UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Diseño de sistemas de realidad virtual con renderizado háptico utilizando los robots *Geomagic Touch* y *Novint Falcon*

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Rafael Huerta Quintero

DIRECTOR DE TESIS:
M. I. José Daniel Castro Díaz

Ciudad Universitaria, Cd. Mx., 2019

JURADO:

PRESIDENTE: M. I. Luis Sergio Valencia Castro

VOCAL: M. I. José Daniel Castro Díaz

SECRETARIO: Ing. José Roque Román Guadarrama

1ER. SUPLENTE: M. I. Roberto Giovanni Ramírez Chavarría

2DO. SUPLENTE: M. I. Sergio Teodoro Vite

Esta tesis se realizó en el Laboratorio de Robótica del edificio de estudios de posgrado de la Facultad de Ingeniería con el apoyo del proyecto **IN114617** de la **DGAPA-UNAM** a cargo del Dr. Marco Antonio Arteaga Pérez.

ASESOR:

M. I. José Daniel Castro Díaz

Dedicatoria

A mis padres Norma y Rafael por brindarme todo su amor, apoyo, regaños, enseñanzas y consejos durante toda mi vida, así como cada uno de los esfuerzos realizados para que pudiera lograr una de mis metas. Por siempre estar ahí conmigo en las buenas y en las malas, no tengo más que agradecerles por todo lo que soy, ya que ustedes son los motores que me impulsan a ser mejor cada día.

A mi hermana Sol, por ser mi gran apoyo en cada momento de flaqueza, así como por las risas, enojos y bromas, pero sin duda por todo su cariño, palabras de aliento y consejos. En verdad gracias hermana por siempre creer en mí.

Agradecimientos

A mi Universidad Nacional Autónoma de México por abrirme las puertas desde el bachillerato y darme la oportunidad de convertirme en un profesional.

A toda mi familia por su cariño, confianza y apoyo incondicional en cada etapa de mi vida.

A mi asesor Daniel Castro por su gran apoyo, dedicación, confianza, esfuerzo y guía en cada etapa de este proyecto, así como sus consejos, palabras de aliento y amistad.

Al profesor Sergio Teodoro por su apoyo en cada consulta que necesité durante la elaboración de este trabajo.

A todos mis amigos que han estado conmigo apoyándome en las diversas etapas de mi vida, ya sea con risas, preocupaciones, alegrías y tristezas pero siempre sabiendo que puedo contar con ellos.

A todos los profesores que fueron parte de mi formación como ingeniero.

A mis compañeros del Laboratorio de Robótica por sus consejos y experiencias durante el desarrollo de este proyecto.

A la UNAM por su apoyo a través del proyecto **IN114617** de la **DGAPA-UNAM**.

Índice general

1. Introducción **1**
- 1.1. Definición del problema . . . 3
- 1.2. Motivación . . . 3
- 1.3. Estado del arte . . . 3
  - 1.3.1. Realidad virtual . . . 4
  - 1.3.2. Características de un sistema de RV . . . 9
  - 1.3.3. Funciones básicas de un sistema de RV . . . 10
  - 1.3.4. Tipos de sistemas de RV . . . 10
  - 1.3.5. Háptica . . . 12
  - 1.3.6. Arquitectura de un sistema de realidad virtual con interacción háptica . . . 13
- 1.4. Contribuciones . . . 14
- 1.5. Estructura de la tesis . . . 14

2. Preliminares **15**
- 2.1. Renderizado visual . . . 16
  - 2.1.1. Motor gráfico . . . 17
  - 2.1.2. Modelado geométrico . . . 17
  - 2.1.3. Color . . . 21
  - 2.1.4. Iluminación . . . 22
  - 2.1.5. Texturización . . . 24
- 2.2. Renderizado háptico . . . 25
  - 2.2.1. Detección de colisiones . . . 26
  - 2.2.2. Renderizado de fuerzas . . . 29
    - 2.2.2.1. Renderizado de fuerzas desde la perspectiva gráfica . . . 29
    - 2.2.2.2. Renderizado de fuerzas desde la perspectiva de control 30
- 2.3. Simulación . . . 33
- 2.4. Comentarios del capítulo . . . 34

3. Los robots *Geomagic Touch* y *Novint Falcon* como interfaces hápticas **35**
- 3.1. Características generales . . . 35
  - 3.1.1. Arquitectura . . . 36
  - 3.1.2. Espacio de trabajo . . . 37

3.1.3. Grados de libertad . . . 37
3.1.4. Cinemática directa . . . 38
3.1.5. Cinemática inversa . . . 39
3.1.6. Interfaz de programación de aplicaciones (API) . . . 39
3.2. Robot háptico *Geomagic Touch* . . . 40
3.2.1. Librería Open Haptics . . . 41
3.2.2. Librería Quick Haptics . . . 42
3.2.3. Haptic Device API (HDAPI-HD) . . . 44
3.2.4. Haptic Library API (HLAPI-HL) . . . 46
3.3. Robot háptico *Novint Falcon* . . . 48
3.3.1. Haptic Device Abstraction Layer (HDAL) . . . 48
3.4. Comentarios del capítulo . . . 51

4. Implementación y resultados **53**
4.1. Sistema de realidad virtual para objetos rígidos utilizando el robot *Geomagic Touch* . . . 53
4.1.1. Renderizado visual . . . 53
4.1.2. Renderizado háptico . . . 56
4.2. Sistema de realidad virtual para objetos rígidos utilizando el robot *Novint Falcon* . . . 63
4.2.1. Renderizado visual . . . 63
4.2.2. Renderizado háptico . . . 64
4.3. Sistema de realidad virtual con interacción háptica para objetos deformables utilizando el robot *Geomagic Touch* . . . 70
4.3.1. Renderizado visual . . . 71
4.3.2. Renderizado háptico . . . 73
4.4. Comentarios del capítulo . . . 77

5. Conclusiones **79**
5.1. Trabajo futuro . . . 80

A. Manual de configuraciones entre el robot *Geomagic Touch* y *Visual Studio* **81**
A.1. Configuración *Visual Studio* . . . 81
A.2. Configuración *Geomagic Touch* . . . 82

B. Manual de configuraciones entre el robot *Novint Falcon* y *Visual Studio* **87**
B.1. Configuración de *Visual Studio* . . . 87

Bibliografía **91**

Capítulo 1

Introducción

A lo largo del tiempo los seres humanos han interactuado con el ambiente que los rodea utilizando la vista, el oído, el olfato, el gusto y el tacto. Gracias a ello fueron capaces de analizar, experimentar y desarrollar herramientas que les han permitido alcanzar una cumbre tecnológica a la que continúan llegando nuevos retos que desafían los límites de su ingenio. Actualmente uno de los campos tecnológicos de mayor desarrollo en donde se están poniendo a prueba dichos límites y que está teniendo un amplio crecimiento es el de la realidad virtual. En éste los seres humanos interactúan de una manera más cercana con la tecnología al estar inmersos dentro de ambientes en los que realizan actividades un tanto complejas de llevar a cabo dentro del mundo real. Llegados a este punto es conveniente plantear las siguientes preguntas: ¿Qué quieren decir exactamente las palabras *realidad virtual* (en lo subsecuente RV)? ¿es acaso una nueva forma de generalizar todo aquello que existe en nuestro alrededor pero no es posible percibir con facilidad? y por lo tanto ¿podemos interactuar o no de la misma forma a la que estamos acostumbrados? En cierto sería correcto contestar negativamente esta última pregunta ya que al adentrarse en un entorno virtual se puede interactuar con sus elementos de maneras que jamás se imaginaría poder hacer en la vida real. El caminar, saltar algún obstáculo o incluso el simple hecho de tocar cualquier objeto se convierte en una experiencia única, pero ¿cómo saber si se ha interactuado alguna vez con un sistema de realidad virtual? Para poder responder esta última pregunta es indispensable tener en cuenta tanto las bases como los alcances que está teniendo este campo de la tecnología.

Hoy en día la RV está presente en gran cantidad de situaciones comunes por lo que puede llegar a ser difícil identificar cuándo se está haciendo el uso de algún sistema virtual. Puede ser desde abrir un navegador de internet y utilizar una herramienta de localización que permite visualizar la superficie terrestre como *Google Earth*, o el simple hecho de sentarse, prender una consola e iniciar una partida en algún videojuego en donde se nos presente una proyección de algún personaje de un mundo de fantasía. En este caso es el usuario quien, de cierta forma, se adentra a vivir una experiencia en el juego. Otra circunstancia habitual en la que se puede interactuar con la RV es una visita a un museo ya que debido a esta innovación tecnológica, muchas instituciones

han buscado brindarle a sus visitantes una experiencia mucho más interactiva. A su vez se pueden encontrar campos con alcances menos comunes como lo son la medicina, en donde se llegan a encontrar desde simuladores quirúrgicos, hasta aplicaciones que pueden ayudar a los pacientes a combatir diversas fobias o bien en el área de la construcción con desarrollos que permiten visualizar futuros edificios, estadios, presas, puentes, etc. Incluso en el campo militar existen simuladores tácticos de entrenamiento enfocados en la conducción, vuelo y disparo, tal como se muestra en la Figura 1.1.

