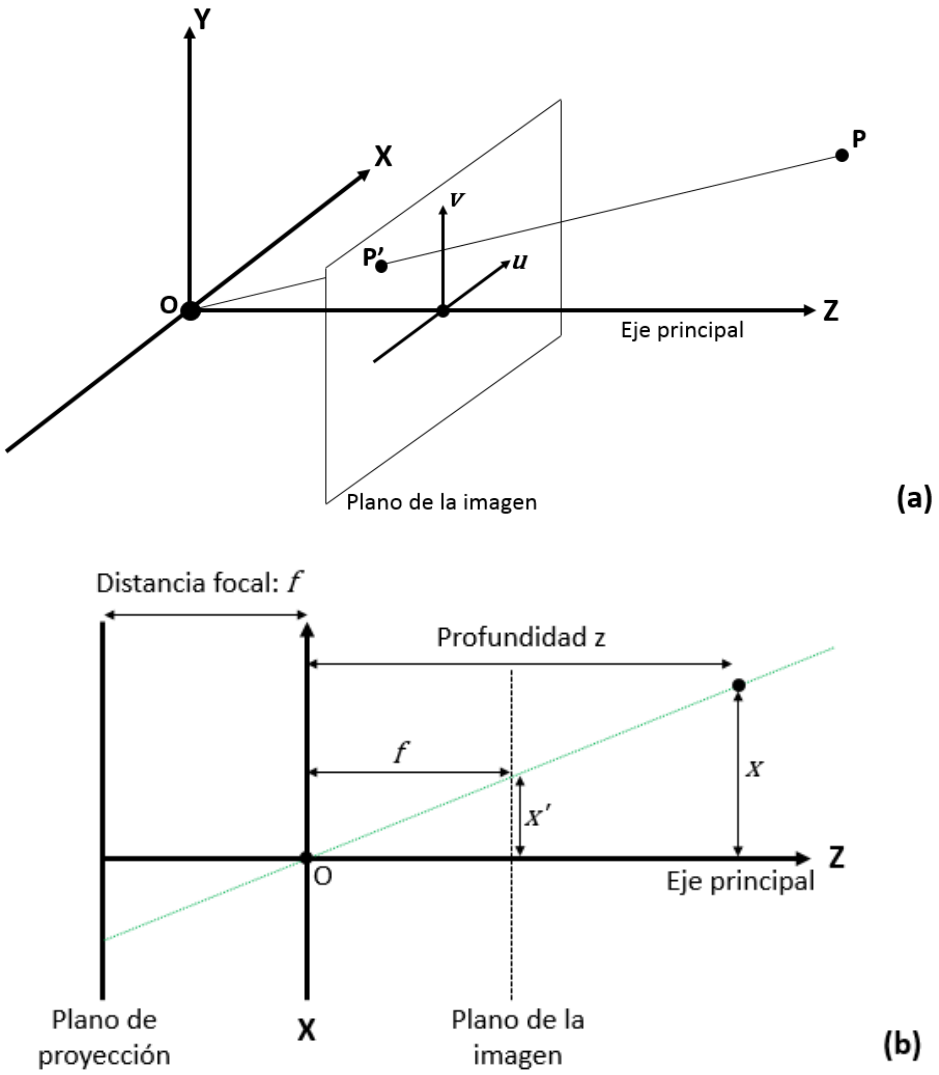


***Figura 8**. Proyección del espacio tridimensional a un plano siguiendo el modelo de la Cámara de Pinhole.*

La **Figura 8 (a)** muestra la geometría básica del modelo de la cámara de *Pinhole*. Se tiene un origen definido ***O***, que coincide con la posición de la apertura, o el centro de la cámara en este caso. El eje principal (***Z***) apunta en la misma dirección que la visión de la cámara. También se conoce el punto de intersección (**punto principal**) entre el eje principal y el plano de la imagen, además cuenta con las coordenadas de un punto ***P*** (*x, y,* z) en el espacio, y ***P'*** *(u, v)* que se obtiene como resultado de proyectar el primero sobre la imagen.

Siguiendo la base de este modelo se tiene una línea recta imaginaria, **línea de proyección**, que va desde el punto ***P*** hasta el plano de proyección, pasando por el origen del sistema. El plano de proyección, paralelo a los ejes ***X*** y ***Y***, se encuentra a una distancia focal ***f*** en dirección negativa sobre el eje principal, reduciendo el campo visual de la cámara. Como se puede observar en la **Figura 8 (b)**, existe un plano entre el origen y el punto 3D llamado **plano de la imagen**, que se encuentra a la misma distancia ***f*** del origen pero en dirección positiva, este representa la imagen 2D obtenida tras la proyección.

El objetivo principal de este modelo es obtener las coordenadas del objeto proyectado sobre el plano de la imagen. Para conseguirlo se sigue una serie de 4 pasos *(Tsai, R. Y., 1987)*, de los cuales se explican los tres primeros:

a) Transformación del sistema coordenado del mundo real (*x, y, z*) al sistema coordenado de la cámara (*x', y', z'*):

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \bar{t}, \qquad (1)$$

donde *R* es la matriz de rotación:

$$R \equiv \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix}, \qquad (2)$$

y *T* es el vector de traslación:

$$\bar{t} \equiv \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}. \qquad (3)$$

b) Transformación del sistema coordenado de la cámara (*x, y, z*) al sistema coordenado ideal (sin distorsión) de la imagen (*u, v*), utilizando el método de proyección de la geometría de la cámara de *Pinhole*. Si observamos en la **Figura 8 (a)**, se forman dos triángulos equivalentes, de distinto tamaño, que comparten la hipotenusa, de los cuales podemos obtener:

$$\frac{x'}{f} = \frac{x}{z'}, \qquad (4a)$$

$$\frac{y'}{f} = \frac{y}{z}. \qquad (4b)$$

Si tomamos en cuenta que $x'$ equivale a la coordenada horizontal $u$ del punto ***P*** sobre la imagen, y $y'$ a la coordenada $v$ podemos decir que:

$$u_u = f\frac{x}{z}, \qquad (5a)$$

$$v_u = f\frac{y}{z}. \qquad (5b)$$

c) Las ecuaciones anteriores definen una proyección ideal, en donde no existen factores que alteran el resultado de la imagen obtenida. Sin embargo, uno de los principales problemas es la distorsión ocasionada por las lentes de las cámaras. Las ecuaciones de distorsión radial serían:

$$u_d + d_u = u_u\,, \qquad (6a)$$

$$v_d + d_v = v_u\,, \qquad (6b)$$

donde $(U_d, V_d)$ es la coordenada distorsionada, o real, sobre el plano de la imagen, y

$$d_u = u_d(\kappa_1 r^2 + \kappa_2 r^4 + \cdots)\,, \qquad (7a)$$

$$d_v = v_d(\kappa_1 r^2 + \kappa_2 r^4 + \cdots)\,, \qquad (7b)$$

$$r = \sqrt{u_d^2 + v_d^2}\,, \qquad (8)$$

siendo $\kappa_i$ los coeficientes de distorsión y $r$ la distancia del punto principal (intersección del eje principal sobre el plano de la imagen) al punto que se está analizando.

### **1.2.2** *Dispositivos*

Como se menciona anteriormente, los sistemas de Realidad Aumentada no están limitados al sentido de la vista, se pueden tener distintos tipos de interacción con el ambiente, como lo son las pantallas táctiles, sensores de movimiento o cualquier otro dispositivo de entrada, por ejemplo: ratón y teclado de la computadora.

Los dispositivos principales involucrados en el despliegue de la información, permitiendo al usuario percibir su entorno, son las pantallas en donde se muestra e ambiente *(Carmigniani, J. et al., 2010)*. Se definen en tres principales categorías: proyección, *Handheld* o dispositivos portátiles y *Head Mounted Displays* (HMD) o cascos.

#### **1.2.2.1** *Realidad Aumentada mediante proyección*

Se define a la Realidad Aumentada basada en proyecciones como el uso de tecnología de proyección para aumentar y mejorar o complementar objetos y espacios del mundo real, de forma que las imágenes virtuales sean visibles en sus superficies *(Mine, M. et al., 2012)*.

Por lo general se utiliza uno o más proyectores distribuidos alrededor del objeto o superficie sobre el cuál se quiere mostrar la imagen. Este tipo de despliegue necesita un registro geométrico previo para que funcione correctamente, en especial si se trabaja con más de un proyector, pues se debe tomar en cuenta la diferencia de las imágenes a proyectar.

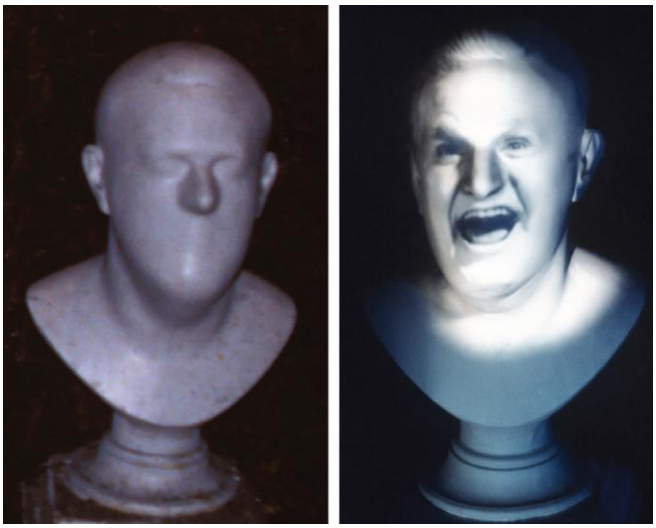

***Figura 9****. Animación por proyección de uno de los bustos cantantes, parte de la atracción "Mansión Embrujada" en uno de los parques de Disney (Mine, M. et al., 2012).*

Esta forma de Realidad Aumentada permite que sea una experiencia grupal, pues no es necesario utilizar ningún tipo de dispositivo intermedio para percibir los elementos virtuales. Otra ventaja principal es que presenta una menor latencia al compararlo con otras formas de despliegue, pues los elementos que se proyectan tienen un menor grado de movimiento relativo.

### **1.2.2** *Realidad Aumentada mediante dispositivos portátiles*

Tabletas y teléfonos inteligentes son los principales dispositivos móviles, a los cuales mucha gente tiene acceso. Todos los elementos necesarios para ejecutar una aplicación de Realidad Aumentada vienen incluidos en un equipo ligero, con pantallas de gran resolución y cámaras de alta definición. Al compararlos con otras modalidades de dispositivos podemos encontrar ciertas ventajas, como la accesibilidad en precio y disponibilidad mundial.

Estos equipos funcionan como un dispositivo dos en uno, pues ambos componentes principales se encuentran en el mismo: la pantalla y la cámara. Como la mayoría de los sistemas de Realidad Aumentada, la imagen capturada por la cámara es desplegada sobre la pantalla del dispositivo, procesándola y encontrando la posición en la cual se desplegará el objeto virtual. Muchas aplicaciones utilizan marcadores para facilitar esta tarea *(Gervautz, M. y Schmalstieg, D., 2012)*.

Uno de los mayores desafíos se presenta en la localización y el seguimiento de la cámara del dispositivo en relación a los objetos del ambiente. Los equipos cuentan con sensores incluyendo GPS (precisión en metros), acelerómetros, giroscopios y brújulas internas (precisión en milímetros) que permiten conocer la posición del aparato en todo momento. Dependiendo del tipo de aplicación, si es para uso bajo techo o al aire libre, se recurre al uso de estos sensores, por ejemplo: una aplicación de Realidad Aumentada que calcule una ruta (utilizando el GPS del dispositivo) entre las calles de la ciudad, y muestre una flecha en las esquinas en las que hay que girar.

La Realidad Aumentada mediante dispositivos móviles cuenta con dos tipos de interacción, por medio del mismo dispositivo o una interacción tangible. La primera hace referencia a los movimientos que se pueden hacer a través del dispositivo: acercar, alejar, tomar un punto de vista distinto, navegar sobre la imagen, etc. Mientras que la segunda interacción se refiere a las acciones que se pueden realizar sobre los objetos en el mundo real: mover los marcadores de posición, oclusión que podría causar que los objetos virtuales aparezcan y desaparezcan de la pantalla.

Actualmente el desarrollo de aplicaciones enfocadas a tal fin, es decir, la Realidad Aumentada en dispositivos móviles, ha ido creciendo pues hay una gran variedad de campos donde puede aplicarse. Una de las áreas principales donde se han implementado estas aplicaciones es en la publicidad: anuncios en revistas, en postes de luz, muros, que pueden ser escaneados con el celular o tableta y que muestran información adicional a aquella impresa, incluso videos (**Figura 10**).


***Figura 10**. Anuncio en el periódico del Zoológico de Wellington, Nueva Zelanda[9]. Se despliega un modelo 3D del animal correspondiente al marcador.*

[9] http://theinspirationroom.com/daily/2007/augmented-reality-at-wellington-zoo/

#### **1.2.2.3** *Realidad Aumentada mediante cascos (HMDs)*

Una tercera modalidad de la Realidad aumentada es el uso de dispositivos montados sobre la cabeza del usuario, o como parte de un casco, mientras este se encarga de mostrar las imágenes del mundo real y el mundo virtual sobre la vista que el usuario tiene a través del mismo. Existen dos tipos de estos cascos o lentes de Realidad Aumentada: ópticos (*optical-see-through*) y de video (*video-see-through*) *(Moser, K. et al., 2015)*.

En los cascos ópticos, el mundo real es visto a través de espejos semitransparentes ubicados frente a los ojos del usuario. Estos espejos también son utilizados para mostrar los elementos generados por computadora, esto es, los elementos virtuales se reflejan a la vista del usuario complementando la vista del mundo real. Actualmente no todos los dispositivos de este tipo utilizan espejos, también se utilizan otros materiales, como cristal o acrílico.

En el caso de los cascos o lentes de video, se percibe el mundo real por medio de imágenes obtenidas por cámaras externas. La Realidad Aumentada se genera por medio de cálculos computacionales, mostrándole al usuario una sola imagen resultante sobre una pantalla digital ubicada en la cara interna del casco, a nivel de los ojos.

***Figura 11***. *Casco óptico Google Glass (izquierda) y casco de video AR-Rift*[10]

[10] AR-Rift, fuente: https://www.youtube.com/watch?v=Bc_TCLoH2CA

En cuanto a funcionamiento, ambos tipos de casco cumplen con el objetivo de la Realidad Aumentada: combinar imágenes del mundo real y el virtual; sin embargo, sí existen distinciones entre ellos. Mientras que los cascos ópticos no son tan invasivos, bloqueando por completo la vista del usuario, como lo son los de video, estos últimos consiguen un mejor resultado al tener una mayor capacidad para mezclar las dos imágenes obtenidas y así mejorar la inmersión *(Rolland, J. P. y Fuchs, H., 2000)*.

### **1.2.3** *Despliegue e interacción.*

Dependiendo de los objetivos de cada aplicación, el resultado deseado al combinar las imágenes obtenidas del mundo real y el mundo virtual puede variar. Si se requiere resaltar los objetos virtuales, la tarea de integración es mucho más sencilla, pues no requieren un procesamiento avanzado para colocarlos en la escena del mundo real. Sin embargo, si lo que se busca es generar un entorno en donde los objetos generados artificialmente se mezclen por completo con la escena real, el trabajo de procesamiento es mucho mayor.

La escena virtual, generalmente se despliega asumiendo que la cámara es un modelo perfecto de la Cámara de *Pinhole*, pero esto no es así. Las cámaras utilizadas en estos sistemas añaden distorsión e imprimen sus características propias en las imágenes obtenidas, por lo que para combinar estas escenas existen dos opciones: remover los desperfectos de la imagen propios de la cámara, o crear estas imperfecciones artificialmente sobre la imagen generada *(Klein, G. y Murray, D. W., 2010)*. La segunda opción es la más utilizada ya que es más factible mimetizar la escena generada, regulando las características que la hacen destacarse sobre la escena real, como el contraste, el matiz y la intensidad, a diferencia de los recursos utilizados para generar una escena realista en su totalidad.

Existen distintas formas de interactuar con el ambiente virtual generado, desde cambiar nuestra posición hasta utilizar gestos con nuestras manos que pueden ser reconocidos como instrucciones.

La utilización de dispositivos de seguimiento permite el desarrollo e implementación de técnicas de reconocimiento de gestos, en especial de las manos, para realizar las acciones requeridas. El avance en la tecnología permite que estos gestos sean cada vez más intuitivos y reconocidos por una mayor cantidad de dispositivos. Podemos utilizarlos de una manera táctil (sobre una superficie sensible) o en el aire (observados a través de cámaras) *(Billinghurst, M. et al., 2014)*. Algunos ejemplos de estos gestos son los que utilizamos día con día en nuestros teléfonos y tabletas: acercamientos al "pellizcar" la pantalla, desplazarnos deslizando el dedo en la misma, etc. Estos mismos ejemplos pueden ser utilizados y reconocidos por cámaras, y al ser intuitivos, el usuario se acostumbra con facilidad.

El desarrollo de nuevos HMDs amplía el campo de interacción de cualquier sistema, no sólo de Realidad Aumentada, pues han permitido una mayor interacción al no limitar al usuario a un espacio reducido, como una pantalla táctil, teclado de computadora, o cualquier dispositivo físico *(Kolsch, M. et al., 2006)*. Utilizando distintas modalidades de vista (por ejemplo: simular visión de Rayos X), se le da al usuario la oportunidad de interactuar con objetos que no están dentro de su campo de visión actual.

### **1.2.4** *Aplicaciones*

El desarrollo de aplicaciones de Realidad Aumentada ha tenido un incremento significativo en la última década, esto debido al desarrollo de la tecnología que brinda un mayor acceso a estas áreas. Existen desde aplicaciones simples (lectores de códigos bidimensionales que despliegan información) hasta aplicaciones complejas (simuladores quirúrgicos).

Sin embargo, poco se ha tomado en cuenta al momento de crear estas aplicaciones *(Sabelman, E. E. y Lam, R., 2015)*, pues este desarrollo no considera si se está dando un manejo adecuado de la Realidad Aumentada, es decir, si su aplicación es apta para utilizar esta tecnología. El despliegue de información adicional puede ser un distractor para el usuario, y hasta podría llegar a ser peligroso (por ejemplo: al manejar un auto).

Existen distintos factores que han afectado la creación de nuevas aplicaciones de Realidad Aumentada, como lo es la expectativa del usuario y cómo la tecnología a veces no logra satisfacerla. Gracias a la experiencia previa de los usuarios, así como a los medios de comunicación, se plantean conceptos que aún no pueden ser implementados o se encuentran en desarrollo *(Livingston, M. A., 2005 y MacIntyre, B., et al., 2013)*.

# Capítulo II. HARDWARE Y SOFTWARE EN EL SISTEMA DE REALIDAD AUMENTADA

La integración del sistema de Realidad Aumentada se llevó a cabo, como menciona el objetivo, interconectando los distintos elementos que forman un sistema comercial. Para la integración de este sistema se recurrió a distintos dispositivos que se encuentran ya en el mercado.

A continuación se encuentra una descripción más detallada de estos componentes, el funcionamiento de cada uno de manera independiente, así como las versiones que fueron utilizadas y los procesos de calibración utilizados para algunos de ellos.

## **2.1** *Casco de Realidad Virtual: Oculus Rift*

El Oculus Rift es un casco de Realidad Virtual desarrollado por la empresa Oculus VR. Inicialmente fue propuesto en un campaña recaudadora de fondos en donde recaudó 2.5 millones de dólares para su desarrollo inicial.

El casco utiliza sensores de posición y un sistema de lentes ópticos para dar la ilusión al usuario de que realmente se encuentra inmerso en el ambiente virtual. Dos imágenes similares pero con un ligero cambio en la perspectiva, cumpliendo así con los principios de la estereovisión, son desplegadas sobre una pantalla ubicada muy cerca de los ojos del usuario. Estas imágenes son enviadas hacia cada uno de los ojos a través del sistema de lentes cubriendo por completo el campo de visión (FOV por sus siglas en inglés) del usuario, creando la ilusión visual de encontrarse sumergido en el mundo virtual.

Haciendo uso del sistema de posicionamiento se tiene interacción con la cámara del mundo virtual, siguiendo los movimientos que realiza el usuario con la cabeza y replicándolos por medio de transformaciones geométricas. Esto incrementa la ilusión de inmersión, lo que brinda realismo a la experiencia desde un punto de vista visual.

### **2.1.1** *Versiones del casco de Realidad Virtual Oculus Rift*

A la fecha existen dos versiones comerciales, ambas dirigidas a los desarrolladores. Actualmente la compañía está trabajando sobre una tercera versión ahora dirigida directamente a los consumidores, siendo el primer casco de Realidad Virtual con este enfoque. La siguiente versión estará disponible a partir del año 2016.

El dispositivo utilizado para este proyecto corresponde a la primera versión dirigida a desarrolladores, conocida como kit de desarrollo 1 (DK1 por sus siglas en inglés). Al presente esta versión ya no se encuentra disponible, pues fue retirada del mercado al presentar el segundo kit de desarrollo en marzo de 2014.

### **2.1.2** *Principales componentes*

El equipo DK1 consta principalmente del **casco**, el cual utiliza relleno de espuma en los bordes destinados a estar en contacto con la piel de la cara así como correas ajustables que ayudan a sostener el casco en su lugar. Además se pueden encontrar pequeñas ruedas de ajuste, lo que permite alejar o acercar la pantalla al rostro del usuario (**Figura 12**).


***Figura 12****. Casco de Realidad Virtual Oculus Rift. Frente (izquierda) y perfil (derecha).*

El dispositivo incorpora una **pantalla** LCD con una resolución de 1280x800 y una relación de aspecto de 16:10, así como un sistema de seguimiento de movimiento que considera la aceleración, ángulo de rotación y magnitud y dirección del campo magnético. La pantalla se divide entre los ojos obteniendo una resolución parcial de 640x800 para cada uno. Tiene un campo de visión mayor a los 90 grados en horizontal y 110 grados en diagonal (**Figura 13**).

***Figura 13.*** *Pantalla del casco Oculus Rift, vista sin los lentes.*

La pantalla es considerada el elemento más importante del casco, pues gracias a las características de la misma permiten que el usuario logre la inmersión necesaria para sentirse parte del mundo virtual. Esta función de inmersión trabaja en conjunto con los pares de lentes (descritos más adelante), al lograr ampliar el campo de visión.

Cada una de las mitades de la pantalla representa el campo de visión de los ojos izquierdo y derecho respectivamente (**Figura 14**). Las imágenes desplegadas en ellas nunca se cruzan, sin embargo, deben cumplir con los principios de estereovisión al presentar tomas de una misma escena desde distintos puntos de vista, los cuales equivalen al punto de vista de cada ojo, y así permitir al cerebro interpretar la profundidad en la escena.

El casco Oculus Rift logra un mayor campo de visión gracias a la combinación de la pantalla del dispositivo con un juego de lentes, dando la impresión de que la pantalla es más grande de lo que en realidad es, al permitir que los rayos de luz provenientes de los puntos más alejados de la pantalla lleguen a los ojos del usuario, cubriendo por completo su visión para así poder experimentar una mayor inmersión en el ambiente que se le está mostrando.

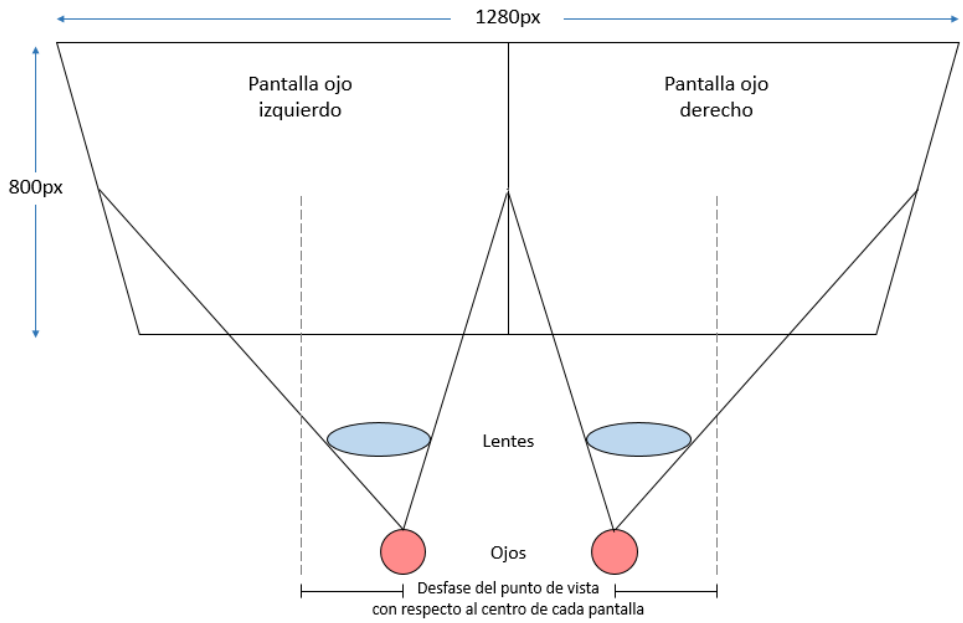


***Figura 14**. Relación entre el punto de vista del usuario y el despliegue en la pantalla del casco Oculus Rift*

## **2.2** Leap Motion

El controlador Leap Motion es un sensor de movimiento desarrollado con el objetivo de reemplazar al teclado y mouse de una computadora al poder sustituir su funcionamiento (aún en desarrollo). El dispositivo realiza un seguimiento de la forma y posicionamiento de las manos dentro de un espacio tridimensional con una precisión de 1.0 mm

El dispositivo se coloca sobre una superficie plana y éste toma el espacio sobre él como una zona interactiva en donde el usuario puede utilizar sus manos para controlar diferentes funciones de la computadora. El campo de visión del sensor está limitado a una semiesfera de 60cm de radio (**Figura 15**). Esta limitación se da por la propagación de la luz LED en el espacio, ya que se vuelve mucho más complicado inferir la posición de las manos en un espacio tridimensional más allá de una cierta distancia.

La intensidad de la luz de los LEDs se define por el máximo de corriente que se pueda transmitir por medio de la conexión USB.

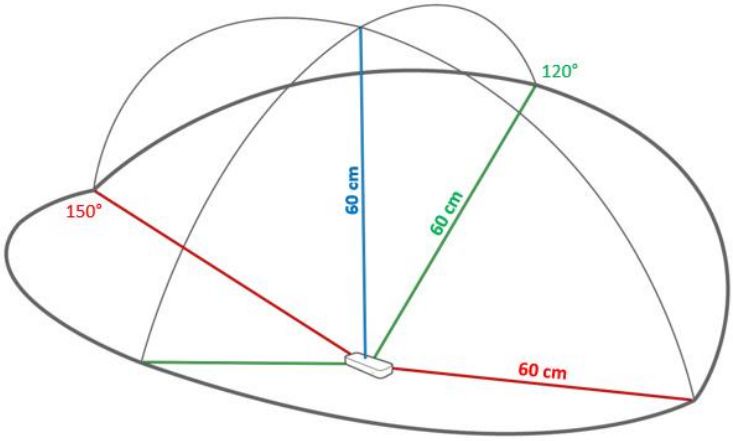


***Figura 15***. *Espacio de interacción del sensor Leap Motion.*

Leap Motion es compatible con los diferentes sistemas operativos y una variedad de aplicaciones diseñadas específicamente para funcionar con el dispositivo. Algunas aplicaciones conocidas mundialmente, *Google Maps*, *Cut the rope*, por mencionar unas cuantas, han incluido el uso del controlador en algunas de sus funciones.

El sensor Leap Motion cuenta con un par de cámaras. Estas son un componente fundamental del equipo pues son las encargadas de capturar las imágenes que constituyen el funcionamiento restante del sistema. Cada cámara cuenta con un sensor monocromático sensible a la luz infrarroja. La velocidad de este sensor depende del rendimiento de la computadora a la que se haya conectado. El controlador (*driver*) del dispositivo toma la información obtenida por el sensor, la almacena en su propia memoria y realiza los ajustes necesarios en la resolución.

La información almacenada toma la forma de imágenes estereoscópicas en escala de grises del espectro de luz infrarroja a una distancia cercana. Normalmente los únicos objetos que serán visibles en las imágenes son los directamente iluminados por el dispositivo, sin embargo, focos incandescentes y la luz del Sol pueden iluminar la escena y por ende la imagen resultante.

Los LEDs son los que se encargan de la iluminación del domo de cobertura por medio de una luz infrarroja. Éstos varían su consumo eléctrico, y por lo tanto la iluminación que otorgan dependiendo de qué tan iluminado esté el entorno en donde se está utilizando el controlador. Los cambios en la intensidad de la luz, y por ende de los LEDs, permiten que se obtenga una misma resolución de imagen en todos los escenarios.

Ya que la información es transferida a la computadora el Servicio de Leap Motion (*Leap Motion Service*) es el encargado de analizar las imágenes. Después de identificar los objetos del fondo y la luz ambiental, las imágenes son procesadas para reconstruir una representación 3D de lo que el dispositivo puede ver. El objetivo principal de lo anterior es encontrar la posición en el mundo real del objeto a seguir, las manos en nuestro caso.

A continuación se evalúan las escenas 3D para identificar información relevante para el sistema de seguimiento, como son los dedos y algunas herramientas. Los algoritmos de seguimiento interpretan la información 3D y deducen la posición de objetos ocluidos. Se aplican distintas técnicas de filtrado para asegurarse que existe continuidad entre las imágenes. Para finalizar, el Servicio de Leap Motion entrega los resultados, expresados como una serie de cuadros que contienen la información de seguimiento, como parte del protocolo de comunicación.

### **2.2.1** *Características*

El dispositivo Leap Motion tiene unas dimensiones de 75mm de largo, 25mm de ancho y 11mm de profundidad, lo que resulta muy reducido al compararlo con otros controladores similares.

Viéndolo desde una perspectiva totalmente física, el controlador Leap Motion es bastante simple. La pieza principal del dispositivo consiste de dos cámaras y tres LEDs infrarrojos. Gracias a que los lentes de las cámaras tienen un ángulo bastante amplio es que el espacio de interacción del dispositivo es de aproximadamente 0.226 metros cúbicos, los cuales son considerados dentro de la intersección de los campos de visión de las cámaras integradas en el sensor.

El diseño físico del dispositivo separa los LEDs con pequeñas barreras plásticas, lo que permite una iluminación uniforme sobre toda el área, además de evitar la saturación de los sensores al no recibir la luz de una forma directa.

## **2.3** *Cámaras web*

La cámara web Logitech c920 HD Pro (**Figura 16**) obtiene imágenes y video de alta definición. Brinda una resolución de video de 1080p, incluso cuando corre a 30 cuadros por segundo. Otro tipo de cámaras que ofrecen esta misma resolución no consiguen esta misma velocidad, lo que ocasiona que se salten cuadros y se pierda información.

La resolución que puede obtenerse en las imágenes capturadas por esta cámara llega hasta los 15 mega pixeles, con una calidad de imagen lo bastante buena para que la imagen se muestre claramente en la pantalla.

Estas cámaras son fáciles de utilizar gracias a su capacidad de autoenfoque. Además, cuentan con la característica de seguimiento de rostro, por lo que pueden enfocar a pesar de que la persona se encuentre en movimiento (frente a la cámara). Esta característica las hace compatibles con software de reconocimiento facial, como FastAccess[11] un programa que reemplaza el uso de contraseñas.


***Figura 16****. Cámara Logitech HD Pro c920.*

[11] http://www.sensiblevision.com/home.aspx

Dentro de la elaboración del sistema de Realidad Aumentada, las cámaras son utilizadas para la obtención de la escena correspondiente al mundo real, por lo que la buena resolución es un requisito indispensable para mejorar los resultados al permitir una mejor definición en el resultado final.

## **2.4** *Software*

Para el desarrollo del programa (código) del sistema, se recurrió al apoyo de bibliotecas existentes que contaban con las funciones requeridas por el sistema. A continuación se describe cada una de ellas, así como los elementos que se utilizaron de las mismas.

### **2.4.1** *Biblioteca Boost 1.57.0*

La biblioteca *Boost*[12] representa un conjunto de bibliotecas de software libre creadas con el objetivo de complementar las capacidades del lenguaje de programación C++, con el cual está desarrollado el proyecto. Esta biblioteca puede ser utilizada en cualquier tipo de proyectos, sean o no comerciales.

La serie de bibliotecas pertenecientes a *Boost* cubren todas las necesidades del desarrollador, pues abarcan desde desarrollos de propósito general hasta abstracciones del sistema operativo. Más adelante se mencionan algunos ejemplos. Se hace uso de plantillas permitiendo así una mayor flexibilidad al momento de implementar la biblioteca.

Algunas de las bibliotecas más utilizadas son las de álgebra lineal, generación de números pseudoaleatorios, multihilos, procesamiento de imágenes, expresiones regulares y pruebas unitarias, llegando a conformar un total aproximado de 80 bibliotecas individuales, que a su vez no tienen que ser construidas antes de su uso.

De todo el conjunto, nuestro proyecto de Realidad Aumentada toma como clases dos de estas bibliotecas:

[12] http://www.boost.org/

a) **Boost::circular_buffer**: este objeto se comporta como cualquier *buffer*, refiriéndose a un área de memoria utilizada para almacenar información, sin embargo, tiene una estructura circular por lo que también es conocido como "*buffer* de anillo" o "*buffer* cíclico" (**Figura 17**).

El *buffer* circular está diseñado para brindar una capacidad de almacenamiento fija por lo que al exceder esta capacidad los nuevos elementos toman el lugar de los que se habían almacenado originalmente. Dependiendo de la operación utilizada, los nuevos elementos se insertan al principio o al final del anillo.