Figura 1.1: Aplicaciones de la RV en distintos entornos [13].

Teniendo en cuenta los diversos campos en que la RV puede estar presente, es de gran importancia conocer que los ambientes virtuales (AV) son sistemas generados por medio de una computadora en donde los usuarios pueden interactuar con diversos entes, cuerpos o proyecciones haciendo uso de sus sentidos. Esto se logra por medio de monitores, cascos, altavoces o sistemas hápticos; siendo estos últimos los que generan una experiencia mucho más inmersiva, pues hace percibir al usuario que se encuentra en contacto con lo que está visualizando, es decir, busca simular la sensación que se presenta al tocar algún cuerpo en el mundo real. En esta tesis se presenta un estudio completo de la interacción entre las partes gráfica y háptica con el fin de conocer el proceso que conlleva implementar un sistema de realidad virtual con retroalimentación táctil.

1.1. Definición del problema

Tal como se mencionó anteriormente, la RV tiene como objetivo primordial estimular los sentidos de las personas para que puedan interactuar de una manera inmersiva con diversos ambientes y objetos ficticios, no solo utilizando la vista como regularmente se hacía en un principio sino también agregando otros sentidos como el tacto. Por ello es fundamental contar con una metodología detallada que permita desarrollar diversas aplicaciones con las que el usuario pueda interactuar tanto de manera visual como háptica, lo que implica asumir retos como:

- Utilizar robots rígidos de baja inercia como interfaz entre el usuario y el ambiente virtual, mismos que generen las fuerzas presentes durante la interacción entre el usuario y los objetos virtuales.
- Aprovechar los diversos conocimientos adquiridos tanto de computación gráfica, robótica y teoría del control con el fin de obtener aplicaciones funcionales que permitan al usuario tener un primer acercamiento con sistemas de realidad virtual con interacción háptica.
- Crear un puente sólido entre los entornos hápticos y gráficos con el fin de desarrollar sistemas que proporcionen una interacción en donde el usuario experimente a través del tacto lo que está visualizando.

1.2. Motivación

En el Laboratorio de Robótica del Departamento de Control de la División de Ingeniería Eléctrica de la Facultad de Ingeniería de la UNAM se ha trabajado anteriormente de manera independiente con los robots *Novint Falcon* y *Geomagic Touch*. Con dichos trabajos se han obtenido resultados que van desde el análisis matemático detallado de cada una de sus arquitecturas hasta la creación de que estuvieron enfocadas primordialmente en el funcionamiento de los algoritmos de control utilizados en cada robot. Sin embargo se dejó de lado la parte gráfica debido a que no se estableció como objetivo el desarrollo de un sistema gráfico complejo. Por lo tanto este trabajo busca complementar dichos proyectos con aplicaciones gráficas mucho más detalladas y específicas, logrando así obtener una interacción mucho más realista e inmersiva.

1.3. Estado del arte

Retomando las preguntas planteadas al inicio de este capítulo es probable que el lector se llegue a imaginar una infinidad de conceptos relacionados con la RV, que pueden ser desde videojuegos, películas de ciencia ficción, simulaciones gráficas, atracciones de

entretenimiento o hasta todo un mundo ficticio en donde se es alguien completamente diferente, por lo tanto, es importante aclarar algunas ideas y plantear nuevas preguntas que al responderlas profundicen el conocimiento acerca del tema: ¿Cuáles fueron los inicios de la Realidad Virtual? y ¿cómo ha sido su evolución hasta nuestros días?

1.3.1. Realidad virtual

Es un tanto difícil establecer una definición concisa debido a la complejidad y el alcance que se desee tener en las aplicaciones que utilicen esta herramienta tecnológica, un claro ejemplo es el que se muestra en la Figura 1.2 en donde se observa a una persona que busca interactuar de una manera más inmersiva en la lectura que está realizando. Por otro lado se encuentran las conceptualizaciones que varios autores han establecido a lo largo de la historia, por ejemplo Jaron Lainer, a quién se le atribuye la definición actual, propone que la RV es un conjunto de técnicas y tecnologías basadas en ordenador que aproximan la visualización de conceptos, objetos y acciones en tres dimensiones de forma interactiva buscando que se asemeje a la realidad [14].

Iván Sutherland, quién es considerado uno de los grandes pioneros de la RV, establece que lo importante no es encontrar una definición exacta sino entender el concepto y sus posibilidades, pudiéndose resumir como la simulación de un mundo virtual con el que se pueda interactuar y que a su vez permita resolver un problema conciso de simulación avanzada [9].

Figura 1.2: Representación de un sistema de realidad virtual [15].

Tomando en cuenta las anteriores propuestas, se ha podido establecer una ruta en el desarrollo de esta disciplina tecnológica ya que en la actualidad las aplicaciones de RV y los dispositivos de interacción son diversos. Sin embargo, el proceso evolutivo que se ha tenido para llegar a obtener dichos avances no fue del todo fácil, pues en un principio no se tenía contemplado un alcance tan grande como el que se ha dado. Por lo tanto es pertinente recapitular de manera breve la historia de la Realidad Virtual y el impacto que ha dejado a través del tiempo. Uno de los primeros pasos se remonta a 1950. En ese año el inventor Morton Heilig, considerado hoy en día como el padre de la RV, vio en el teatro la oportunidad de estimular todos los sentidos del espectador de una forma mucho más eficaz, desarrollando así uno de los primeros dispositivos con tecnología multisensorial: el Sensorama que se puede ver en la Figura 1.3. Este era un dispositivo mecánico capaz de mostrar un cortometraje y en donde se podía tener una interacción mayor a la convencional puesto que el usuario se exponía a una estimulación que incluía sonido, sensación de viento, aromas, visión en 3D y vibración.

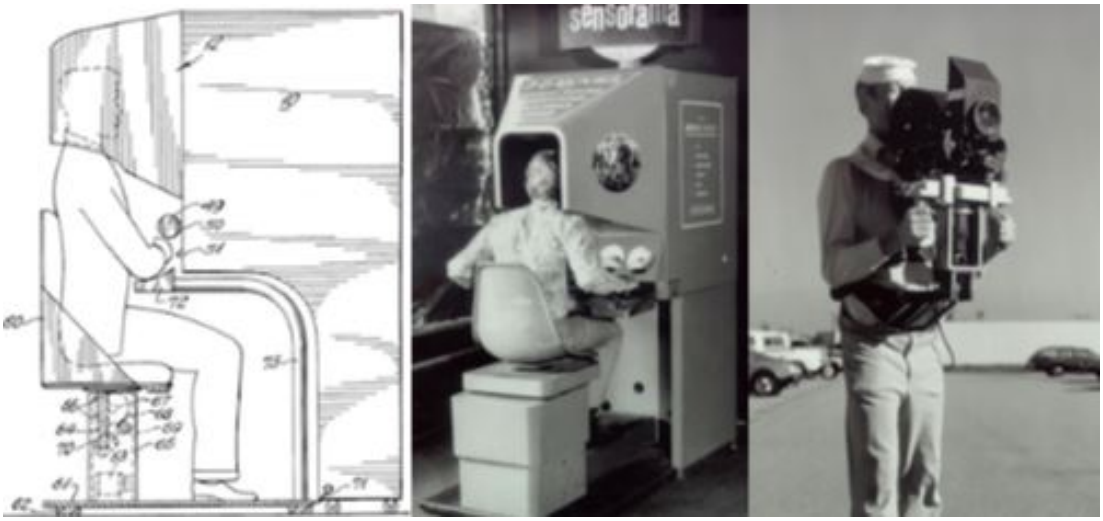


Figura 1.3: Morton Heilig y el desarrollo del dispositivo Sensorama [15].

Años más tarde en 1965, Iván Sutherland plasmó en el artículo *The Ultimate Display* su visión de un mundo virtual en el cual establece que la interacción no solamente debería ser visual o auditiva sino también táctil, convirtiéndose así en uno de los primeros precursores de la interacción háptica. Éste no fue su único aporte hacia la RV ya que tres años después en 1968, junto con su alumno Robert Sproull, desarrollaron el primer sistema de visualización montado en la cabeza del usuario el *Head Mounted Display HMD (HMD)* conocido también como la espada de Damocles y que se puede observar en la Figura 1.4. Dicho sistema usaba tubos de rayos catódicos similares a los de una televisión y presentaba por separado imágenes para cada ojo junto con una interfaz mecánica y ultrasónica.