Tiene la capacidad de reservar la memoria asignada en sólo algunos casos: al momento de la creación del objeto, al realizar un ajuste de tamaño explícito, o al realizar una operación de asignación.

Soporta iteradores, los que permiten realizar un acceso aleatorio a los elementos almacenados por medio de un índice de posición. Además, la inserción y eliminación de elementos se realiza en tiempo constante, siempre y cuando la operación se haga al inicio o al final del anillo.

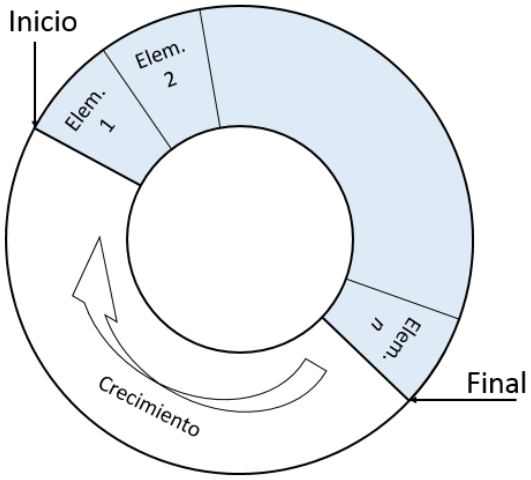


***Figura 17**. Estructura del buffer circular.*

b) **Boost::thread**: esta biblioteca permite el uso de múltiples hilos de ejecución con la posibilidad de compartir datos entre ellos, siendo un código de C++ portable lo que permite su ejecución en las diferentes plataformas de desarrollo (**Figura 18**).

Brinda al programador una diversidad de clases y funciones que permiten el manejo de los hilos, así como variables de sincronización y seguridad al compartir la información, o dando la posibilidad de crear copias de la misma dándole una a cada hilo.

Cada uno de los objetos *Boost* representa un solo hilo de ejecución. Su manejo y funcionamiento no difieren de los hilos que utilizan las interfaces específicas del sistema operativo. Lo anterior es posible pues la biblioteca sustrae todo el código que sea específico para alguna plataforma, conservando la esencia del mismo y permitiendo tener un código portable.

Los hilos creados con la biblioteca *Boost* pueden establecerse con un estado especial, llamado "*not-a-thread*" (no-hilo), permaneciendo inactivos al no recibir una función específica para ejecutar.

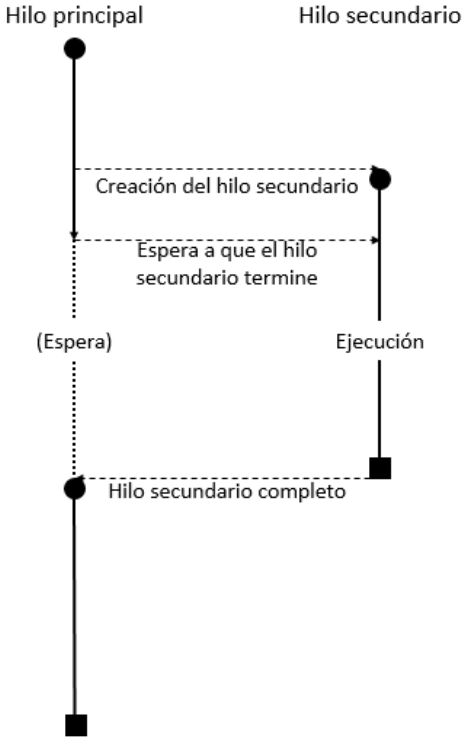


***Figura 18**. Funcionamiento de un sistema con múltiples hilos de ejecución.*

### **2.4.2** *Biblioteca GLFW 3.1.1*

Antes de introducir la biblioteca, primero debemos definir *OpenGL* [13] como una especificación estándar de una interfaz de programación de aplicaciones (API) multilenguaje y multiplataforma para desarrollar aplicaciones que produzcan gráficos en dos y tres dimensiones.

La biblioteca *GLFW* [14] está diseñada como una herramienta ligera para trabajar con OpenGL. Tiene la capacidad de crear y manejar ventanas, así como recibir las señales de entrada provenientes del teclado, mouse o joystick.

OpenGL no proporciona, por sí solo, los mecanismos necesarios para encargarse del manejo de la interfaz incluyendo los métodos de entrada. Existen diferentes bibliotecas que apoyan la administración de este segmento entre las que se encuentran GLUT y SDL, ambas implementaciones de código abierto las cuales presentan ciertas desventajas como el enfoque con el que fueron diseñadas o su inestabilidad en ciertos casos. A pesar de que no es la única biblioteca disponible, *GLFW* está diseñada para ser una biblioteca ligera, con la capacidad de sustituir a las ya existentes, que permita gestionar la interfaz e interacción del usuario.

Para la parte gráfica, habilita la creación y el manejo de ventanas de escritorio por medio de OpenGL, así como reconocer, enumerar y administrar distintos monitores o pantallas de despliegue junto con sus modos de video.

*GLFW* ofrece compatibilidad a todas aquellas plataformas que ejecuten aplicaciones con OpenGL, pues brinda una relación sencilla entre estas dos bibliotecas permitiendo que la comunicación fluya, lo que da la posibilidad al desarrollador de tener control de funciones específicas de la plataforma sobre la que está trabajando y al mismo tiempo seguir utilizando un código general que podría correr en cualquiera de ellas.

[13] https://www.opengl.org/
[14] http://www.glfw.org/

### **2.4.3** *Biblioteca OpenCV 3.0*

*OpenCV* [15] es una biblioteca de software libre que contiene funciones principalmente enfocadas en tratar problemas de cómputo visual en tiempo real. Es una biblioteca multiplataforma lo que permite desarrollar aplicaciones para cualquier ambiente utilizando las versiones equivalentes.

Cuenta con un gran número de funciones que, si bien todas están enfocadas a la rama de visión por computadora, cubren una amplia variedad de áreas. Dentro de estas funciones podemos encontrar algunos ejemplos de las más utilizadas, como reconocimiento de objetos, reconocimiento facial y de gestos, interacción humano-computadora, segmentación, estereovisión, seguimiento, calibración, realidad aumentada, entre otras.

*OpenCV* está desarrollada en C y C++, aprovechando así las capacidades de procesamiento que brindan estos lenguajes. Además, al ser creado originalmente por Intel, puede tener acceso a las herramientas de rendimiento para los procesadores correspondientes.

### **2.4.4** *Biblioteca LibOVR*

El kit de desarrollo de Oculus Rift fue diseñado para integrarse fácilmente a cualquier proyecto. La biblioteca *LibOVR* [16] permite controlar la información que se obtiene del casco de Realidad Virtual Oculus Rift, como posición y giros, permitiendo así una interacción fluida con el ambiente.

La pantalla del dispositivo no se controla por medio de esta biblioteca pues se considera como un monitor, por lo que se utiliza *GLFW* para acceder al mismo, así como para ajustar la resolución y opciones de pantalla completa.

Existen tres etapas principales de implementación de esta biblioteca: configuración, ciclo de juego y cierre. Para poder hacer uso de la biblioteca *LibOVR* es necesario seguir una serie de instrucciones en cierto orden para que todos los componentes estén activos e inicializados al momento que sean requeridos. Las acciones a realizar son las siguientes:

a) Inicializar la biblioteca *LibOVR*.

[15] http://opencv.org/
[16] https://developer.oculus.com/downloads/

b) Crear un objeto "*hmd*" que representa al casco. Este objeto sirve para verificar que exista una conexión correcta entre el programa y el casco real, además es por medio de este objeto que se tiene acceso a la información del mismo, como posición, pantalla, etc.
c) En caso de requerir la posición del dispositivo es necesario configurar el sistema de seguimiento. Existen instrucciones que permiten obtener esta información de una manera directa y fácil de utilizar en una aplicación.
d) Inicializar el proceso de despliegue sobre la pantalla del dispositivo. Esto permite que las imágenes generadas sean transmitidas de manera adecuada a la pantalla del casco. Es en esta etapa en donde se indican los parámetros de resolución y campo visual, según sean requeridos por la aplicación
e) Ajustar el tiempo de despliegue de los cuadros (*frame rate*)
f) Crear una configuración adecuada para una buena interacción con las imágenes que se estarán desplegando en el dispositivo.
g) Destruir el objeto creado inicialmente cuando se termine de utilizar y liberar la memoria correspondiente.

### **2.4.5** *Biblioteca LM*

El sistema de Leap Motion, mediante la biblioteca *LM*[17], reconoce y realiza seguimiento de manos, dedos o herramientas semejantes a éstos. El dispositivo funciona con buena precisión al momento de localizar los puntos destacados, así como la continuidad de los cuadros en las imágenes.

El controlador utiliza sensores ópticos y luz infrarroja para realizar la detección de manos dentro de un intervalo establecido. La detección funciona mejor cuando el sensor tiene una vista clara y con un alto contraste entre los objetos a detectar, en este caso las manos, y el fondo, destacando así la silueta. La biblioteca *LM* combina la segmentación de imágenes con un modelo interno de la mano humana para hacer un mejor reconocimiento de la misma.

[17] https://developer.leapmotion.com/libraries

El sistema coordenado que se utiliza representa al sistema denominado de la mano derecha, siendo el origen el punto medio de la superficie del sensor. Los ejes X y Z se encuentran en el plano horizontal, donde el eje X corre a lo largo del dispositivo y el eje Y de manera vertical, con valores positivos alejándose de la superficie (**Figura 19**).

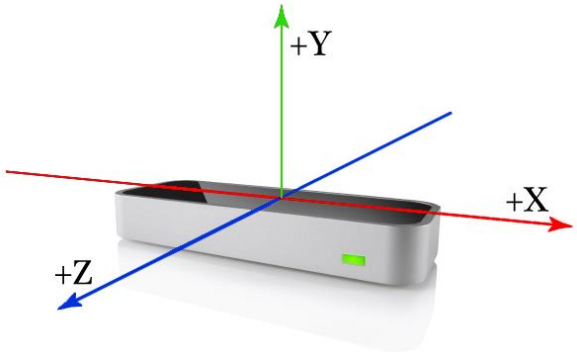


***Figura 19***. *Sistema coordenado del sensor Leap Motion.*

Para el seguimiento de las manos, dedos y herramientas, la biblioteca almacena la información de la posición de los mismos como un conjunto de datos en cada instante, este conjunto representa un cuadro (*Frame*) del sensor. Cada uno de estos objetos contiene una lista de todos los componentes de los que se registró la información, principalmente de posición, así como si se reconoció algún gesto en la escena.

El modelo interno de la mano que es utilizado por esta biblioteca está provisto de un seguimiento predictivo a pesar de haber oclusiones de alguna de las partes de la misma. Siempre se conocen las posiciones de los cinco dedos, teniendo un mejor desempeño cuando todos los elementos son visibles en las imágenes captadas por el sensor. Además de la estructura de la mano, se almacenan otras características que hacen referencia a ella, como el brazo al que iría unida.

Cada dedo en el modelo contiene información propia. Si el dedo o parte de él no es visible, su posición actual se estima a partir de observaciones que se realizan en los estados anteriores así como en la relación anatómica con el modelo de la mano. Cada uno es identificado por su nombre en inglés: "*thumb"* (pulgar), "*index*" (índice), "*middle*" (medio), "*ring*" (anular) y "*pinky*" (meñique).

Además de las estructuras anatómicas, la biblioteca también proporciona funciones de reconocimiento de gestos básicos, como trazar una línea en el espacio, realizar un movimiento circular en el aire, subir y bajar un dedo, etc. Así como también posiciones establecidas de la mano.

# Capítulo III. IMPLEMENTACIÓN DEL SISTEMA DE REALIDAD AUMENTADA

El desarrollo de este proyecto consta de dos etapas: la construcción física del dispositivo y la programación (código) del sistema.

El capítulo anterior describió el equipo utilizado para la construcción y las funciones individuales de cada uno de sus elementos. Ahora se presenta la interacción de los mismos para conformar el sistema de Realidad Aumentada y el papel que cumple cada uno en el funcionamiento del todo.

En la etapa de programación se detalla cada una de las piezas que conforman el software del sistema. Se explica cada uno de los elementos utilizados, obtención de imágenes, manejo de modelos, coordinación de los hilos, trabajo en texturas, *shaders* y despliegue de imágenes finales sobre la pantalla del casco Oculus Rift.

## **3.1** *Diseño del sistema*

Hoy en día se pueden encontrar distintos dispositivos de Realidad Aumentada en el mercado. Desde los que están enfocados al entretenimiento hasta los que son desarrollados con fines de investigación.

Para cumplir con el objetivo de crear un sistema de Realidad Aumentada, se propuso la implementación de uno partiendo de los elementos que lo componen y coordinando su funcionamiento. El sistema se basa en la integración de un casco de Realidad Virtual, un par de cámaras web y un sensor de movimiento (**Figura 20**).

Los dispositivos utilizados (casco, cámaras y sensor) para la construcción física del sistema, se eligieron por su accesibilidad y simpleza al momento de integrarlos, así como soporte de sus bibliotecas.

Todos estos elementos son autónomos, sin embargo, se utiliza un programa para coordinar sus funciones. La implementación de este programa considera el concepto de sistemas multihilos o *multi-threading*, es decir, cada elemento es ejecutado en un hilo de procesamiento independiente, agilizando el funcionamiento del mismo.


***Figura 20**. Sistema de Realidad Aumentada.*

## **3.2** *Integración de los elementos del sistema*

La construcción del sistema se basa en la recreación de dispositivos comerciales existentes tomando como base el casco de Realidad Virtual Oculus Rift para el despliegue y añadiendo las cámaras y el sensor sobre el mismo.

### **3.2.1** *Colocación de las cámaras*

Las cámaras web son colocadas a los costados del casco en posición vertical, es decir, con una rotación de -90° para la cámara correspondiente al ojo izquierdo y 90° para la cámara correspondiente al ojo derecho, las rotaciones se consideran tomando el sistema desde una vista frontal (**Figura 21**). Estas cámaras representan la visión del usuario y las imágenes obtenidas por ellas son desplegadas sobre la pantalla del Oculus Rift dando la sensación de estar viendo a través del mismo.


***Figura 21**. Colocación de las cámaras en el sistema de Realidad Aumentada.*

Las cámaras son colocadas con cierta separación horizontal, lo que permite obtener imágenes de la misma escena con diferentes perspectivas o puntos de vista. En la **Figura 22** se muestra el campo de visión de cada una de las cámaras, teniendo un área de intersección en el centro. El área común de los campos de visión se presenta a la derecha de la imagen resultante izquierda, y a la izquierda de la imagen resultante derecha.

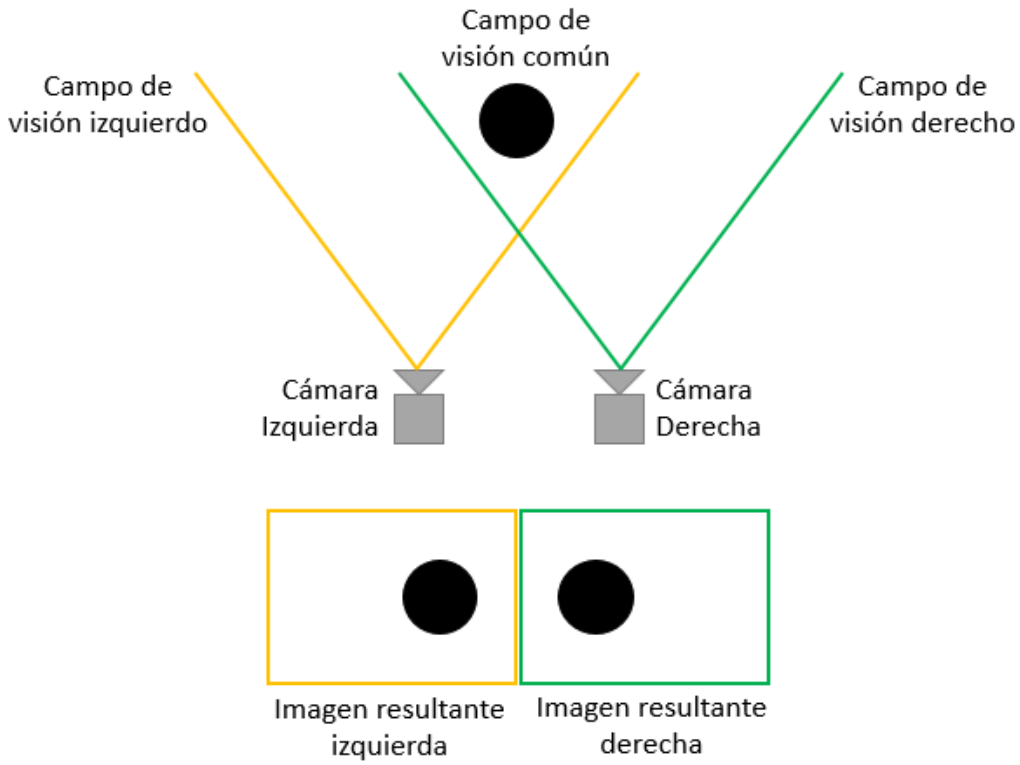


***Figura 22**. Campos de visión de un sistema de cámaras estéreo.*

Cada una de estas imágenes resultantes corresponde a una mitad en la pantalla del casco de Realidad Virtual de la siguiente manera: la imagen obtenida por la cámara que corresponde al ojo izquierdo se muestra en la mitad izquierda de la pantalla, lo mismo ocurre con la imagen, cámara y mitad de pantalla derechas (**Figura 23**).

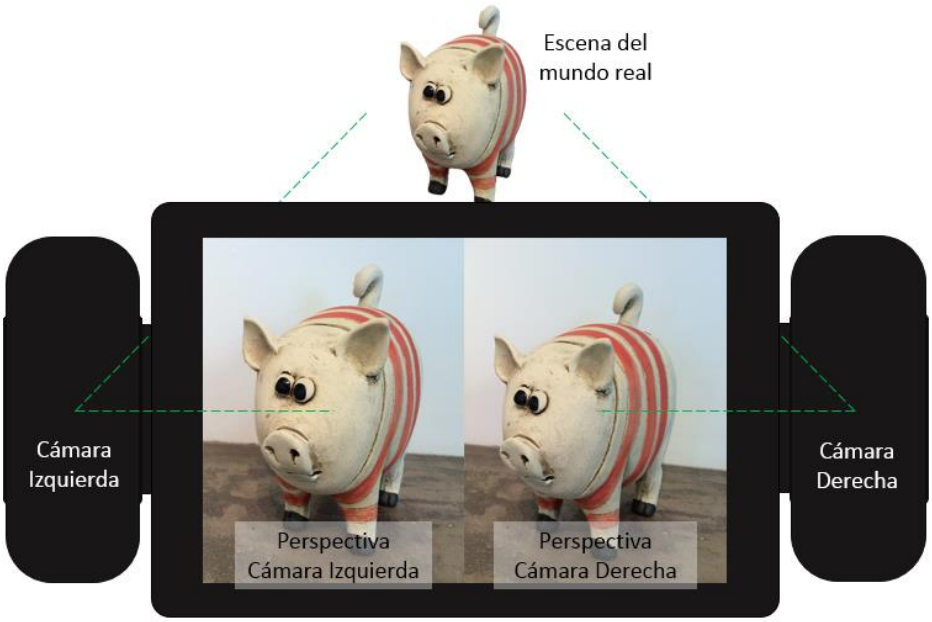


***Figura 23***. *Perspectivas obtenidas por las cámaras desplegadas en la mitad de la pantalla correspondiente*[18].

Esta forma de disposición de las imágenes sobre la pantalla del casco permite que cada uno de los ojos del usuario sólo vea la mitad que le corresponde, permitiendo al cerebro realizar el proceso de estereovisión y darle la profundidad correspondiente a la escena.

### **2.2.2** *Calibración de las cámaras web en estéreo*

Como se describió en el capítulo de Antecedentes, las cámaras reales no presentan el modelo ideal de la cámara de *Pinhole* pues los lentes introducen distorsión, y pueden llegar a generar ruido a las imágenes. Por lo anterior fue necesario realizar un proceso de calibración, utilizado para rectificar las imágenes obtenidas, así como para conocer la ubicación de las cámaras con respecto una de la otra.

[18] Imágenes obtenidas de la base de datos de imágenes estéreo del Laboratorio de *Image Vision Systems* de la Universidad de Auckland, Nueva Zelanda. http://www.ivs.auckland.ac.nz/quick_stereo/

La calibración determina los parámetros que nos permiten proyectar un punto en el espacio tridimensional a sus coordenadas equivalentes sobre una imagen 2D. Estos parámetros representan las características internas y externas de la cámara. Las características internas se refieren a aquellas relacionadas con el sistema óptico de la cámara, mientras que las externas nos indican las transformaciones matemáticas requeridas para pasar del marco de referencia del mundo al de la cámara.

El modelo de proyección central de la cámara *pinhole* se presenta en coordenadas homogéneas de la siguiente manera *(Malek, S., et al., 2011)*:

$$\begin{pmatrix} fx \\ fy \\ z \end{pmatrix} = \underbrace{\begin{bmatrix} f & & \\ & f & \\ & & 1 \end{bmatrix}}_{Matriz\ de\ proyección} \underbrace{\begin{bmatrix} 1 & & & 0 \\ & 1 & & 0 \\ & & 1 & 0 \end{bmatrix}}_{Matriz\ de\ transformación} \underbrace{\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}}_{Coordenadas}, \qquad (9)$$

donde $f$ es la distancia focal de la cámara; $x, y$ y $z$ las coordenadas espaciales del punto que se está analizando. Siendo la ecuación (9) la ecuación base del modelo de proyección, al introducir valores específicos se puede obtener lo siguiente:

$$\begin{pmatrix} su \\ sv \\ s \end{pmatrix} = \underbrace{\begin{bmatrix} \alpha_u & \gamma & u_0 & 1 \\ 0 & \alpha_v & v_0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}}_{Matriz\ de\ proyección} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{Matriz\ de\ transformación} \underbrace{\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}}_{Coordenadas}, \qquad (10)$$

siendo $\alpha_u, \alpha_v, \gamma$ los valores distorsión y enfoque, $u_0, v_0$ las coordenadas del centro del plano de proyección, $r_{ij}$ los valores de la matriz de rotación $R$ y $t_i$ los valores del vector de traslación $\overline{t}$.

En el proceso se recurrió al *toolbox*[19] de calibración para MATLAB. Para comenzar este proceso, el primer paso es generar e imprimir un patrón de calibración y pegarlo en una superficie plana. En esta ocasión se utilizó un patrón de tablero de ajedrez de 8x6 cuadros, donde cada cuadro tiene 15mm por lado (**Figura 24 (a)**).

[19] http://www.vision.caltech.edu/bouguetj/calib_doc/

El siguiente paso es adquirir las imágenes. Para lograrlo se montaron las cámaras sobre el casco y se realizó la adquisición de 35 imágenes para cada cámara de manera simultánea (**Figura 24 (b)**), cuidando que el patrón de calibración se encontrara en ambos campos de visión y tratando de cubrirlos por completo. Posteriormente se seleccionaron 25 de las 35 imágenes obtenidas, descartando las borrosas, mal iluminadas, o que fueran muy similares a otras ya seleccionadas.

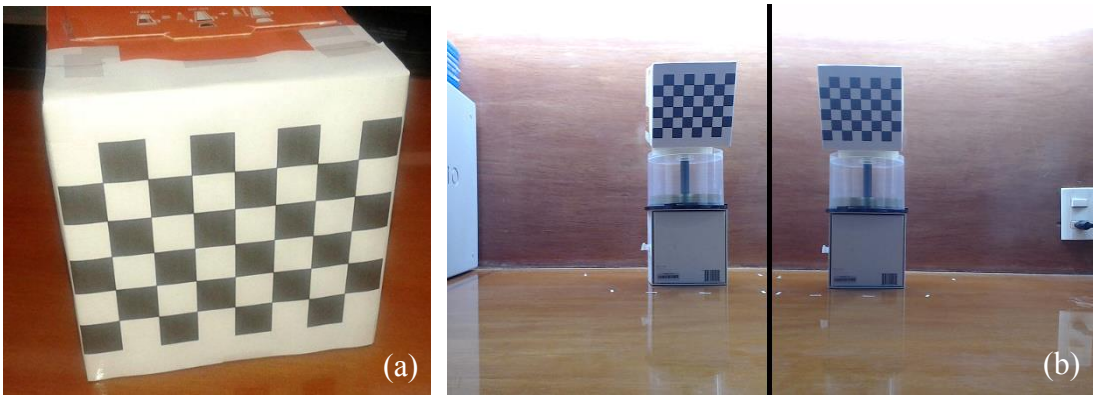


***Figura 24***. *Patrón de calibración (izquierda) y ejemplo de imágenes obtenidas por cada cámara (derecha).*

La calibración se realiza inicialmente de manera individual para cada cámara. Habiendo seleccionado el grupo de imágenes se procede a iniciar el programa cargando la selección a la memoria, para después indicar, por medio del *toolbox*, las esquinas correspondientes al patrón de calibración (**Figura 25**). Estas esquinas se seleccionan en el mismo orden en cada imagen, pues son el marco de referencia para posicionar las cámaras.

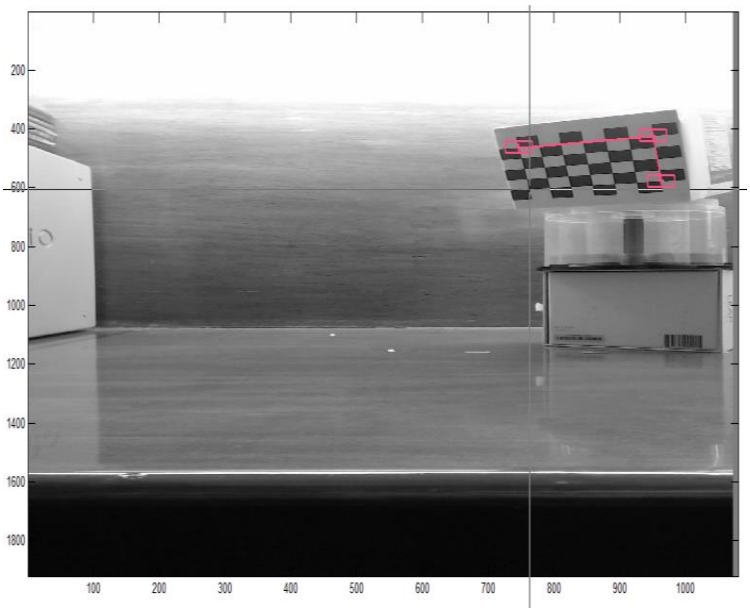


***Figura 25**. Selección de esquinas exteriores del patrón de calibración en una de las imágenes.*

Posteriormente el algoritmo de calibración utiliza un método lineal como aproximación inicial a los parámetros extrínsecos, la distancia focal y el punto principal en la imagen. De la ecuación (10) tenemos:

$$P = \lambda(V^{-1}B^{-1}F)R\bar{t}, \qquad (11)$$

siendo *R* las matrices de rotación y $\bar{t}$ el vector de traslación, *F*, *V* y *B* la descripción de los parámetros intrínsecos:

$$V = \begin{bmatrix} 1 & 0 & -u_0 \\ 0 & 0 & -v_0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad (12)$$

$$B = \begin{bmatrix} 1+k_1 & k_2 & 0 \\ k_2 & 1-k_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad (13)$$

$$F = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (14)$$

de los cuales $u_0, v_0$ son las coordenadas del punto central del plano de proyección, $k_i$ son los coeficientes de distorsión y $f$ la distancia focal.

A continuación se realiza un método iterativo que ajusta el valor de estos parámetros para así calcular el factor de proporción y los coeficientes de distorsión radial y tangencial, descomponiendo la matriz de proyección:

$$\sum (u - u_f)^2 + \sum (v - v_f)^2 \qquad (15)$$

Obteniendo los siguientes resultados para cada una de las cámaras, izquierda y derecha respectivamente:

**Izquierda**

| | |
|---|---|
| Distancia focal: | $fc = [1482.65 \quad 1497.32\,] \pm [116.94 \quad 122.15]$ |
| Punto principal: | $cc = [538.85 \quad 1052.00\,] \pm [117.03 \quad 116.89]$ |
| Distorsión: | $kc = [0.153 \quad -0.154 \quad 0.029 \quad -0.006] \pm [0.15 \quad 0.21 \quad 0.03 \quad 0.01]$ |
| Error de pixeles: | $err = [0.91208 \quad 0.42315]$ |

**Derecha**

| | |
|---|---|
| Distancia focal: | $fc = [1371.30 \quad 1400.16\,] \pm [232.48 \quad 269.93]$ |
| Punto principal: | $cc = [411.20 \quad 1123.39\,] \pm [106.69 \quad 201.78]$ |
| Distorsión: | $kc = [0.428 \quad -0.394 \quad 0.075 \quad -0.001] \pm [0.40 \quad 0.58 \quad 0.10 \quad 0.00]$ |
| Error de pixeles: | $err = [2.18522 \quad 0.61509]$ |

Donde la distancia focal representa la distancia del origen del sistema coordenado de la cámara al plano de proyección, el punto principal es el punto sobre la imagen donde interseca con el eje principal, la distorsión equivale a cada uno de los coeficientes de distorsión y el error en pixeles es el error promedio en todas las imágenes.

Con los datos obtenidos en la primera etapa de calibración se proyectan los puntos de la imagen sobre el marco de referencia del mundo, y obteniendo los valores de dispersión de las coordenadas equivalentes a los puntos retro proyectados comparados con los reales (obtenidos en la etapa de selección de esquinas). Las gráficas de dispersión de la primera etapa se muestran en la **Figura 26**

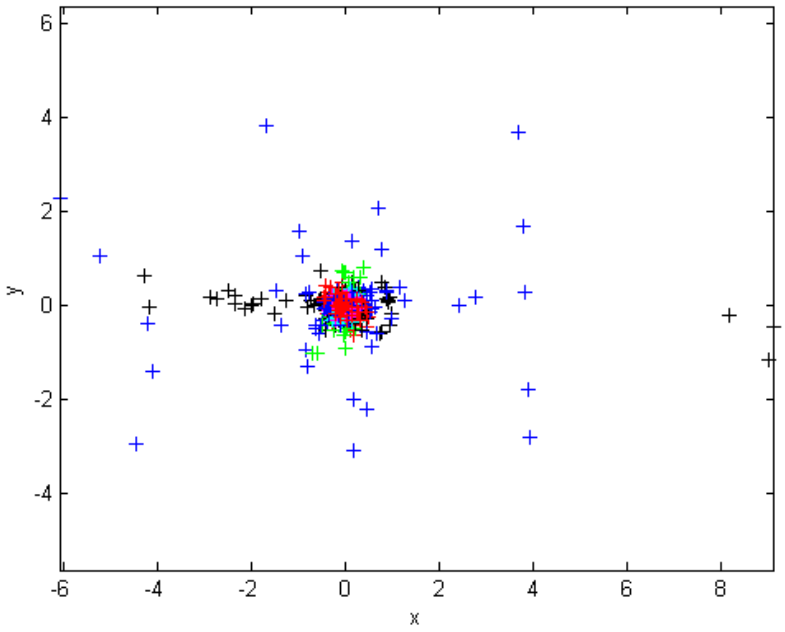


***Figura 26**. Error en pixeles después de hacer la proyección del espacio real a la imagen. Primera ejecución del algoritmo de calibración.*

En las gráficas de error en los pixeles se aprecia que algunas de las imágenes se destacan más que otras. Esto sucede porque al momento de seleccionar las esquinas, es un proceso que se realiza a mano, y dependiendo de la imagen, puede causar un mayor error al momento de localizar la esquina correcta.

Para mejorar nuestros valores de calibración, se hace un ajuste en la selección de las esquinas exteriores (proceso de detección automático) y se ejecuta la calibración una segunda vez, esta ocasión con los resultados siguientes:

**Izquierda**

| | |
|---|---|
| Distancia focal: | $fc = [1357.71 \quad 1365.31\ ] \pm [48.84 \quad 48.65]$ |
| Punto principal: | $cc = [627.76 \quad 974.47\ ] \pm [58.37 \quad 53.04]$ |
| Distorsión: | $kc = [0.091 \quad -0.137 \quad 0.008 \quad 0.008] \pm [0.06 \quad 0.14 \quad 0.01 \quad 0.02]$ |
| Error de pixeles: | $err = [0.30165 \quad 0.40829]$ |

**Derecha**

| | |
|---|---|
| Distancia focal: | $fc = [1326.11 \quad 1342.85\ ] \pm [72.51 \quad 46.49]$ |
| Punto principal: | $cc = [598.41 \quad 1002.93\ ] \pm [86.67 \quad 67.40]$ |
| Distorsión: | $kc = [0.281 \quad -0.327 \quad 0.044 \quad -0.003] \pm [0.18 \quad 0.33 \quad 0.04 \quad 0.01]$ |
| Error de pixeles: | $err = [0.81444 \quad 0.54163]$ |

Se puede observar, que los resultados mejoran, reduciendo el error en los pixeles, así como la diferencia en las distancias focales y el punto principal de cada cámara, que como se dijo en el capítulo anterior, representa el punto en la imagen en donde el eje principal interseca.

Al igual que en la ejecución previa, se realiza una retroproyección para verificar la dispersión de los pixeles (**Figura 27**). Se puede notar que los errores que se encontraban en la ejecución pasada son disminuidos debido al nuevo cálculo de las esquinas.