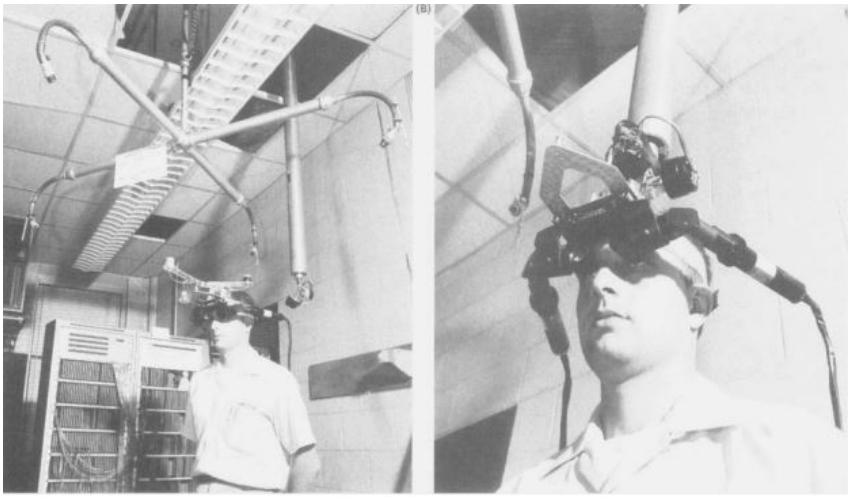


Figura 1.4: Ivan Southerland probando la espada de Damocles [15].

Posteriormente en la década de los setenta se construyó uno de lo grandes pilares de la RV: el desarrollo y lanzamiento de las primeras consolas de videojuegos. La *Magnavox Odyssey* y la *Atari Pong* que se pueden ver en las Figuras 1.5 y 1.6 respectivamente revolucionaron la historia, no sólo del entretenimiento, sino de la tecnología al crear una aplicación que simulaba una partida de ping pong. Este avance se puede considerar la piedra angular de un nuevo camino que hoy en día ha generado desarrollos inimaginables, en los cuáles prácticamente se puede sentir que uno es parte del juego.

Figura 1.5: *Magnavox Odyssey*, primera consola de videojuegos [15].

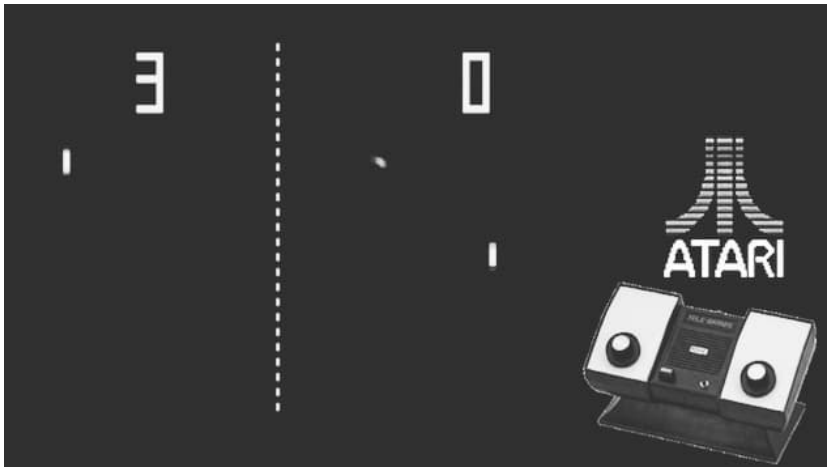


Figura 1.6: *Atari Pong*, segunda consola de videojuegos [15].

Al paso de los años el término RV se fue popularizando cada vez más, por lo que se empezaron a fundar las primeras compañías dedicadas al desarrollo de dispositivos que permitieran tener una mejor interacción con los sistemas virtuales. Una de ellas fue *VPL Research* fundada por Jaron Lanier en 1984, en donde se crearon diversos dispositivos como *The DataGloves*, *The EyePhone* o *The DataSuite*, mismos que se pueden visualizar en la Figura 1.7. El objetivo de estos desarrollos era poder utilizarlos ya fuese en el ámbito médico, simuladores de vuelo, diseño automovilístico o con fines militares.

Figura 1.7: Dispositivos de realidad virtual desarrollados por VPL Research [15].

En el año de 1991 la RV tuvo uno de los cambios más grandes de su historia; dejando un poco de lado el paradigma de un sistema montado en el usuario, científicos de la Universidad de Illinois en Chicago desarrollaron el sistema inmersivo CAVE (*Cave Assisted Virtual Environmen*) mostrado en la Figura 1.8. Este tenía la forma de una sala o habitación en donde se proyectaban imágenes con las que se podía interactuar

haciendo uso de gafas de tercera dimensión. La sala contaba con suficiente espacio por lo cual el usuario podía recorrerla sin ningún problema, logrando así tener una mejor percepción.

Figura 1.8: Sistema CAVE desarrollado por la Universidad de Illinois [15].

Ya en el nuevo milenio la RV se ha catapultado como una de las herramientas tecnológicas de mayor importancia, estableciendo pauta y tendencia dentro de los ámbito científico, tecnológico y económico. Un claro ejemplo es la compañía *Oculus VR*, una empresa fundada por Palmer Luckey a principios del año 2010, la cual desarrolló *Oculus Rift* un HMD que buscaba ser más efectivo y económico para el público en general y que se muestra en la Figura 1.9. El dispositivo resultó ser un rotundo éxito y su impacto fue tal que la compañía fue comprada por Facebook, demostrando así que la RV se ha vuelto un campo totalmente rentable y una de las tecnologías fundamentales para el futuro.

Figura 1.9: Dispositivo HMD Oculus Rift diseñado por Oculus VR [3].

1.3.2. Características de un sistema de RV

Todos los sistemas de RV buscan cumplir con tres características que se relacionan entre sí tal como se muestra en la Figura 1.10. Y que se enlistan a continuación [15]:

- La posibilidad de estar en tiempo real, permitiendo así controlar la dirección del movimiento dentro un escenario virtual.
- La inmersión determina el porcentaje o nivel de percepción que se está teniendo con los estímulos virtuales ya que se define como la sensación de estar presente en un ambiente.
- La interacción con el usuario, la cual se da gracias a los elementos de hardware o periféricos de entrada y salida que componen al sistema y que son los encargados de estimular nuestros sentidos.

De igual forma se necesita una estación de trabajo o computadora especializada que cuente con los elementos fundamentales que permitan la creación de aplicaciones: un procesador que interprete todas las instrucciones a ejecutar durante el desarrollo, una tarjeta gráfica adecuada que permita procesar dichas tareas y que cuente con una unidad de procesamiento gráfico (GPU) que aligere la carga del procesador central del equipo. Así mismo, se debe contar con elementos de software que modelen gráficamente los objetos del entorno real en 3D, generando así una simulación con la cual se pueda interactuar.

Figura 1.10: Características fundamentales de un sistema de RV [13].

Dicha estructura está configurada con el fin de, no sólo realizar estimulaciones sensoriales como la visual, auditiva o táctil, sino también simulaciones físicas en las cuales se puedan detectar los movimientos de la cámara, colisiones con otros objetos y en algunos casos, cálculo de deformaciones.

1.3.3. Funciones básicas de un sistema de RV

Todo sistema de este tipo busca cumplir las siguientes funciones:

- Calcular las estructuras de datos, texturizados si es el caso, luces y sombreados de todos los componentes virtuales del sistema.
- Mantener seguimiento de cada uno de los objetos virtuales.
- Almacenar y actualizar los datos sobre la localización y aspecto de cada objeto.
- Simular el comportamiento de los cuerpos.
- Renderizar el ambiente virtual en tres dimensiones.
- Permitir al usuario navegar a través del entorno virtual.
- En algunos casos poder generar los sonidos correspondientes a cada objeto virtual.
- Proporcionar al usuario los medios para interactuar con el entorno virtual.
- Realizar dicha interacción en tiempo real, es decir, que se cuente con un lapso de tiempo determinado para ejecutar dicha acción o respuesta.

Al ejecutar correctamente los puntos anteriores se obtiene una aplicación eficaz que cumpla con los parámetros establecidos para un sistema de este tipo.

1.3.4. Tipos de sistemas de RV

La inclusión de los elementos y funcionalidades de los sistemas de RV dependerán del enfoque al cual estén orientados, así como el grado de inmersión y la forma en que el usuario interactúe con ellos. Así mismo dichos parámetros permiten clasificarlos en tres tipos: inmersivo, semi-inmersivo y no inmersivo, los cuales se describirán detalladamente a continuación.