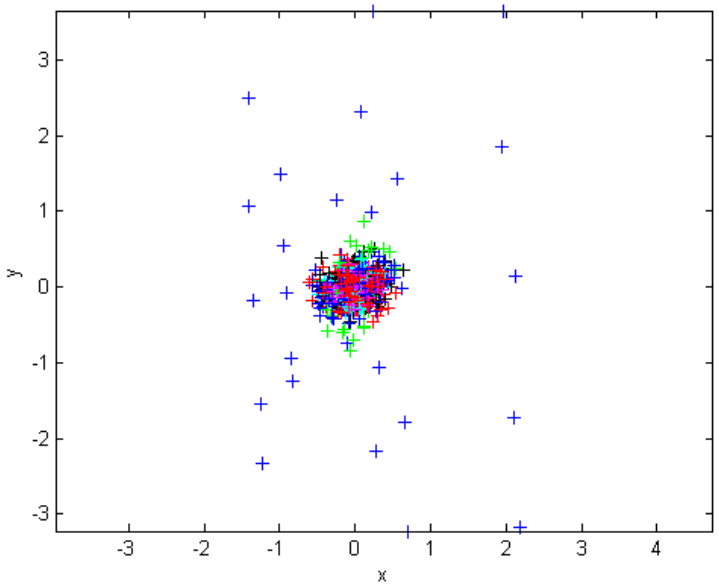


***Figura 27**. Error en pixeles después de la proyección del espacio real a la imagen. Segunda ejecución del algoritmo de calibración.*

Para finalizar esta etapa se hace un despliegue gráfico de la posición de las imágenes con respecto a cada una de las cámaras (**Figura 28**).

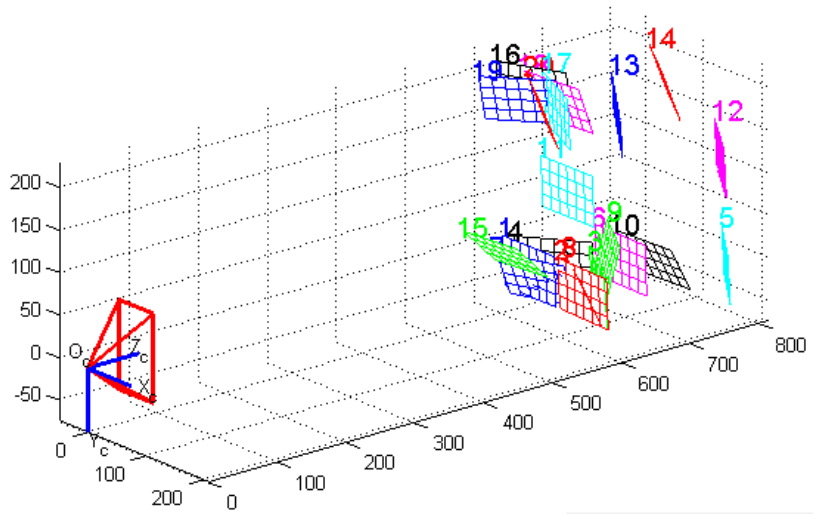


***Figura 28***. *Conjunto de imágenes mostradas según su posición con respecto a la cámara.*

A partir de la calibración de cada cámara, se puede realizar el proceso para una calibración en estéreo, es decir, la relación que hay entre una cámara y la otra. Los pasos son los mismos que para la calibración individual, pero en vez de cargar imágenes y seleccionar sus esquinas, se cargan los archivos con la información de cada cámara, generados en el proceso anterior. Los resultados son los siguientes:

| **Estéreo** | |
|---|---|
| Distancia focal: | $fc = [1476.02 \quad 1497.97\,] \pm [36.01 \quad 39.25]$ |
| Punto principal: | $cc = [526.81 \quad 1076.65\,] \pm [41.27 \quad 49.75]$ |
| Distorsión: | $kc = [0.14 \quad -0.19 \quad 0.02 \quad -0.00] \pm [0.06 \quad 0.14 \quad 0.01 \quad 0.00]$ |
| | |
| Vector de rotación: | $\bar{r} = [-0.035 \quad 0.020 \quad 0.035\,] \pm [0.03 \quad 0.02 \quad 0.00]$ |
| Vector de traslación: | $\bar{t} = [-204.20 \quad -0.99 \quad -2.70\,] \pm [2.53 \quad 1.05 \quad 6.82]$ |

De los resultados de la calibración estéreo se obtiene el vector de rotación $\bar{r}$, que corresponde a los valores de la diagonal principal de la matriz de rotación $R$, donde

$$r_i \equiv R_{ii}, \tag{16}$$

además del vector de traslación $\bar{t}$.

Como punto final del proceso, se muestra la representación gráfica de la posición de las cámaras con respecto una de la otra, tomando como referencia las imágenes obtenidas por las cámaras (**Figura 29**).

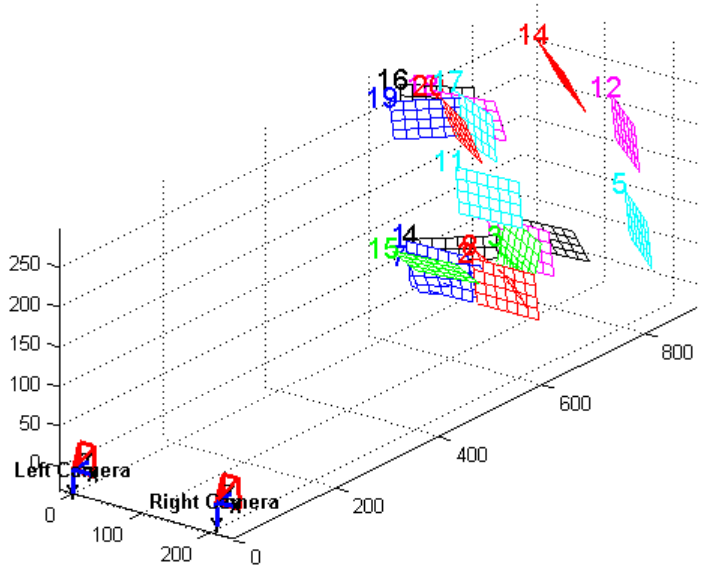


***Figura 29**. Posicionamiento del conjunto de imágenes con respecto a ambas cámaras.*

**3.2.2** *Colocación del sensor*

Algunos de los elementos de este sistema ya han sido considerados para funcionar en conjunto, como es el caso del casco Oculus Rift y el sensor Leap Motion. Existen desarrollos que incluyen ambos dispositivos, sin embargo el enfoque es solamente sobre Realidad Virtual. El sensor de movimiento cuenta con un soporte comercial que puede ser adherido al frente del casco para así captar los gestos realizados por el usuario al colocar las manos delante de él.

El usuario deberá realizar los movimientos de manos frente a su cabeza para que el sensor pueda reconocerlos. Esto no representa un problema pues el campo de interacción del sensor queda completamente cubierto por el campo de visión del casco de Realidad Virtual, el cual despliega los objetos virtuales frente al usuario en todo momento. El dispositivo realiza el seguimiento de las manos del usuario, así como reconocimiento de algunos gestos simples, lo que permite una interacción con la escena. Es posible hacer seguimiento de marcadores, sin embargo, esa característica no está implementada en esta versión del proyecto.

### **3.2.3** *Despliegue de elementos virtuales*

En el desarrollo de este proyecto los elementos virtuales consisten de modelos simples, cubos o esferas, que son desplegados sobre las imágenes del mundo real en una posición fija. Aparecer en las mismas coordenadas simplifica la implementación al no hacer uso de marcadores en la escena, ya sean fiduciarios (bidimensionales) o retroreflectivos (esferas) (**Figura 30**).

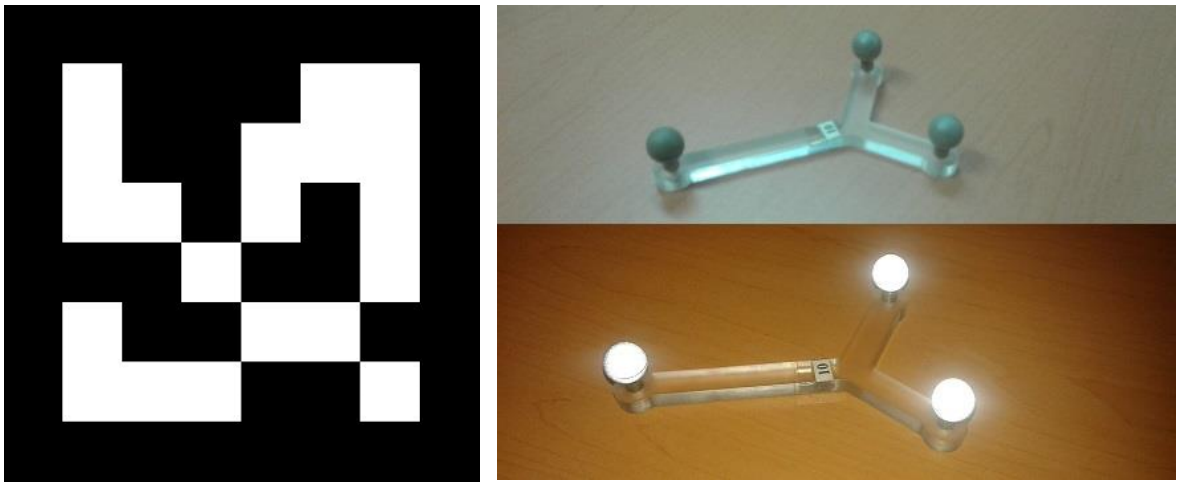

***Figura 30****. Ejemplo de marcador fiduciario (izquierda) y marcadores retroreflectivos (derecha).*

## **3.3** *Programación del sistema*

El soporte central del sistema de Realidad Aumentada es el programa que hace que cada uno de los elementos independientes se integre con los demás de manera coordinada, manteniendo un funcionamiento general estable.

### **3.3.1** *Manejo de modelos 3D*

Los modelos 3D son representaciones computacionales de una superficie tridimensional. Pueden ser desplegados como una imagen 2D por medio de un proceso llamado **Renderizado**, o pueden utilizarse dentro de simulaciones de fenómenos físicos.

Existen distintos y muy variados métodos para generar los modelos 3D, sin embargo, el resultado final es similar en todos ellos. Se obtiene una imagen digital que representa un objeto y usualmente es caracterizada por puntos en el espacio conectados por líneas rectas o curvas, o superficies. Al momento de desplegar estos modelos el principal componente visible es el conjunto de superficies delimitadas (mallados en general y mallados "de alambre" cuando se visualizan sólo aristas).

Antes de continuar la descripción del manejo de modelos 3D se debe mencionar lo que nosotros consideramos como una **textura** u **objeto de textura**. Esta representa una imagen de dos dimensiones que puede ser mapeada sobre una superficie, dando a la misma las características y variaciones de color de la imagen.

Desplegar la información gráfica requiere una estructura que contenga los elementos a mostrar. En el caso de nuestro proyecto, esta estructura funciona como un arreglo de vértices, en donde cada uno de ellos contiene los atributos que caracterizan a cada punto. El vértice puede contener cualquier información referente al punto al que representa. En este proyecto se almacenan las coordenadas de posición *(*x, y, z), los valores de color *(R, G, B, A)* de ese vértice y las coordenadas de correspondencia con la textura *(u, v)* en un mapa 2D.

Para poder llevar a cabo el despliegue de la escena se cuenta con un ***shader***, que es un programa que se ejecuta sobre la unidad de procesamiento gráfico (GPU) y se ocupa de calcular los valores de despliegue en la pantalla durante el proceso de renderizado. Más específicamente, se requiere de un ***vertex shader*** que defina los atributos de cada vértice y cómo serán utilizados durante el proceso de despliegue, y un ***fragment shader*** encargado de definir los atributos de cada pixel de la pantalla (color, coordenadas de textura, etc.).

Habiendo almacenado los vértices que corresponden a los modelos que se quieren mostrar, se puede iniciar el proceso de graficación. Los modelos virtuales son sólo un elemento en el despliegue de nuestro sistema de Realidad Aumentada, por lo que debe haber concordancia entre todos sus componentes.

Se tienen distintos sistemas coordenados que corresponden a los diferentes componentes de este proyecto. Esta versión aún no cuenta con la correlación de todos estos sistemas coordenados, por lo que es imposible conocer la posición del objeto virtual con respecto a la posición del usuario en el mundo real, así como la perspectiva desde la cual se estaría observando el objeto.

Dado lo anterior, la profundidad de la escena virtual se vuelve irrelevante respecto a la profundidad que tiene la escena del mundo real, por lo que se utiliza el método de renderizado, generando una imagen 2D a partir del modelo basándose en los valores de color e iluminación de la escena.

**Renderizar a un elemento de textura** es un método práctico muy simple en donde, en vez de mostrar la escena directamente sobre la pantalla de despliegue, se hace sobre un *buffer* intermedio que se encarga de almacenar esa escena como una imagen temporal, obteniendo así el objeto de textura que será utilizado más adelante. Un *buffer* se define como un espacio de memoria de almacenamiento temporal de información, que permite transferir los datos entre métodos aunque tengan características de transferencia distintas.

Utilizando la biblioteca de OpenGL podemos implementar esta técnica mediante el uso de objetos *framebuffer* (FBO, por sus siglas en inglés). Estos FBOs permiten desplegar la escena fuera de la pantalla, es decir, se renderiza sobre una escena temporal en memoria. A partir de lo anterior se puede leer el contenido de este *buffer*, que en este caso contiene las imágenes correspondientes a la escena, y crear una textura con él.

Además, el *framebuffer* está compuesto por múltiples sub-buffers. Entre los principales se puede encontrar el *buffer* de color, organizado como vectores RGBA (rojo, verde, azul, transparencia (alfa)), y el *buffer* de profundidad, siendo el primero el más utilizado en el despliegue de las escenas. Todos estos sub-buffers son llamados *renderbuffers*.

Podemos resumir el proceso en cinco pasos:

a) Crear el objeto *framebuffer*.
b) Crear un objeto de textura. Se inicializa el objeto para recibir la imagen generada que fue almacenada en el *framebuffer*. Este objeto se estará actualizando mientras existan imágenes almacenadas.
c) Vincular el *framebuffer*. Sin importar el tipo, todos los buffers tienen la misma estructura y funcionamiento, por lo que se debe indicar sobre cuál de todos los buffers se está trabajando.
d) Renderizar la escena fuera de la pantalla, después copiarla al objeto de textura. Realizar la proyección de la escena tridimensional a una imagen bidimensional y convertir esta imagen a un objeto con las características de textura.
e) Desvincular el *framebuffer*. Se debe indicar que ya no se está trabajando con este *buffer* para que las demás funciones puedan hacer uso del mismo o de otros.

Habiendo realizado los pasos anteriores, se tiene el objeto de textura con la imagen correspondiente a un cuadro de la escena 3D. El renderizar la escena fuera de la pantalla permite que esta sea interactiva, es decir, que puede ser modificada en cada cuadro. Lo mismo ocurre con la textura que depende de esa escena, permitiendo mostrar los cambios en ella.

### **3.3.2** *Obtención de las imágenes a partir de las cámaras*

El par de cámaras web está encargado de obtener las capturas del mundo real. Estas imágenes representan la visión del usuario si no tuviera puesto el casco, por lo que es importante que las cámaras hayan sido calibradas correctamente para no perder información, generar cambios en la orientación, alineación y escala, introducción de distorsiones y, en general, evitar la pérdida de sincronización de las imágenes obtenidas pudiendo ocasionar confusión, ambigüedad o incomodidades para el usuario.

Con ayuda de la biblioteca *OpenCV* se extraen las imágenes (cuadros) almacenados en el *buffer* interno de cada una de las cámaras. El cuadro que se obtiene representa el último instante capturado por la cámara, y a pesar de que se pierdan algunos de ellos, la secuencia no pierde continuidad para el usuario.

Los cuadros obtenidos son almacenados como matrices de *M*x*N*x*3*, representando así cada uno de los canales de color de la imagen (R, G, B). En esta ocasión no se toma en cuenta el canal alfa (transparencia) de la imagen pues ésta pasará a ser el fondo de la composición, la cual debe conservar un color sólido en toda la pantalla.

Ya que se tienen los datos en la matriz se le aplican las operaciones de rotación correspondientes pues en el montaje de las cámaras estas se giran para reducir la distancia entre los centros y así las imágenes se obtienen con el mismo giro.

Habiendo aplicado las correcciones a los cuadros se convierten en un objeto de textura de OpenGL y de esta forma se almacenan en el *buffer* circular al que pertenecen (las imágenes capturadas por la cámara izquierda se almacenan en el *buffer* "izquierdo", lo mismo sucede del lado derecho).

El programa cuenta con dos objetos de *buffer* circular para almacenar las imágenes obtenidas. Estos objetos son globales, por lo que cualquier hilo de ejecución puede acceder a ellos, como sucede con el proceso de renderizado final (expuesto más adelante).

### **3.3.3** *Obtención de la información del sensor*

En este módulo se hace un mayor uso de la biblioteca *LM* del sensor Leap Motion comparado con las demás bibliotecas, pues al ser un producto probado cuenta con ciertas funciones que nos permiten reducir el trabajo de detección y rastreo de las manos.

La captura de las manos se basa en los cuadros o *frames*, siendo estos un conjunto de características de la escena adquirida por las cámaras del sensor en un instante. Estos cuadros contienen información sobre las siluetas de las manos encontradas y su posición, así como la posición de los dedos, y la estimación de la posición del brazo. Existen funciones sobre el objeto del cuadro que nos permiten obtener los datos que necesitamos, como la posición de la mano y los dedos, que será utilizada a futuro para lograr una interacción con los objetos virtuales.

A partir de la información almacenada se puede reconstruir la representación gráfica de las manos, obtenidas utilizando las funciones de esqueleto pertenecientes a la biblioteca. Con esto obtenemos una escena tridimensional, como las generadas por los objetos virtuales, a la que podemos aplicar el mismo proceso de renderizado y obtener una imagen a partir de la escena.

Para este proceso de renderizado se utilizan los mismos *shaders* que en la etapa de manejo de modelos, pues el proceso de despliegue es el mismo en ambos casos.

En las aplicaciones de Realidad Virtual se pueden utilizar modelos 3D prediseñados y acoplarlos al esqueleto generado, brindando un mayor realismo e inmersión a la escena observada al simular manos reales. Sin embargo, en nuestro caso no es indispensable el despliegue de modelos, pues se cuenta con la imagen de las manos reales del usuario tomadas por las cámaras web.

Dado lo anterior, el despliegue del esqueleto sobre las manos del usuario pasa a ser una función opcional pues el acoplamiento de las imágenes (explicado más adelante) no es exacto, por lo que podría causar confusión al no coincidir en su totalidad las manos con el esqueleto.

### **3.3.4** *Acoplamiento de los cuadros de video provenientes de diferentes fuentes*

Habiendo obtenido las imágenes de cada una de las distintas fuentes, estas nos servirán para conformar la imagen final a desplegar en la pantalla del casco de Realidad Virtual Oculus Rift.

Las imágenes se encuentran almacenadas como objetos de textura dentro de sus respectivos buffers, de donde se extraen de una por una (el proceso de extracción se encuentra descrito más adelante), respetando el orden en el que fueron almacenadas y así mantener la continuidad de los cuadros.

La primera textura a considerar es la obtenida por las cámaras. Este elemento representa el fondo de nuestra escena final, por lo que será la base sobre la cual se añadirán las siguientes texturas.

La segunda fuente incorpora el modelo virtual, generado por computadora. A pesar de que este se encuentra en una posición fija dentro de la pantalla del dispositivo, el usuario tiene la capacidad de moverse en el entorno sin perder de vista el objeto, pues éste se mueve con él.

La tercera fuente se considera un elemento opcional, pues incorpora los elementos visuales generados a partir de la información recopilada por el sensor de movimiento Leap Motion. Como se describe anteriormente, esta textura representa un esquema simplificado de los huesos de la mano. Utilizado principalmente en ambientes puramente virtuales, esta imagen proporciona al usuario la posibilidad de ubicar sus manos y brazos espacialmente en dicho ambiente. Sin embargo, aún existen errores al incluir esta última textura, pues la alineación entre el esqueleto y las manos no es precisa y podría llegar a confundir al usuario. Por lo anterior, este elemento puede o no ser desplegado.

Dos de los tres objetos de textura cuentan con un canal alfa, es decir, un valor de transparencia para cada uno de sus pixeles. Esta capa facilita la combinación de las imágenes, pues se pueden encimar sin modificar los pixeles de la textura base, que permanecen en las zonas traslúcidas. Para el desarrollo de este proyecto sólo se manejan objetos sólidos, por lo que los valores de este canal son 1 o 0 solamente (totalmente opaco o totalmente transparente).

#### **3.3.4.1** *Uso de shaders*

Como se mencionó anteriormente, los *shaders* son los encargados de calcular los valores de despliegue de la escena que se muestra en la pantalla. En este proyecto se hace un gran uso de ellos, pues la etapa de programación se basa, principalmente, en el acoplamiento de los distintos elementos obtenidos de cada uno de los dispositivos. Los *shaders* tienen su propio lenguaje de programación GLSL[20].

Por un lado se tiene un solo *vertex shader*, pues todos los elementos gráficos que se quieren desplegar mantienen el mismo conjunto de características en sus vértices: posición, color y coordenadas de textura. Sin embargo, se utilizan dos *fragment shaders* distintos, ambos encargados de definir las características de cada pixel en la pantalla.

El primero se enfoca en aplicar el proceso de renderizado a los modelos generados computacionalmente (el modelo 3D y el esqueleto de las manos obtenido por el sensor). Tomando los atributos obtenidos del *vertex shader* y la perspectiva de la cámara en la escena virtual, ubica cada vértice en una posición de la pantalla por medio del proceso de proyección descrito en el capítulo de Antecedentes.

Tomando los resultados de distancia entre las cámaras obtenidos en el proceso de calibración, se hace uso de estos para relacionar los sistemas coordenados de las cámaras y el sensor. El sensor se encuentra justo a la mitad del casco de Realidad Virtual, por lo que su centro es considerado el origen en este sistema conjunto de coordenadas.

Las cámaras del sensor de movimiento se encuentran a una distancia aproximada de 175mm del origen, con un margen de error de ±1mm. Las cámaras web tienen una separación de 204mm entre sí, al encontrarse colocadas de manera equidistante, una a cada lado del casco, su distancia al origen es de -102mm para la cámara izquierda y 102mm para la cámara derecha, alineadas a la misma altura y profundidad que el sensor, con un margen de error de ±2mm en dirección a los tres ejes coordenados.

[20] https://www.opengl.org/documentation/glsl/

El segundo *fragment shader* se considera el principal, pues es el encargado de combinar las imágenes obtenidas de las distintas fuentes (**Figura 37**), además de aplicar funciones de distorsión explicadas más adelante. Se tienen dos de estos *shaders* (uno para cada mitad de la pantalla), sin embargo ambos son iguales, difiriendo en los valores de corrimiento los cuales se consideran como espejo (por ejemplo: 1 para izquierda y -1 para derecha).

Las imágenes obtenidas han sido almacenadas como matrices, sin embargo el *shader* trabaja individualmente por lo que utiliza un vector de tamaño 4, el cual representa el valor de cada una de las capas de color y canal alfa en ese fragmento de la superficie.

Para conseguir la imagen resultante se realiza lo siguiente:

a) Se tiene un parámetro de corrimiento (valor obtenido a partir de los resultados de calibración), con el cual se generan nuevas coordenadas de textura para las imágenes correspondientes al sensor y al objeto virtual.
b) Se generan 3 vectores que representan el valor de la textura en las coordenadas originales para el fondo, y en las nuevas coordenadas para las imágenes a superponer.
c) Existe una función prediseñada del lenguaje GLSL que permite combinar los valores de dos texturas por medio de un factor en un intervalo de 0 a 1, donde:

$$mix(b, s, f) = \begin{cases} b & si\ f = 0 \\ s & si\ f = 1 \end{cases} \qquad (17)$$

   Es decir, si el valor de opacidad de la imagen a superponer es 1 (totalmente sólido), el valor de color de esta imagen sobrescribe el valor de la imagen base.
d) Primero se realiza la operación con la textura obtenida del objeto virtual, y posteriormente con la textura obtenida por el sensor siendo esta última opcional.
e) El resultado final se toma como el color de salida para el pixel de la pantalla que se está calculando.

```
vec2 nuevoTexcoord = vec2(Texcoord.x + corrimiento, Texcoord.y);

vec4 texura_Base = texture(camaras, Texcoord);
vec4 texura_Modelo = texture(modelo, nuevoTexcoord);
vec4 texura_Esqueleto = texture(esqueleto, nuevoTexcoord);

vec4 texura_Final = mix(texura_Base, texura_Modelo, texura_Modelo.a);

if(display_on)
    vec4 texura_Final = mix(texura_Final, texura_Esqueleto,
                                texura_Esqueleto.a);

outColor = textura_Final;
```

***Apéndice A**. Shader encargado de combinar 3 imágenes generadas por distintas fuentes.*

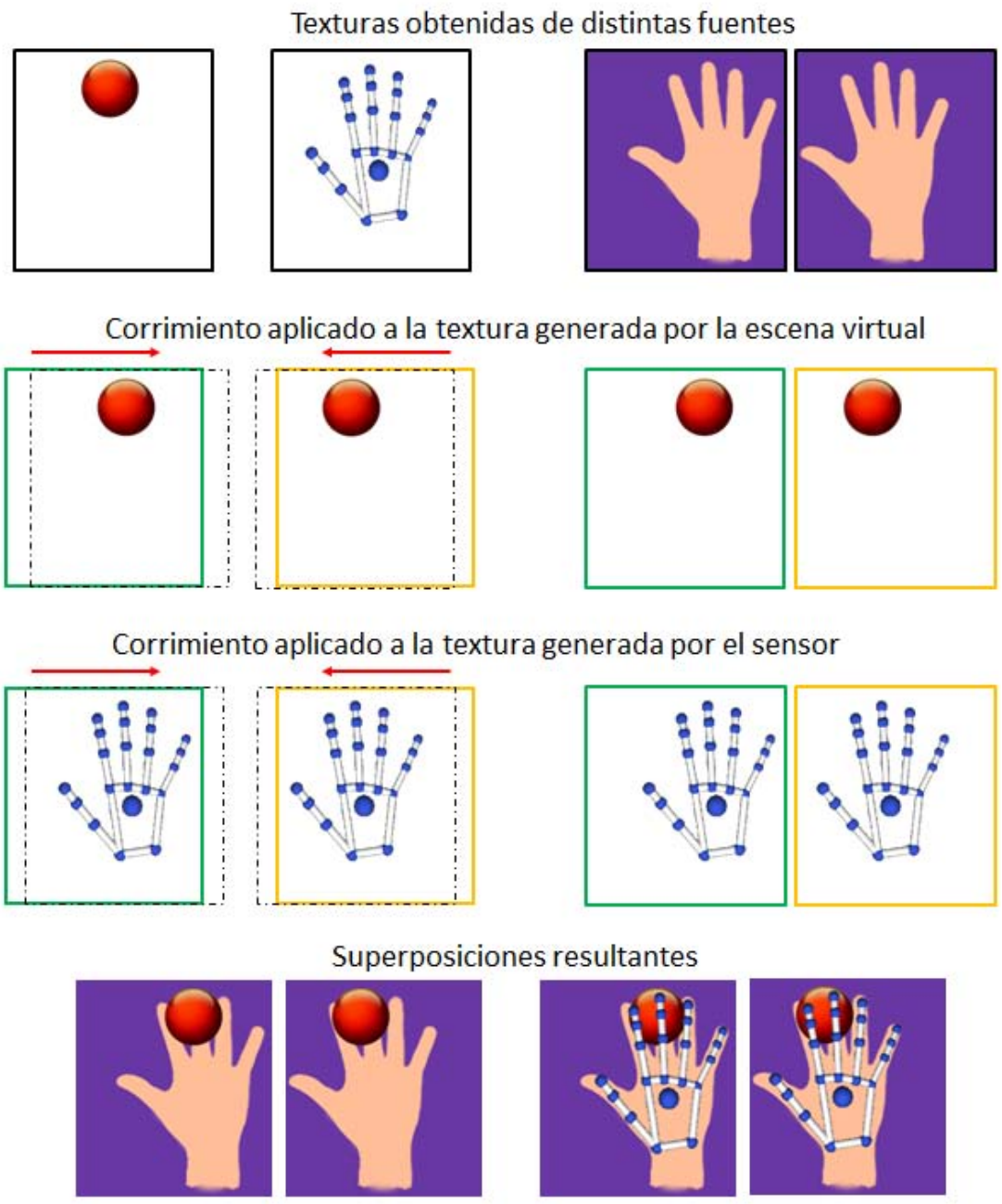


***Figura 31***. *Proceso de corrimiento y combinación de imágenes de distintas fuentes.*

### **3.3.5** *Integración del sistema sobre la pantalla del casco de Realidad Virtual*

El despliegue de la textura fusionada a partir de las imágenes de distintas fuentes se lleva a cabo por el proceso de renderizado de OpenGL. Se crea un arreglo de 6 vértices que conforman dos elementos gráficos (planos) unidos por los vértices centrales. Cada uno de estos planos representa una mitad de la pantalla del casco de Realidad Virtual.

Los planos son como cualquier objeto gráfico de OpenGL, cuentan con sus coordenadas de textura que corresponden a las texturas combinadas (izquierda y derecha), que indican sobre cuál de los planos se despliega cada una.

Como se menciona al describir el casco de Realidad Virtual Oculus Rift, el juego de lentes amplía el campo de visión permitiendo una mayor inmersión en la escena, sin embargo esto provoca una distorsión de efecto cojín, o *pincushion*, que debe ser contrarrestada en el despliegue de la imagen, es decir, aplicando la distorsión contraria conocida como distorsión de barril (*barrel*) (**Figura 32**).

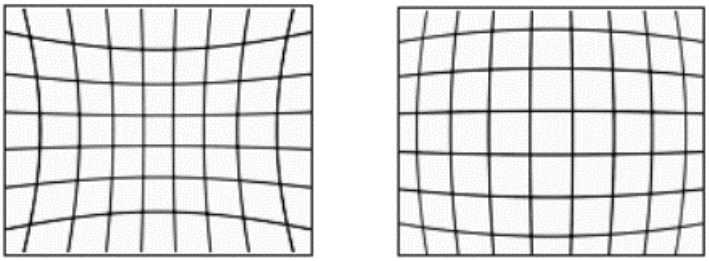

***Figura 32**. Distorsión pincushion (izquierda) y distorsión de barril (derecha).*

En la **Figura 33 (a)** se presenta un ejemplo de cómo se ve una escena vista en la pantalla del casco de Realidad Virtual sin tener los lentes de por medio. Utilizando esta misma escena sin ninguna modificación, y colocando los lentes en su lugar, se puede observar en la **Figura 33 (b)** cómo éstos afectan la visión del usuario, curvando las líneas de la imagen conforme estas se acercan más a las esquinas de la pantalla, provocado por la distorsión de *pincushion.*


***Figura 33***. *Despliegue de una escena sobre la pantalla del casco de Realidad Virtual, sin distorsión (a) y con distorsión ocasionada por los lentes (b).*

El *fragment shader* es el encargado de calcular los cambios de posición de la textura sobre los planos. Corregir la distorsión de *pincushion* generada por los lentes se consigue reposicionando cada pixel de una manera radial, tomando como base una función polinomial y teniendo en cuenta que los coeficientes de esta función, equivalen a los brindados por la biblioteca *LibOVR*.

La función relaciona la distancia entre un pixel al centro de la imagen original ($r_u$) con la misma distancia correspondiente en la imagen distorsionada ($r_d$):

$$r_d = r_u(\kappa_1 r_u^3 + \kappa_2 r_u^2 + \kappa_3 r_u + \kappa_4), \quad (18)$$

$$r_u = \sqrt{u^2 + v^2}, \quad (19)$$

$$r_d = \sqrt{u'^2 + v'^2}, \quad (20)$$

Donde $(u, v)$ son las coordenadas de textura de la imagen original, $(u', v')$ las coordenadas de textura de la imagen distorsionada y $\kappa_i$ los coeficientes de distorsión brindados por el dispositivo.

El proceso para la corrección de la textura final es como sigue:

a) Se toma la coordenada de textura que se va a calcular y se le aplica el factor de corrimiento, determinado por la distancia entre el centro del lente al centro de la pantalla.
b) Se calcula el factor de distorsión que se va a aplicar a ese punto, dependiendo de la distancia al centro de la distorsión:
    a. Encontrar la distancia entre el centro de la distorsión y el punto que se está calculando.
    b. Dados los coeficientes de distorsión. Resolver la ecuación polinomial indicada en la ecuación (17), multiplicando la distancia anterior por cada uno de los coeficientes de distorsión. Esto nos da lo que llamamos "factor de distorsión".
    c. Multiplicar los valores originales de las coordenadas del punto a calcular por el factor de distorsión. Los nuevos valores obtenidos, son las coordenadas donde se debe desplegar este punto.
c) Se escala la coordenada dados los factores de "escalamiento de distorsión", propios del *shader* que se indican al momento de su definición.
d) Se convierten las nuevas coordenadas de textura obtenidas para que estén dentro del rango válido [0.0, 1.0]
e) Se verifica que no salga del área a desplegar, en caso de que suceda, el valor corresponde al color de fondo establecido por OpenGL.

```
//Grab the texture coordinate, which will be in the range 0-1 in both X and Y
vec2 offset = textureCoordsToDistortionOffsetCoords(vUV);

//Determine the amount of distortion based on the distance from the lens
center
float scale = distortionScale(offset);

//Scale the offset coordinate by the distortion factor introduced by the Rift
lens
vec2 distortedOffset = offset * scale;

//Now convert the data back into actual texture coordinates
vec2 actualTextureCoords = distOffsetCoordsToTextureCoords(distortedOffset);
outColor = texture(u_texture, actualTextureCoords );

//The actual texture data doesn't necessarily occupy the entire width and
height of the texture, so we apply a scale that has been provided by the
application to only access the parts of the texture that are valid
actualTextureCoords *= u_texRange;

//Ensure that the distorted coordinates are not outside the range of the
texture
vec2 clamped = clamp(actualTextureCoords, vec2(0, 0), u_texRange);

if (!all(equal(clamped, actualTextureCoords))) {
    outColor = vec4(0, 0, 1, 1);
} else {
    outColor = texture(u_texture, actualTextureCoords );
}

outColor = texture(u_texture, vUV);
```

***Apéndice B***. *Fragment shader que corrige la distorsión generada por los lentes del casco de Realidad Virtual Oculus Rift.*

### **3.3.6** *Integración del sistema en tiempo real*

El sistema está pensado para correr en tiempo real por lo que se recurrió al uso de funciones multihilos, que permiten la ejecución de distintas tareas independientes al mismo tiempo. En este proyecto se utilizan 4 hilos que corren en paralelo: 3 hilos secundarios además del hilo de ejecución principal.