- Sistemas inmersivos: Buscan que el usuario se adentre totalmente al mundo virtual que se está simulando, para ello se hace uso de diversos dispositivos como visores, guantes, cascos y trajes especiales. De esta manera logran un grado de inmersión mayor pero no total ya que es un nivel muy difícil de alcanzar debido a la multitud de sentidos involucrados, aunque si lo considerable para obtener una interacción mucho más realista en la que se desconecta al operador del mundo real tal como se visualiza en la Figura 1.11. Este tipo de sistemas es usualmente usado para aplicaciones de entrenamiento o capacitación [14].
- Sistemas semi-inmersivos: Conocidos también como sistemas de proyección inmersiva, una de sus principales características es la de contar con cuatro paredes en forma de cubo las cuales simulan una habitación en la que se proyectan distintos gráficos con los que el usuario puede interactuar, de forma parecida al sistema

CAVE. De igual forma se utilizan distintos dispositivos que permiten interactuar con el ambiente virtual pero a diferencia de los sistemas anteriores, el grado de inmersión es menor [14], por lo tanto el usuario aún mantiene contacto con el entorno real tal como se puede visualizar en la Figura 1.12.

Figura 1.11: Sistema de realidad virtual inmersivo realizado por la NASA [14].

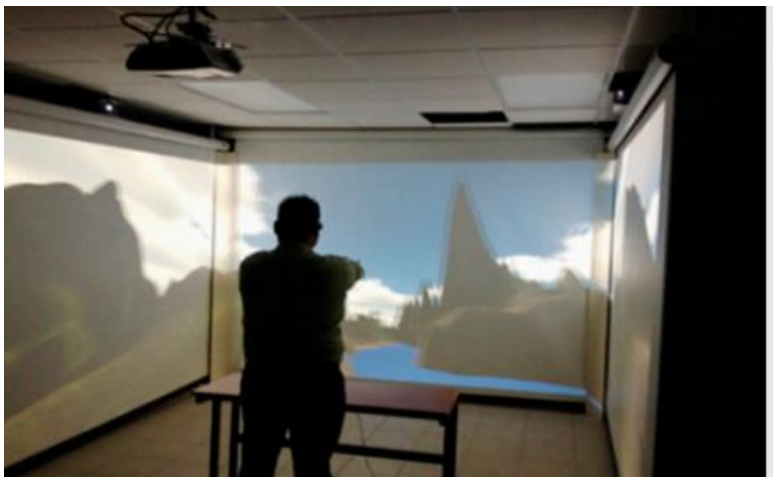


Figura 1.12: Sistema semi inmersivo que representa un recorrido virtual del Instituto Tecnológico Superior de Zapotlanejo, Jalisco [16].

- Sistemas no inmersivos: Este tipo de sistemas, a comparación de los anteriores

suelen ser más simples debido a que su visualización se presenta por medio de una pantalla o monitor [14]. De igual manera los dispositivos utilizados para la interacción llegan a ser un tanto básicos como un *mouse* o un *joystick* hasta algunos dispositivos hápticos. A diferencia de los citados previamente, no le presentan al usuario un alto nivel de inmersión tal como se puede ver en la Figura 1.13.

Figura 1.13: Sistema no inmersivo con interacción háptica del centro médico Beth Israel Deaconess [13].

1.3.5. Háptica

Así como la acústica y la óptica estudian los fenómenos que involucran el sonido y la luz en los cuales se tiene una estimulación los sentidos del oído y la vista respectivamente, la háptica es la disciplina que se encarga del estudio de las diversas sensaciones físicas que los seres humanos experimentan al hacer uso del tacto. Siendo este un sentido esencial para el desarrollo de la humanidad pues es el único que permite percibir diversos estímulos táctiles, generando así un intercambio de información mucho mayor con el entorno que los rodea [12].

Actualmente la háptica está jugando un papel trascendental pues, tal como se mencionó anteriormente, junto con la RV está revolucionando la forma de interactuar con la tecnología. Sin embargo no es una herramienta nueva ya que sus inicios se dieron al principio de los años noventa, cuando se buscó evolucionar las interfaces o dispositivos de comunicación entre usuarios y equipos de computo, concentrándose en obtener una manipulación y percepción más cercana e interactiva. Tal y como se muestra en la Figura 1.14, estos instrumentos pueden clasificarse en función del tipo de actuadores que utilicen así como en la escala de fuerzas de reacción o bien, en su portabilidad y soporte, pudiendo ser interfaces de escritorio, fijas o portátiles.

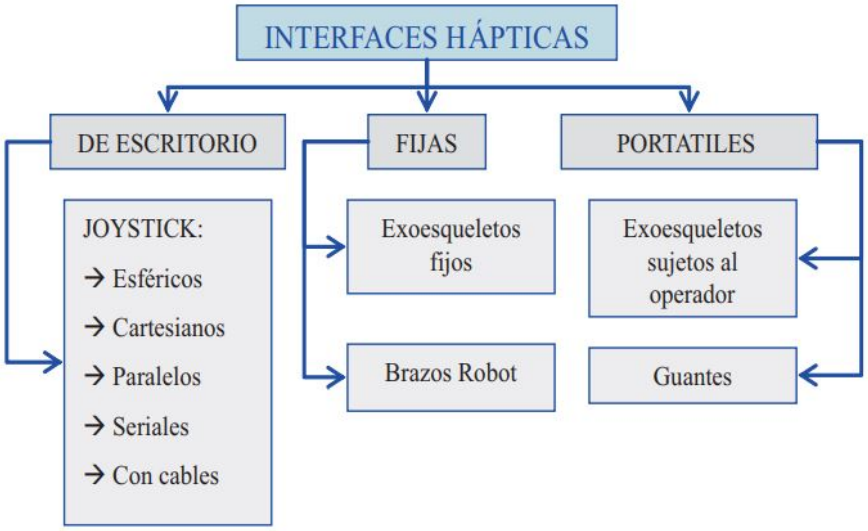


Figura 1.14: Clasificación de dispositivos hápticos según su portabilidad [12].

1.3.6. Arquitectura de un sistema de realidad virtual con interacción háptica

Para que un sistema de realidad virtual pueda establecer una interacción háptica tal como se muestra en el diagrama de bloques de la Figura 1.15, es indispensable conocer su arquitectura así como las etapas y elementos que lo conforman y los cuales son:

- El usuario, que es el operador del sistema.
- El dispositivo háptico, el cual permitirá la interacción táctil con el sistema.
- La renderización háptica, que es la responsable de generar y validar la fuerza de contacto así como el motor de simulación que tiene como objetivo computar el entorno virtual.
- La renderización visual, que es la encargada de crear los objetos gráficos con los que se está interactuado y finalmente, el dispositivo de video que es el receptor de salida visual del sistema.

En esta arquitectura se puede dar una comunicación unidireccional o bidireccional según los elementos que interactúen, lo que se indica en el diagrama mediante las flechas simples o dobles respectivamente.

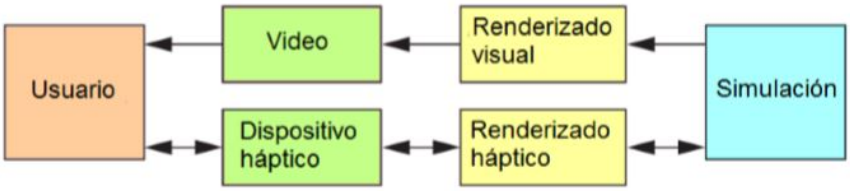


Figura 1.15: Arquitectura de un sistema de RV con interacción háptica [1].

1.4. Contribuciones

Las principales contribuciones de este trabajo de tesis son:

- El desarrollo y puesta en marcha de un sistema de RV compatible con los robots *Novint Falcon* y *Geomagic Touch* utilizando la herramienta gráfica *OpenGL* y en donde el usuario pueda interactuar visual y hápticamente con objetos rígidos.
- Un ambiente virtual en donde se lleve a cabo la interacción háptica con objetos deformables utilizando el robot *Geomagic Touch*.
- Elaboración de la documentación en forma de guía en la que se muestre la metodología utilizada para el desarrollo de cada uno de los entornos virtuales, con el objetivo que se pueda aprovechar para el desarrollo de futuros sistemas de mayor complejidad.

1.5. Estructura de la tesis

Este trabajo se encuentra dividido en cinco capítulos. En el Capítulo 2 se presentan detalladamente las diversas herramientas y técnicas matemáticas utilizadas para obtener cada una de las etapas de los sistemas implementados. El Capítulo 3 se centra específicamente en las características de los robots *Geomagic Touch* y *Novint Falcon* los cuales fueron utilizados como interfaces hápticas en este trabajo. A su vez, el Capítulo 4 se enfoca en la implementación y análisis de resultados de los sistemas propuestos. Por último, el Capítulo 5 muestra las conclusiones de la tesis, así como algunas propuestas sobre trabajo futuro. Adicionalmente en el anexo se encuentran los apéndices A y B, los cuales servirán como manuales de guía para la correcta configuración de las herramientas utilizadas en el desarrollo de los sistemas de realidad virtual.

Capítulo 2

Preliminares

En este capítulo se indicarán los conceptos, ecuaciones y métodos necesarios para el desarrollo de cada una de las etapas que conforman un sistema de realidad virtual con interacción háptica. De manera muy general dichas etapas pueden enumerarse de la siguiente manera:

1. *Renderizado visual*: Es la parte encargada de generar una representación gráfica o imágen de un cuerpo a partir de un modelo en 2D o 3D. Para ello se necesita un motor gráfico, que es el responsable de la eficacia que pueda alcanzar dicha representación.

2. *Renderizado háptico*: Se encarga de enviar al usuario las fuerzas de interacción generadas en el sistema. Incluye las etapas de *detección de colisiones* y *renderizado de fuerzas*, las cuales permiten determinar si existe contacto con algún objeto del sistema virtual así como la generación de la fuerza al momento del contacto.

3. *Simulación*: Esta etapa es la responsable de recrear las fuerzas establecidas en el renderizado háptico, así como vincularlas correctamente con los gráficos creados en el renderizado visual. Para lograrlo cuenta con un motor de simulación que tiene como objetivo realizar el proceso que permite la unión de las etapas anteriores.

4. *Dispositivos hápticos*: Son los dispositivos de entrada-salida encargados de permitir la interacción táctil entre el usuario y el sistema.

5. *Video*: Engloba los diferentes dispositivos visuales de salida con los que puede contar el sistema. En ellos el usuario podrá apreciar el resultado del renderizado visual obtenido.

Durante el desarrollo del presente trabajo de tesis fue necesario conocer a detalle la relación que estas etapas tienen entre sí, así como la arquitectura que conforman al integrarlas en un solo sistema. Para ello se tomó como apoyo el diagrama de bloques que se muestra en la Figura 2.1, en donde se pueden observar las fases que componen su diseño.

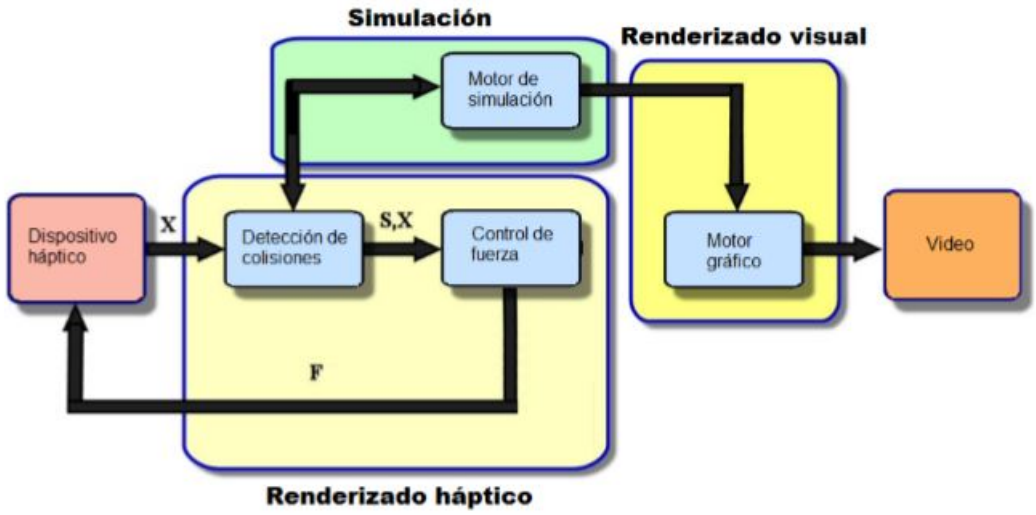


Figura 2.1: Esquema de un sistema virtual con renderizado háptico [1].

Tal como se indica en el diagrama, las partes fundamentales son *el renderizado visual*, *el renderizado háptico* y *la simulación*, las cuales se describirán a profundidad en este capítulo.

2.1. Renderizado visual

Mientras que la renderización háptica se encarga de suministrar las fuerzas que simulan el contacto con un objeto, el renderizado visual es el responsable de generar una representación gráfica o imagen de un cuerpo a partir de un modelo en 2D o 3D. Este proceso suele ser complejo, tal como se puede ver en la Figura 2.2, ya que cuenta con varias etapas que van desde establecer la geometría básica que conforma a cada objeto, pasando después por una fase en la que se le agregan diferentes características tales como el comportamiento de luces, el manejo de colores y texturas, pudiendo agregar también animaciones y sonidos dependiendo el grado de realismo que se quiera alcanzar.

Para lograr el proceso mencionado se debe contar con un motor gráfico que permita ejecutar con éxito cada una de estas tareas, por lo que es necesario conocer a fondo cada una de las etapas antes mencionadas siendo el objetivo final contar con una simulación en donde se pueda visualizar alguna estructura o ambiente real.

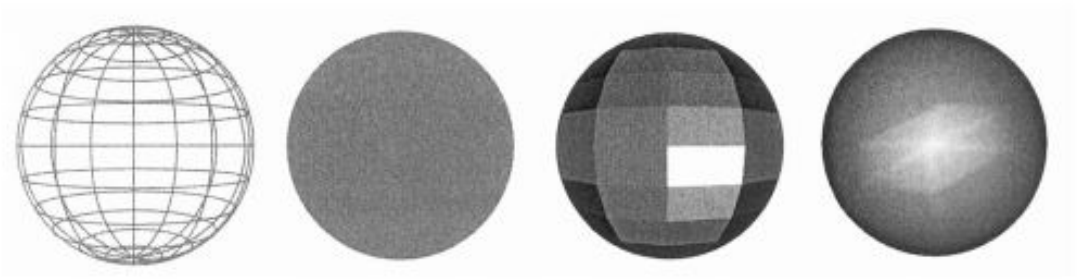


Figura 2.2: Modos de visualización en malla de alambre, iluminación plana y suavizado. [6].

2.1.1. Motor gráfico

Conocido también como motor de videojuegos o *game engine*, es la herramienta responsable de crear y desarrollar una representación de algún objeto. Se trata de un espacio de trabajo o *framework* usado por algunos programas para dibujar gráficos en la pantalla. Es utilizado para desarrollar aplicaciones que pueden ser desde videojuegos para consolas y dispositivos móviles hasta simuladores de realidad virtual.

El término *game engine* tiene origen a mediados de los años 90 gracias al lanzamiento del juego *Doom* perteneciente a la empresa *Id Software* ya que en él se separaron por primera vez el sistema encargado de los gráficos y el responsable de las interacciones físicas. Sin embargo las ventajas de esta separación se hicieron evidentes en entregas posteriores en las que se pudieron reutilizar componentes, ejemplos de dichos desarrollos fueron *Quake III* y *Unreal*. Teniendo en cuenta lo anterior, se puede decir que la funcionalidad básica de un motor gráfico es proporcionar tanto un motor de renderizado para gráficos en 2D y 3D como un motor que detecte las colisiones físicas entre objetos, sonidos, música, animación, inteligencia artificial, más específico en el entorno de videojuegos, la comunicación de red para juegos multijugador, la posibilidad de ejecución en hilos y la gestión de memoria o soporte para localización [5].

Las capacidades gráficas que tengan los motores son una de las claves para su elección, de igual forma es importante tener en cuenta la facilidad de desarrollo y la plataforma en la que se van a utilizar. Hoy en día se cuenta con una gran cantidad de opciones para elegir como son *CryEngine*, *Unity*, *Unreal Engine* y *Sony PhyreEngine*, los cuales han sido utilizados para desarrollar algunas de las aplicaciones y juegos más innovadores de la actualidad.

2.1.2. Modelado geométrico

Retomando lo citado con anterioridad, el modelado de figuras es uno de los pilares en el proceso de renderizado gráfico y sin duda es uno de los más complejos, ya que se trata de un proceso en el que se busca crear una representación gráfica de una enti-

dad abstracta por medio de la computadora. Estos modelos pueden ser clasificados en diferentes tipos como son: *modelos cuantitativos*, que utilizan una ecuación para representar y describir su comportamiento, *modelos organizacionales*, que usan jerarquías para representar sus esquemas de clasificación y *modelos CG*, los cuales se refieren a la representación geométrica de una entidad o cuerpo, permitiendo así visualizar la estructura en que se están modelando dichas entidades [6].

Un modelo puede ser creado tanto en 2D como en 3D tal como se muestra en la Figura 2.3. Sin embargo se limitan muchas características al modelar un objeto en dos dimensiones ya que se tiene una visualización plana a comparación de un modelado en tercera dimensión en donde se permite mostrar características fundamentales de un objeto real, como lo es su volumen. Por esta razón en el presente trabajo se utilizaron objetos 3D para representar de una forma más realista los cuerpos en el entrono virtual. Dicho lo anterior, es necesario tener en cuenta el sistema tridimensional en el que se realizará el modelado así como el tipo de objetos a crear. En este caso fueron implementados por medio de primitivas y polígonos. Todas estas características se detallan a continuación.