El hilo principal está encargado del arranque del programa así como de inicializar las bibliotecas, el casco de Realidad Virtual Oculus Rift, y realizar los procesos de renderizado, tanto el de las escenas 3D, como la graficación de los planos a los que se les aplica la textura generada. Además de la inicialización, y siendo que el objeto virtual estará en una posición fija de la pantalla, no es necesario generar el objeto de textura correspondiente a este objeto para cada cuadro, por lo que esta textura se crea y almacena desde el inicio del programa indicando que utilice los *shaders* adecuados.

Dos de los hilos secundarios son encargados de iniciar las cámaras, así como capturar las imágenes que provienen de estas, girarlas en el sentido correspondiente y almacenarlas en cada uno de los buffers según sea el caso. Estos hilos funcionan a través de una función que se ejecuta en automático desde la creación del hilo.

El tercer hilo representa la ejecución del sensor de movimiento Leap Motion. Su funcionamiento es similar a los hilos correspondientes a las cámaras, pues este se encarga de obtener la información recogida por el sensor y almacenarla. A diferencia de los anteriores, este hilo toma las coordenadas de posición de los puntos importantes de las manos, como las puntas de los dedos, que han sido indicadas por el sensor, y se almacenan en un vector, respetando la estructura de los vértices definida anteriormente: Posición en el espacio, valores de color y coordenadas de textura. En este caso no se aplica ninguna textura al modelo, por lo que se asigna un valor de 0.0 a todos los vértices.

Habiendo obtenido un arreglo de vértices se puede generar el elemento textural correspondiente a este cuadro de la escena. Utilizando los mismos *shaders* que se usan para generar el elemento del modelo, se genera el objeto de textura y se almacena en un *buffer* circular al igual que las imágenes de las cámaras.

Mientras los hilos secundarios realizan sus tareas recabando la información requerida, el hilo principal se encarga de darle forma. Este tiene acceso a todos los buffers de almacenamiento, así como al elemento textural del objeto a virtual a desplegar.

El proceso de renderizado se ejecuta para cada cuadro o *frame* en la secuencia de la siguiente manera:

a) Se toman los tres elementos texturales de los buffers correspondientes al cuadro a mostrar. Lo anterior se logra ubicando el último elemento añadido a los tres distintos buffers, siendo estos los más actuales, y almacenándolos temporalmente.
b) El programa utiliza dos *fragment shaders*, uno para la pantalla izquierda y otro para la derecha, por lo que se tiene que activar el que corresponde al plano sobre el que se va a trabajar. La descripción continúa sólo para un lado de la pantalla, para el otro lado se realizan las mismas instrucciones activando el otro *shader*.
c) Habiendo definido qué *shader* es el que realizará los cálculos, se indican los elementos a desplegar, así como las texturas con las que estará trabajando
d) Por último, se hace el despliegue en pantalla.

```
g_leftTexture = loadMat(g_leftBuffer.back());
g_rightTexture = loadMat(g_rightBuffer.back());
g_sensorTexture = loadMat(g_sensorBuffer.back());

//Use left shader
glUseProgram(g_leftShader.program);

//Bind left quad element array and corresponding texture
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, g_EBO[0]);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(elementsL), elementsL, GL_STATIC_DRAW);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, g_leftTexture);
glUniform1i(g_leftShader.uniform.u_texture, 0);

glDrawElements(GL_QUADS, 4, GL_UNSIGNED_INT, 0);

//Use right shader
glUseProgram(g_rightShader.program);

//Bind right quad element array and corresponding texture
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, g_EBO[1]);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(elementsR), elementsR, GL_STATIC_DRAW);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, g_rightTexture);
glUniform1i(g_rightShader.uniform.u_texture, 1);

glDrawElements(GL_QUADS, 4, GL_UNSIGNED_INT, 0);

//Swap front and back buffers
glfwPollEvents();
glfwSwapBuffers(g_hPrimaryWindow->w_Window);
```

***Apéndice C**. Código de renderizado con OpenGL.*

Al recibir los valores indicados en los parámetros, el *fragment shader* se ejecuta realizando la combinación de las 3 imágenes y generando los cálculos de distorsión, vistos con anterioridad, que corresponden a cada lado de la pantalla. La nueva textura que genera el *shader* es la que será desplegada sobre cada uno de los planos.

Al finalizar la ejecución del programa, el hilo principal también es el encargado de destruir los objetos creados al inicio, cerrar las bibliotecas pendientes y limpiar la memoria.

# Capítulo IV. RESULTADOS Y DISCUSIÓN

Durante la ejecución del programa se pueden identificar etapas intermedias que van cubriendo los objetivos específicos de este proyecto. A lo largo de este capítulo se analizan los resultados obtenidos en cada una de estas etapas, haciendo una recopilación al final de los resultados generales obtenidos.

Para comenzar el análisis, se tienen los módulos de obtención de las imágenes. Como se describió a lo largo de la metodología, las imágenes se obtuvieron de 4 fuentes distintas: dos cámaras web encargadas de representar la visión del usuario, un sensor de movimiento y la cuarta se genera computacionalmente.

El programa no trabaja directamente sobre las imágenes, sino sobre los elementos texturales generados a partir de ellas. Estos elementos requieren de un procesamiento previo de las imágenes obtenidas.

Iniciando por las imágenes obtenidas por el par de cámaras web Logitech c920. La obtención original de las imágenes tiene un giro de 90° y -90° ya que las cámaras están montadas con estos giros sobre el casco. En las **Figura 34** y **35** se puede apreciar el estado original y el resultante de este proceso.

***Figura 34**. Imágenes originales obtenidas por las cámaras (arriba). Imágenes obtenidas por las cámaras corregidas por medio de rotación (abajo).*

***Figura 35**. Par estéreo de imágenes obtenidas por las cámaras web, después de haber aplicado la rotación.*

Continuamos con  la imagen que resulta de procesar los valores de posición de las manos, obtenidos por el sensor de movimiento Leap Motion. Estos datos representan las coordenadas de posición de puntos específicos en la estructura de la mano, como la punta de los dedos, el centro de la mano, la estimación de la posición del brazo, etc.

La biblioteca específica del sensor, biblioteca *LM*, nos permite transformar el conjunto de coordenadas por una representación gráfica equivalente, la cual es utilizada como el elemento textural correspondiente al sensor.

La **Figura 36** muestra el conjunto de información de posición obtenida, desplegado como el esqueleto de las manos. La transformación por medio del proceso de renderizado es la función principal de este fragmento del programa, pues se realiza la conversión de los datos a vértices que mantengan la estructura establecida desde el comienzo y así poder desplegar una escena 3D, para posteriormente convertirla al elemento textural.

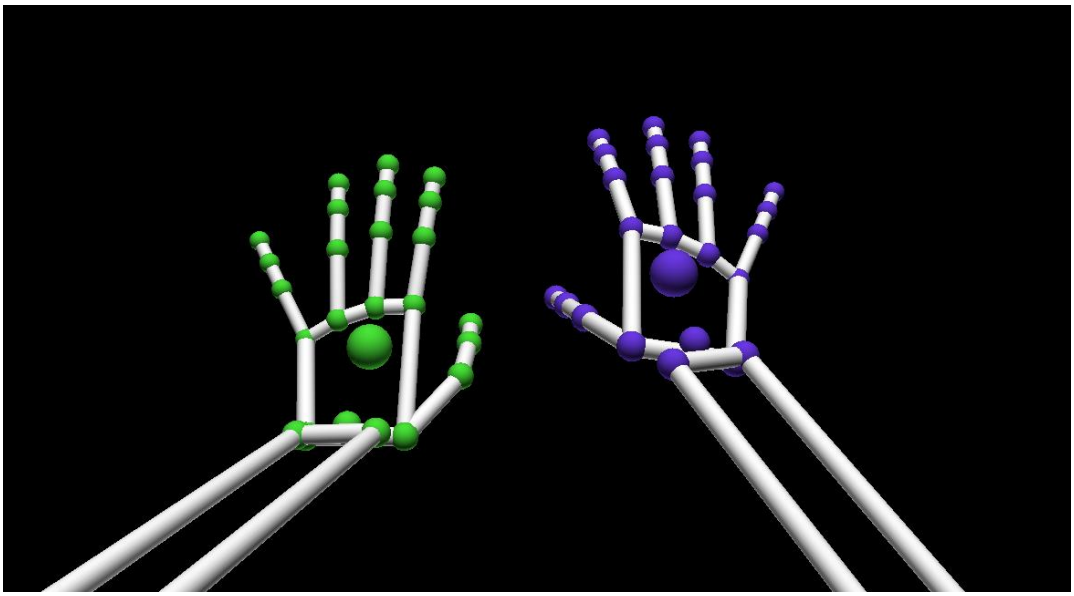

***Figura 36**. Conversión del conjunto de coordenadas obtenidas por el sensor a su representación gráfica.*

Se debe mencionar que la ejecución por la que pasa el conjunto de datos, desde la obtención hasta la textura, consume más tiempo al compararlo con el despliegue de la escena virtual, pues debe atravesar, además, por el proceso interno de convertir la información a su representación gráfica, a diferencia de las demás ejecuciones de este módulo de obtención de imágenes en donde se realiza de manera directa o con un solo proceso.

Para conseguir el elemento de textura del objeto virtual se sigue un procedimiento distinto a los dos anteriores, pues no se tiene ninguna base para comenzar a trabajar la imagen. Se esboza un objeto tridimensional en el espacio, estimando su posición final en la pantalla.

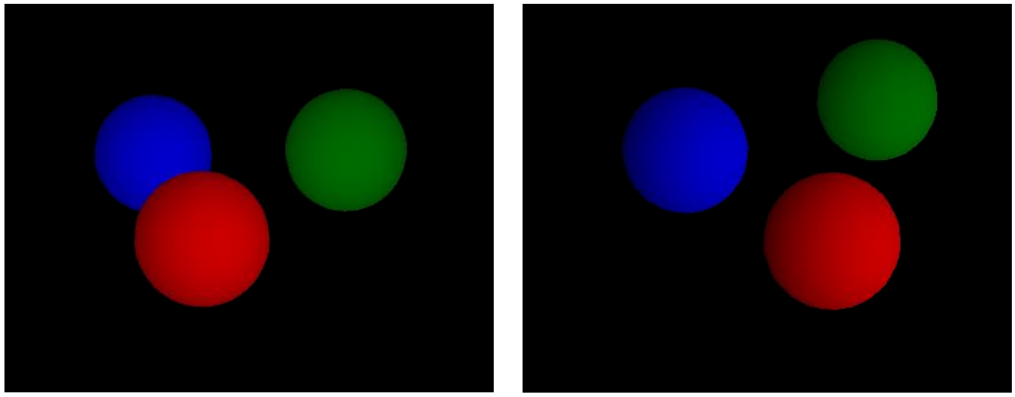

***Figura 37**. Proyección de la escena 3D desde dos puntos de vista distintos.*

Como se puede observar en la **Figura 37**, la textura resultante depende del punto de vista de la cámara. Como se mencionó anteriormente, esta versión del proyecto aún no cuenta con la relación de los sistemas coordenados, por lo que la posición del elemento virtual permanece en una posición fija dentro de la pantalla de despliegue.

Actualmente sólo se obtiene una imagen resultante de la escena virtual, de la cual se obtiene un par de imágenes estéreo, aplicando el corrimiento correspondiente para mostrarse en la pantalla izquierda y derecha.

El siguiente módulo que se describe se enfoca en la alineación de las imágenes. Como primera etapa observamos la combinación de la imagen proveniente de las cámaras con la imagen generada a partir del modelo 3D. Como se puede apreciar en la **Figura 38**, el despliegue del modelo permanece en la misma posición de la pantalla, además, es desplegado por encima de la imagen de las cámaras, lo que puede llegar a ocasionar oclusión de algunos elementos que estarían delante en perspectiva.

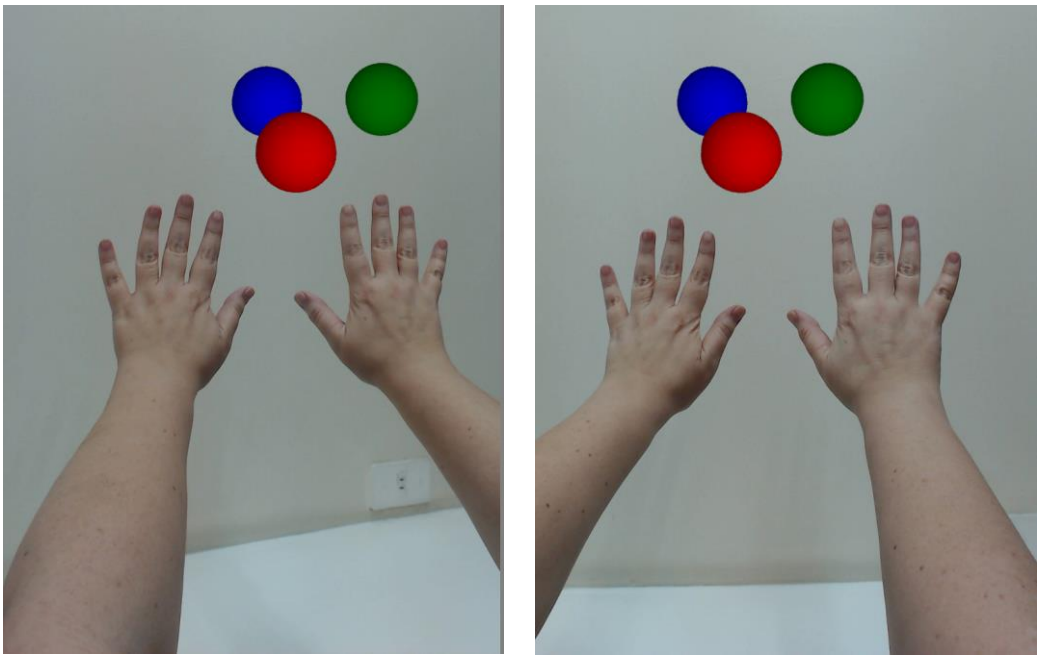

***Figura 38**. Ejemplo de superposición de la escena virtual sobre el par estéreo de imágenes obtenido por las cámaras web.*

Corregir los problemas de profundidad requiere de reconocimiento de características de la escena, uso de marcadores y técnicas de seguimiento que permitan ubicar al objeto en la posición adecuada, no sólo en coordenadas (*x, y*) de la pantalla, sino también hacia el fondo.

A partir del resultado anterior, podemos continuar a la siguiente etapa al superponer la imagen generada a partir del sensor de movimiento (**Figura 39**). Esta operación se realiza al finalizar la primera alineación de imágenes, puesto que es una acción opcional.

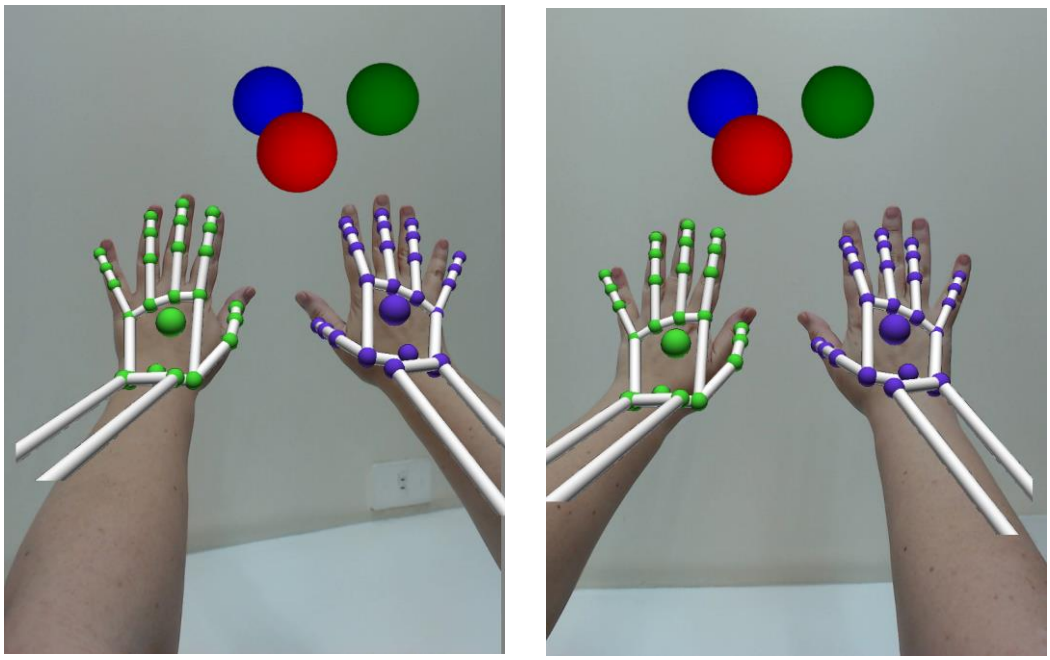

***Figura 39**. Ejemplo de la superposición de la imagen obtenida por el sensor sobre la superposición previa (cámaras y gráficos 3D).*

Aún se pueden observar ciertos errores en la alineación del modelo del esqueleto de la mano propuesto por el sensor, con la imagen de la mano real, esto se debe a que los valores de posición no son exactos y el corrimiento de las imágenes del sensor se realiza por medio de un factor obtenido a partir de la calibración de las cámaras.