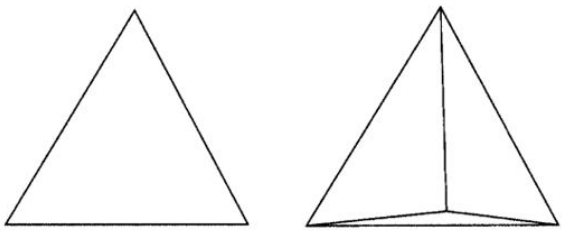


Figura 2.3: Modelado de figuras en 2D y 3D [6].

- *Sistema coordenado en tres dimensiones*

  En el mundo real todos los cuerpos son visualizados en tres dimensiones por lo que las simulaciones virtuales buscan recrear un espacio tridimensional por lo tanto el marco de referencia debe ser presentado como un sistema coordenado de tres ejes, en este caso el sistema XYZ tal como el que se puede ver en la Figura 2.4.

  Es pertinente indicar que el eje Z es perpendicular a los otros dos, siendo ortogonal a ellos. De igual forma cada eje está conformado por un lado positivo y uno negativo, mismos que se unen en un único punto denominado origen y que tiene coordenadas (0,0,0). Este punto marca la referencia al momento de crear y localizar cada uno de los objetos presentes en una escena ya que es a través de puntos coordenados como se establece la posición que tendrá cada uno de los objetos dentro del ambiente virtual.

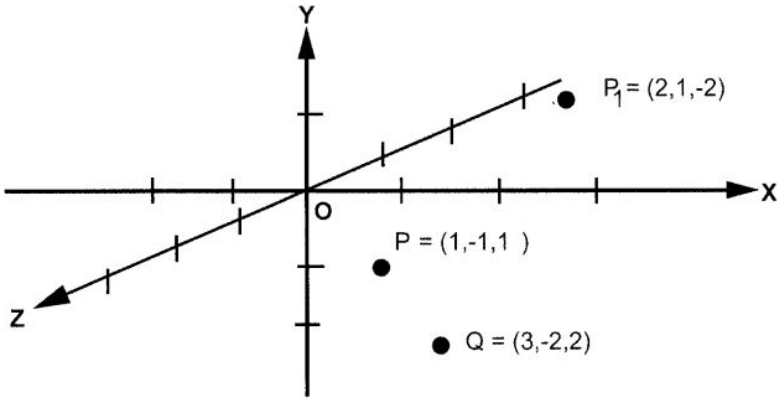


Figura 2.4: Sistema de ejes coordenados XYZ [6].

- *Figuras primitivas*

  Son entidades que cuentan con una fórmula o ecuación matemática que las define, por lo que son denominadas también superficies implícitas. Figuras como cubos, esferas y conos son algunos ejemplos de primitivas, lo que resulta ser algo muy conveniente al momento del modelado gráfico ya que la mayoría de los objetos suelen estar conformados por algunas de ellas. En el desarrollo realizado en esta tesis se utilizaron dos figuras primitivas las cuales fueron el cubo y la esfera.

  - *Cubo.* Conocido también como caja, es una de las figuras más comunes que se pueden encontrar en el mundo real. Este modelo se puede generar en tres dimensiones de una manera sencilla, tal como se puede ver en la Figura 2.5 ya que, si se cuenta con el centro del cubo usado como base, es posible encontrar tanto la longitud como su altura y profundidad.
  - *Esfera.* Siendo la figura más simétrica que se puede encontrar en la naturaleza, resulta ser las más simple al momento de modelar ya que sólo se necesita el valor de su radio R y la localización de su centro. Si dicho punto se encuentra en el origen del plano coordenado entonces cualquier punto P de su superficie puede ser definido por:

    $$P = (R\cos\theta\cos\phi, R\sin\theta, R\cos\theta\sin\phi) \tag{2.1}$$

    en donde los ángulos θ y ϕ son los responsables de delimitar el espacio que la esfera llegue a ocupar en el sistema coordenado tal como se observa en la Figura 2.6.

- *Polígonos de mallas.* Uno de los métodos más populares y comúnmente usados es el modelado a través del uso de pequeños polígonos los que al unirse buscan aproximarse en su totalidad a una superficie deseada. De esta forma cada polígono se puede considerar como una de las piezas que forman parte de la estructura de

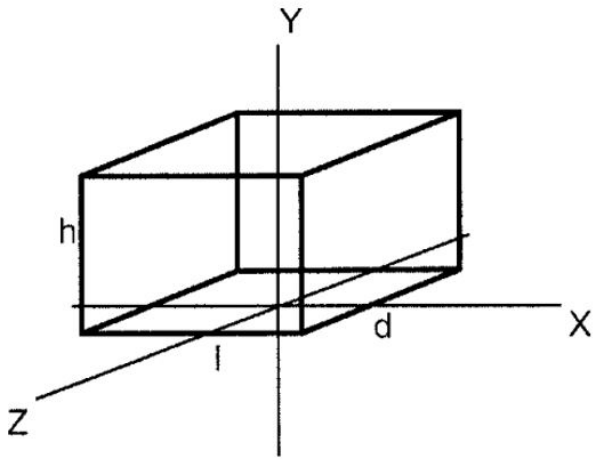


Figura 2.5: Cubo definido en tres dimensiones [6].

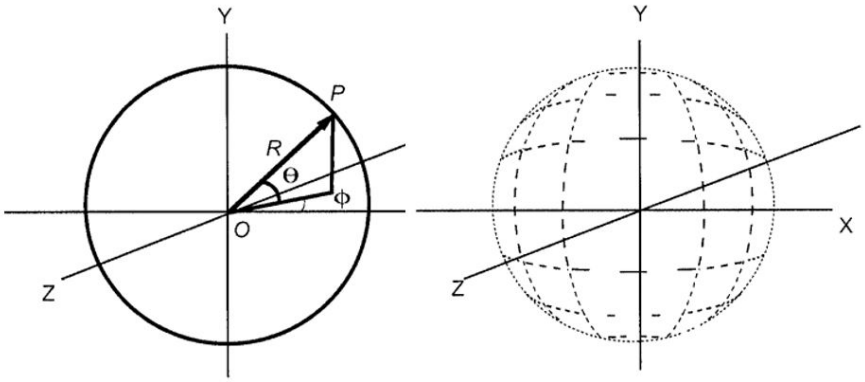


Figura 2.6: Modelado de esfera en tres dimensiones [6].

un objeto. Para poder dibujar cada uno de ellos primero se deben definir los vértices que los conforman. Posteriormente son conectados a través de líneas rectas denominadas bordes las cuales no deben cruzarse entre sí ya que buscan crear un área simple y convexa tal como se muestra en la Figura 2.7. Se debe tener en cuenta que se necesitan por lo menos tres puntos para crear una de estas figuras.

Un polígono por sí mismo cuenta con limitaciones al momento de ser usado en el modelado de objetos. Sin embargo tal como se comentó anteriormente la verdadera ventaja de estas figuras es utilizarlas en conjunto, formando una estructura conocida como malla, la cual permite generar figuras más complejas. Para crear dichas mallas cada polígono debe compartir uno de sus bordes por lo menos con

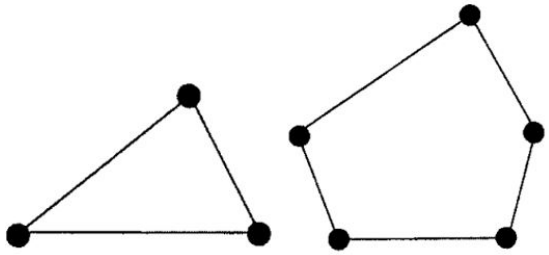


Figura 2.7: Áreas convexas conformadas por polígonos [6].

otros dos, tal como se muestra en la Figura 2.8. Se puede decir que un modelo de este tipo se puede representar visualmente como un marco de alambre o *wireframe* al cual se le puede agregar distintas características como son color, luces y en ciertos casos texturas. Con todo ello se llega a obtener como resultado el renderizado de un modelo sólido tal como se puede visualizar en la Figura 2.2 de la Sección 2.1.

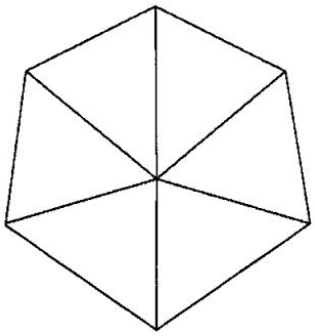


Figura 2.8: Estructura tipo malla triangular [6].