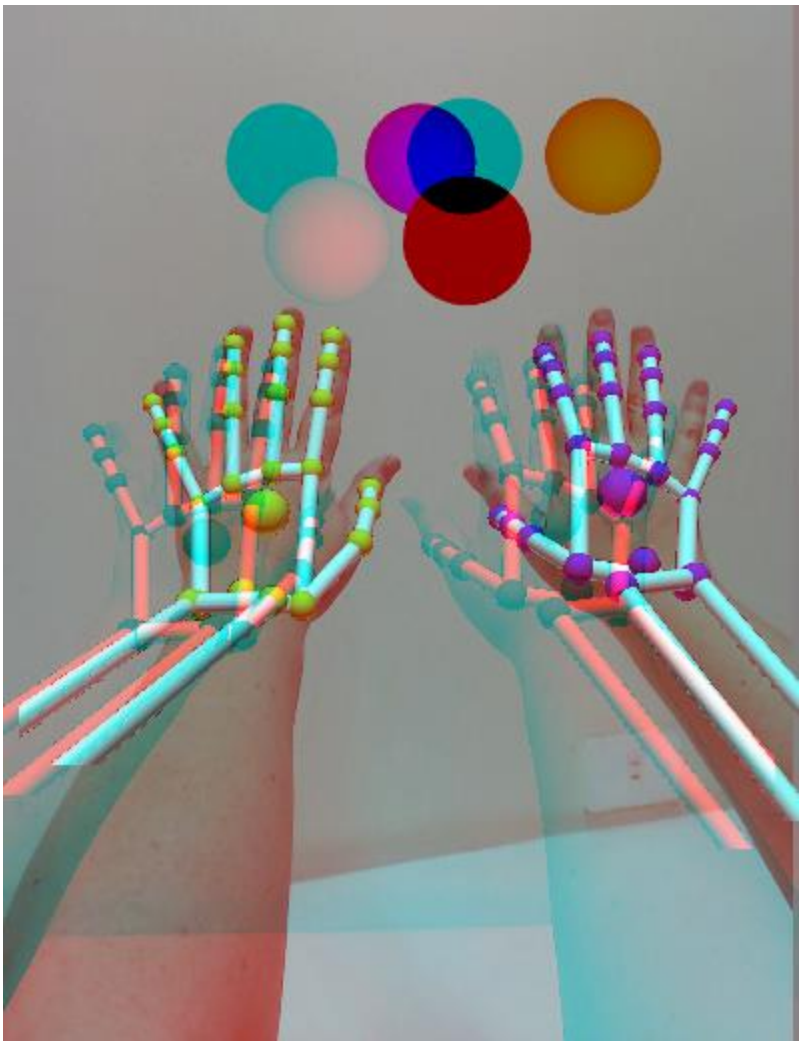

***Figura 40***. *Imagen de anaglifo resultante del par estéreo de texturas finales.*

Continuando con el análisis de la relación de las imágenes se considera que, tanto las cámaras como el sensor, se encuentran alineados sobre el mismo eje coordenado. Esto nos permite que para las operaciones de alineación sólo se requiera hacer movimientos de traslación entre las imágenes. Sin embargo, las alineaciones físicas de los dispositivos se realizaron a mano, por lo que los resultados, aunque corregidos por la rectificación del proceso de calibración, aún tienen cierto grado de error.

El proceso de despliegue en sí, es una etapa sencilla del proyecto, pues resulta de indicar al *fragment shader* los atributos necesarios para que éste realice los cálculos pertinentes para mostrar la imagen final. Sin embargo, el trabajo del *shader* es el que se considera de suma importancia, pues de este depende que haya concordancia entre las imágenes superpuestas, así como la distribución y corrección de distorsión adecuadas para presentarse sobre la pantalla del caso de Realidad Virtual Oculus Rift.

Como se explica en capítulos anteriores, los lentes que complementan la pantalla del casco aumentan el campo de visión del usuario brindando una mayor inmersión, mas estos introducen una gran distorsión (de cojín) a la imagen observada. Para poder obtener una imagen que haga sentido para el usuario, se contrarresta la distorsión generada por los lentes con una distorsión opuesta (de barril).

En la **Figura 39** se observa el despliegue original de las texturas alineadas, así como un conjunto de resultados de la distorsión de barril específica para la primera versión del casco, con distintos valores en sus parámetros en la **Figura 41**.

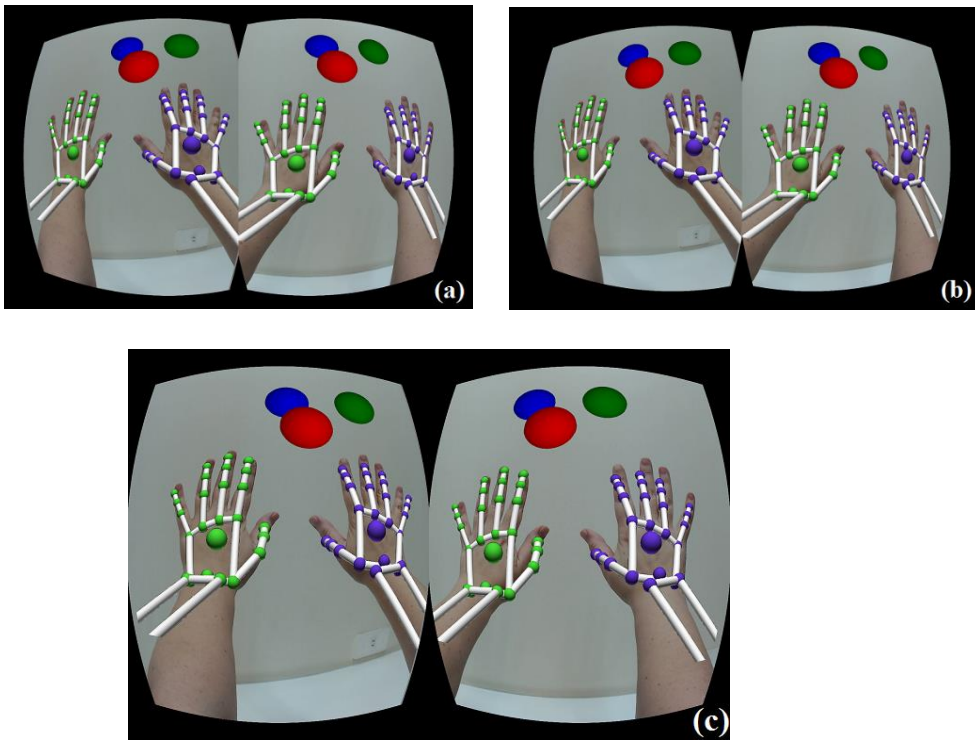


***Figura 41***. *Despliegue de planos con la textura resultante utilizando distintos factores de escalamiento de la distorsión (a), (b), (c).*

Se utilizaron valores de escalamiento de la distorsión (**u_fillScale(*x, y*)** en el *shader*) de (*2.0, 2.0*) para **(a)**, (*3.0, 3.0*) para **(b)** y (*2.0, 3.0*) para **(c)**, siendo la última la que mejores resultados arrojó al momento de probar el sistema, pues mantiene relación con la resolución de la pantalla.

Para concluir el análisis de los módulos tenemos la programación en paralelo de los mismos, utilizando la biblioteca *Boost*. Utilizar este tipo de estructuras permite una ejecución más rápida que si se ejecutara secuencialmente. Este proyecto no podría llevarse a cabo de otra forma, pues dado el retraso al obtener los cuadros provenientes de las cámaras y el sensor, sería imposible lograr la coordinación y alineación de los mismos ya que no corresponderían al mismo instante.

La obtención de imágenes en cada hilo presenta cierta discordancia, pues a pesar de que los hilos corren de manera simultánea, la obtención de las imágenes no es precisa. Se requiere un control externo que sincronice la captura de imágenes en el mismo momento para todos los dispositivos conectados.

El sistema aún presenta algunos problemas de latencia, pues a pesar de la programación paralela, los módulos de carga de las imágenes de las distintas fuentes realizan procesos distintos, retrasando el despliegue de los cuadros.

# CONCLUSIONES

En este trabajo de tesis se desarrolló satisfactoriamente un sistema de Realidad Aumentada coordinando sus distintos componentes. Sin embargo, aún queda mucho trabajo por hacer: corregir la alineación de las texturas, implementar la relación entre los sistemas coordenados de cada módulo, lo que permitirá la interacción con la escena virtual brindando una mayor inmersión al sistema.

La programación paralela de funciones permite que el programa se ejecute en tiempo real, mostrando las imágenes sincronizadas a 60 cuadros por segundo, mismos que el usuario necesita para tener continuidad visual de una secuencia de video. Sin el paralelismo, las imágenes obtenidas por las cámaras y el sensor no están sincronizadas pues su adquisición no sucede en el mismo instante de tiempo.

El proceso de lectura y almacenamiento de imágenes, aún causa cierto retraso o latencia en el sistema, pues a pesar que la velocidad de despliegue es la apropiada, los cuadros se despliegan un instante después de adquiridos. Esto se puede resolver aumentando la capacidad computacional en la cual se ejecuta el sistema, y por otro lado, mejorando los módulos de adquisición y procesamiento de imágenes.

Las cámaras web elegidas están enfocadas a ser utilizadas para videoconferencias, dado que sus características facilitan y mejoran la realización de estas actividades. Sin embargo, su alta resolución, capacidad de autoenfoque y amplio campo de visión nos permite utilizarlas dentro del proyecto, siendo una tecnología fácil de incorporar al mismo, además de accesible.

Este proyecto aún tiene mucho trabajo a futuro, pues además de las mejoras que se pueden hacer sobre el sistema, la tecnología avanza rápidamente, proponiendo nuevos dispositivos que pueden mejorar el rendimiento. En 2016 se contará con la siguiente versión del casco de Realidad Virtual Oculus Rift, mejorando la velocidad de despliegue en la pantalla del dispositivo, así como la integración de marcadores y sensores que permiten la ubicación del casco en el espacio, obteniendo la posición y facilitando la tarea de mostrar los elementos virtuales en el sistema coordenado del mundo real.

Dentro de las mejoras que se pueden hacer al sistema, se encuentra la interacción en tiempo real del usuario con los objetos virtuales, implementando el uso de marcadores (retroreflectivos o fiduciarios) para ubicarlos en la escena, interactuar con ellos y cambiar los ángulos de perspectiva según la posición del usuario. Además del reconocimiento en la escena, también se debe implementar el sistema de posicionamiento del casco de Realidad Virtual, el cual brinda información adicional para calcular las perspectivas.

También puede incluirse el reconocimiento de gestos por parte del sensor de movimiento para una interacción más fluida e intuitiva con el ambiente virtual.

El costo actual de estos sistemas aún es muy elevado para una implementación práctica en los hospitales, por lo que se busca implementar prototipos con componentes de costos accesibles y que mantengan la calidad de los resultados.

# REFERENCIAS


Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S. y MacIntyre, B., "Recent advances in augmented reality," *IEEE Computer Graphics and Applications*, vol. 21, no. 6, pp. 34 – 47, Noviembre 2001.

Billinghurst, M., Piumsomboon, T. y Bai, H., "Hands in Space: Gesture Interaction with Augmented-Reality Interfaces," *IEEE Computer Graphics and Applications*, vol. 34, no. 1, pp. 77 – 80, Enero 2014.

Burdea, G. C. y Coiffet, P., "Introduction," en *Virtual Reality Technology*, primera edición, Hoboken, Nueva Jersey: John Wiley & Sons, Inc. 2003, capítulo 1, sección 1.2, pp. 3.

Burke, J. W., McNeill, M. D. J., Charles, D. K., Morrow, P. J., Crosbie, J. H. y Mc Donough, S. M., "Augmented Reality Games for Upper-Limb Stroke Rehabilitation", en *Second International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES)*, 2010, pp. 75 – 78.

Carmigniani, J., Furht, B., Anisetti, M., Ceravolo, P., Damiani, E. y Ivkovic, M., "Augmented reality technologies, systems and applications," *Multimedia Tools and Applications*, vol. 51, no. 1, pp. 341 – 377, Diciembre 2010.

Comport, A. I., Marchand, E. y Chaumette, F., "A real-time tracker for markerless augmented reality", en *The Second IEEE and ACM International Symposium on Mixed Augmented Reality*, 2003, *pp*. 36 – 45.

Gervautz, M. y Schmalstieg, D., "Anywhere Interfaces Using Handheld Augmented Reality," *Computer*, vol. 45, no. 7, pp. 26 – 31, Julio 2012.

Kishino, F. y Milgram, P., "A taxonomy of mixed reality visual displays," *IEICE Trans. Information Systems*, vol. E77-D, no. 12, pp. 1321 – 1329, 1994.

Klein, G. y Murray, D. W., "Simulating Low-Cost Cameras for Augmented Reality Compositing," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 3, pp. 369 – 380, Mayo 2010.

Kolsch, M., Bane, R. y Hollerer, T., "Multimodal interaction with a wearable augmented reality system," *IEEEComputer Graphics and Applications*, vol. 26, no. 3, pp. 62 – 71, Mayo 2006.

Lee, B. y Chun, J., "Interactive Manipulation of Augmented Objects in Marker-Less AR Using Vision-Based Hand Interaction", en *Seventh International Conference on Information Technology: New Generations (ITNG)*, 2010, pp. 398 – 403.

Livingston, M. A., "Evaluating human factors in augmented reality systems," *IEEE Computer Graphics and Applications*, vol. 25, no. 6, pp. 6 – 9, Noviembre 2005.

MacIntyre, B., Rouzati, H. y Lechner, M., "Walled Gardens: Apps and Data as Barriers to Augmenting Reality," *IEEE Computer Graphics and Applications*, vol. 33, no. 3, pp. 77 – 81, Mayo 2013.

Malek, S., Zenati-Henda, N., Benbelakacem, S. y Belhocine, M., "Tracking chessboard corners using projective transformation for augmented reality", en *International Conference on Communications, Computing and Control Applications (CCCA)*, 2011, pp. 1 – 6.

Mine, M. R., van Baar, J., Grundhofer, A., Rose, D. y Yang, B., "Projection-Based Augmented Reality in Disney Theme Parks," *Computer*, vol. 45, no. 7, pp. 32 – 40, Julio 2012.

Moser, K., Itoh, Y., Oshima, K., Swan, J. E., Klinker, G. y Sandor, C., "Subjective Evaluation of a Semi-Automatic Optical See-Through Head-Mounted Display Calibration Technique," *IEEE Transactions on Visualization and Computer Graphics*, vol. 21, no. 4, pp. 491 – 500, Abril 2015.

Navab, N., Blum, T., Wang, L., Okur, A. y Wendler, T., "First Deployments of Augmented Reality in Operating Rooms," *Computer*, vol. 45, no. 7, pp. 48– 55, Julio 2012.

Reitmayr, G. y Drummond, T. W., "Going out: robust model-based tracking for outdoor augmented reality", en *IEEE / ACM International Symposium on Mixed and Augmented Reality*, 2006, pp. 109 – 118.

Roberts, J. C., Ritsos, P. D., Badam, S. K., Brodbeck, D., Kennedy, J., y Elmqvist, N., "Visualization beyond the desktop – the next big thing," *IEEE Computer Graphics and Applications*, vol. 34, no. 6, pp. 26 – 34, Noviembre 2014.

Rolland, J. P. y Fuchs, H., "Optical Versus Video See-Through Head-Mounted Dsiplays in Medical Visualization," *Presence: Teleoperators and Virtual Environments*, vol. 9, no. 3, pp. 287 – 309, Junio 2000.

Sabelman, E. E. y Lam, R., "The real-life dangers of augmented reality," *IEEE Spectrum*, vol. 52, no. 7, pp. 48 – 53, Julio 2015.

Tsai, R. Y., "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses," *IEEE Journal of Robotics and Automation*, vol. 3, no. 4, pp. 323 – 344, Agosto 1987.