2.1.3. Color

En el proceso de renderización la adición de color permite que los objetos virtuales eleven su parecido con sus contrapartes del mundo real. Por lo tanto es de suma importancia tener en cuenta la metodología que permita representar los diferentes colores a utilizar en un sistema virtual. Existen diversos métodos que cumplen esta funcionalidad como son el *CMYK*, el *HSL* o el *RGB* siendo este último el utilizado en la presente tesis. Se le denomina RGB debido a que utiliza como base tres colores primarios rojo (*red*), verde (*green*) y azul (*blue*) los cuales interactúan entre sí para conformar otros colores tal como se puede observar en la Figura 2.9. Este modelo suele ser representado por medio de un vector de tres dimensiones en donde cada elemento equivale a uno de los colores ya antes mencionados. Para indicar la intensidad que tendrá cada color al momento de generar uno nuevo se utilizan dos escalas, la primera tiene como valor

Color	Red	Green	Blue
Amarillo	1	1	0
Cyan	0	1	1
Mageta	1	0	1
Blanco	1	1	1
Negro	0	0	0

Tabla 2.1: Generación de colores utilizando el modelo RGB - Ejemplos de combinaciones para obtener diferentes colores utilizando números flotantes en el modelo de color RGB.

mínimo el 0 y como valor máximo 255, mientras tanto la segunda que es la comúnmente utilizada tiene como parámetros números flotantes que van del 0 al 1 tal como se puede ver en la Tabla 2.1.

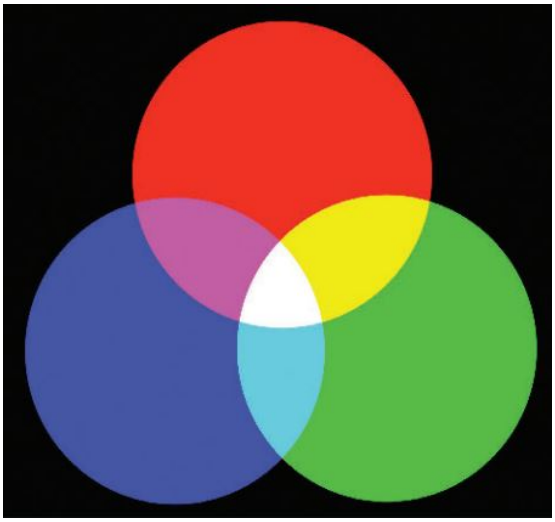


Figura 2.9: Modelo de color RGB [6].

2.1.4. Iluminación

Uno de los aspectos más importantes en el proceso de renderización visual es sin duda la iluminación ya que un mismo objeto puede apreciarse de un tono diferente dependiendo del tipo de luz con el que esté interactuando. Por lo tanto es necesario

tener en cuenta la forma en que los cuerpos son iluminados. Este proceso es posible gracias a la *ley de reflexión* la cual describe el cambio en la dirección de un rayo o haz de luz que puede ser representado como un vector I cuando viaja a través del espacio e impacta con alguna superficie. Una vez hecho el contacto se produce la reflexión y se genera un nuevo vector R el cual termina desviándose y regresando a su fuente de procedencia. En este proceso se crea un ángulo de incidencia que resulta ser igual al ángulo formado por el rayo I al momento del contacto tal como se puede ver en la Figura 2.10. Cabe destacar que las propiedades de la superficie de los objetos determinan la manera en que la luz entrante será reflejada.

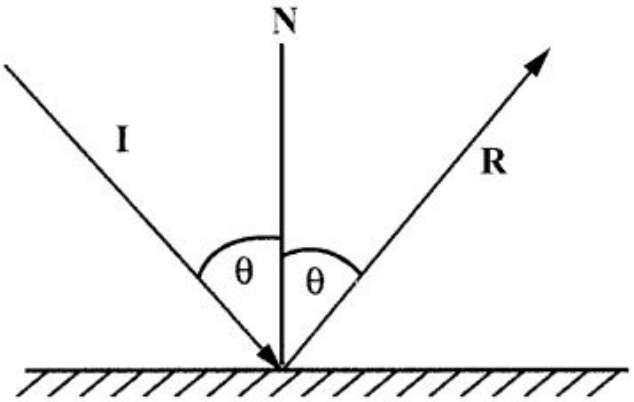


Figura 2.10: Ley de reflexión [6].

En Computación Gráfica se intenta emular el modelo mencionado por lo que se utilizan métodos que permitan aproximarlo. Entre ellos se destaca el modelo de reflectancia propuesto por Bui Tuong Phong [6] el cual se divide en cuatro componentes:

- *Reflectancia ambiental.* Es usado para evitar que las escenas se vuelvan completamente negras. La luz ambiental no tiene dirección; choca por igual en todas las superficies y direcciones además de que es reflejada por un múltiplo constante, dando como resultado superficies de aspecto plano.
- *Reflectancia difusa.* Al igual que la componente anterior puede reflejarse en todas las direcciones, sin embargo su intensidad varía en función del ángulo incidente.
- *Reflectancia especular.* Permite observar reflexiones en cualquier superficie brillante, causando en ellas un brillo gris o blanco opaco.
- *Emisión.* Se utiliza para simular objetos luminosos que parecen brillar por sí mismos.

2.1.5. Texturización

En ciertas ocasiones se busca obtener un realismo mucho mayor en los sistemas virtuales por lo que es necesario alterar drásticamente las características de la superficie de los objetos simulados, para lograrlo se utiliza un proceso conocido como *mapeo de texturas*. Por ejemplo, si se le agrega una imagen de ladrillos a un polígono rectangular éste obtiene un alto grado de vitalidad ya que da la impresión de ser una pared o muro que ha sido modelada a través de ladrillos individuales tal como se muestra en la Figura 2.11.

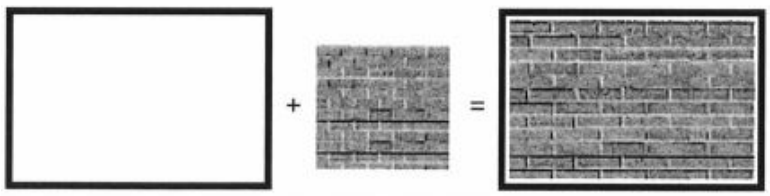


Figura 2.11: Ejemplo del mapeo de texturas [6].

Al asignar una imagen a un objeto, el color de cada pixel que lo conforma es modificado por el color correspondiente al de la imagen utilizada que es también conocida como mapa de textura. Este mapa está conformado por un conjunto de elementos individuales denominados *texeles*. Así mismo reside en su propio espacio coordenado (s, t) el cual cuenta con un rango que va de 0 a 1, permitiendo así delimitar un área rectangular en donde se vean reflejados los datos de textura. La forma más simple de realizar dicho mapeo se da al utilizar los puntos coordenados que conforman las esquinas de la superficie de cada elemento a los cuales se les asignan una coordenada en el espacio de texturas utilizando un *esquema de mapeo* predeterminado. De esta manera se define un cuadrilátero en donde se calcula un promedio ponderado de los *texeles* que se encuentran dentro, así como se muestra en la Figura 2.12. Para superficies poligonales las coordenadas de textura se calculan linealmente en los vértices que las definen.

También es necesario recalcar que cuando se desea texturizar un objeto en 3D éste debe reflejar en su superficie los distintos colores que existan en su entorno, volviéndose así más realista. Sin embargo no suelen ser completamente reflectivos ya que el color de la reflexión se modula con su color real, por lo que este proceso se puede lograr utilizando un método llamado *cube enviroment mapping*, el cual ocupa seis componentes en forma de caras cúbicas que especifican un dirección (frente, atrás, arriba, abajo, izquierda, derecha) tal como se muestra en la Figura 2.13. Este conjunto de caras es proyectado gracias a un mapeo plano sobre el objeto, en donde cada cara del cubo se identifica utilizando los vectores normales (nx, ny, nz) de la superficie del cuerpo tal como lo muestra la Figura 2.14.

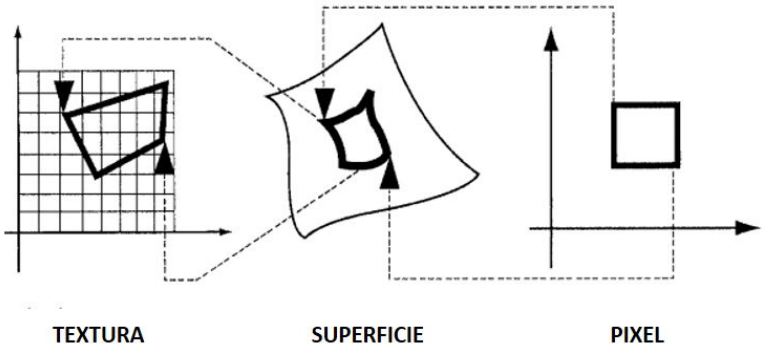


Figura 2.12: Proceso de mapeo de texturas [6].

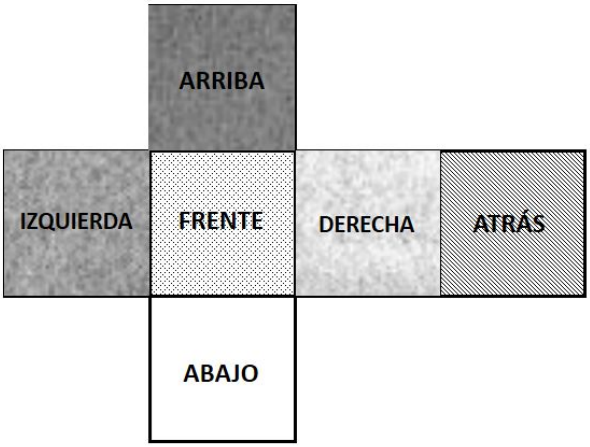


Figura 2.13: Caras del cubo de mapeo [6].

2.2. Renderizado háptico

En esta etapa se maneja la información que fluye entre la interfaz háptica (en el caso del presente trabajo de tesis, los robots *Novint Falcon* y *Geomagic Touch*) y el entrono virtual desarrollado. Tal como se describió anteriormente, es la fase encargada de generar y enviar al usuario las fuerzas de interacción presentes en el sistema.

Uno de los principales aspectos que permiten dicha comunicación es el poder contar con un *avatar*, el cual es una representación virtual de la interfaz táctil que el usuario utiliza para interactuar con el entorno virtual. Este *avatar* dependerá en gran medida de lo que el sistema se encuentre simulando así como de las capacidades del dispositivo háptico. El usuario puede modificar la posición del avatar dentro del entorno virtual. Si éste hace contacto con algún objeto se desencadenarán las fuerzas de reacción en respuesta a los movimientos llevados a cabo. Así mismo su geometría y el tipo de

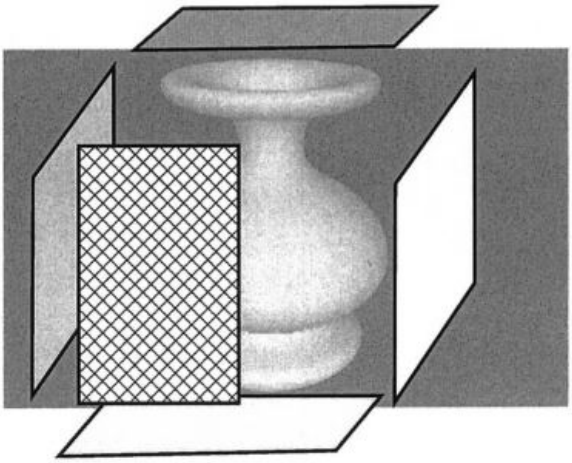


Figura 2.14: Ejemplo de *cube enviroment mapping* [6].

contacto que admita permiten regular el valor de dichas fuerzas, por lo tanto es necesario contar con un avatar dentro de cualquier sistema virtual y sobre todo en donde se utilice interacción háptica, ya que resulta ser el principal responsable de la interacción entre el usuario y las fases siguientes [1].

2.2.1. Detección de colisiones

Un propósito fundamental de los sistemas de realidad virtual es proveer al usuario simulaciones realistas de los objetos con los que esté interactuado. Para lograrlo se busca que sean sistemas dinámicos y físicamente realistas que cuenten con varias características, por ejemplo una serie de cálculos o pruebas de intersección que permitan determinar si el avatar se encuentra en contacto con algún objeto del entorno virtual. Para ello se requiere contar con un algoritmo de detección de colisiones que permita reducir el número de pruebas. Estos algoritmos pueden ser utilizados dependiendo de las características que conformen a los objetos, ya sea el volumen que ocupen dentro del espacio o las formas geométricas por las que estén compuestos, como son pequeños triángulos o planos [10]. A continuación se mostrarán a detalle algunos de los más utilizados.

- *Ray Tracing*

  El objetivo principal de este método es calcular el punto exacto en donde un vector o rayo intersecta con algún objeto de geometría simple como planos, puntos o líneas. Este *ray* parte desde un punto P_0 hasta un objeto con el que puedan existir colisiones. Una vez que el vector atraviesa dicho objeto se calculan los puntos de intersección P_n que existan entre el rayo y sus elementos, tal como se muestra en la Figura 2.15. Obtenidos dichos puntos de intersección se procede a calcular las

distancias existentes entre cada uno de ellos y el punto P_0, identificando así la de menor distancia. Por ejemplo el punto P_1 que se ve en la Figura 2.15 es el que se compara con la posición inicial para determinar si existe o no alguna colisión.

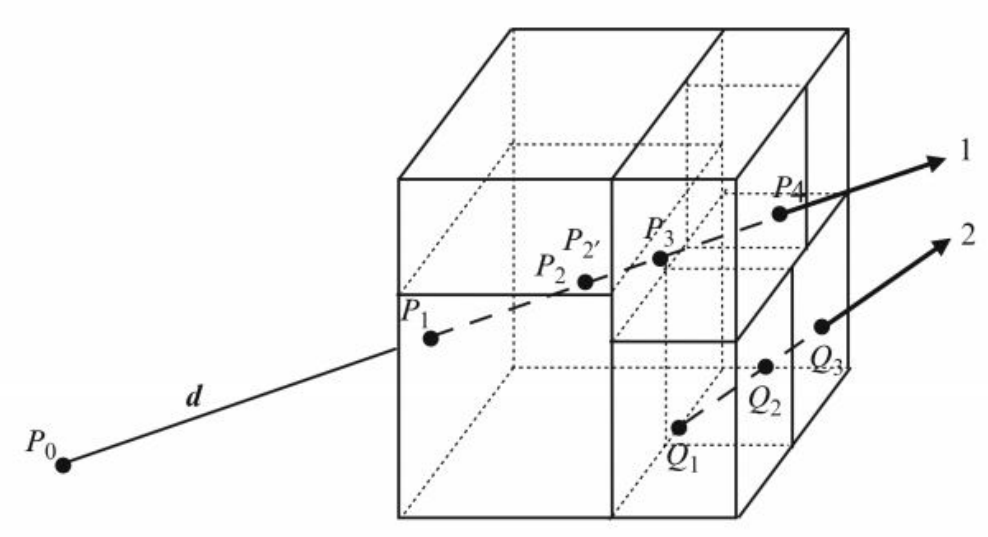


Figura 2.15: Intersección de un rayo con un volumen [10].

Tal como se mencionó anteriormente, los planos son una de las figuras con las que el vector puede intersecar, teniendo como objetivo encontrar un punto P que se encuentre sobre el plano y sobre el rayo. Por lo tanto es importante realizar un análisis vectorial que permita mostrar de una forma clara dicho proceso.

Teniendo en cuenta la ecuación del plano

$$ax + by + cz + d = 0 \tag{2.2}$$

y

$$n = ai + bj + ck \tag{2.3}$$

siendo el vector normal a dicho plano. Y además P es un punto con un vector de posición

$$p = xi + yj + zk \tag{2.4}$$

se puede obtener

$$n \cdot p + d = 0 \tag{2.5}$$

Teniendo además la ecuación del rayo

$$p_v = t + \lambda v \tag{2.6}$$

donde

$$t = x_T i + y_T j + z_T k \tag{2.7}$$

es el vector que proporciona la posición del origen de dicho rayo y

$$v = x_V i + y_V j + z_V k \tag{2.8}$$

su vector de dirección.

Ahora bien, el vector y el plano se deben intersecar para alguna λ tal que

$$n \cdot (t + \lambda v) + d = n \cdot t + \lambda n \cdot v + d = 0 \tag{2.9}$$

por lo tanto,

$$\lambda = \frac{-(n \cdot t + d)}{n \cdot v} \tag{2.10}$$

para el punto de intersección. El vector de posición para P es p_v = t + λv. Si el producto $n \cdot v = 0$ significa que el vector y el plano son paralelos.

- *Superficie implícita*

  Este método es un tanto diferente al anterior ya que, en vez de ocupar un vector que intersecte con algún objeto, utiliza la geometría que lo conforma. Para ello se debe definir una función del tipo *f(x,y,z)*. Por lo tanto, si se desea indicar la existencia de alguna intersección se debe determinar si el avatar se encuentra dentro o fuera de la figura cumpliendo alguna de las siguientes condiciones [19] :

  - Si *f(x,y,z)<0* existe contacto con la superficie.
  - Si *f(x,y,z)=0* se encuentra sobre la superficie
  - Si *f(x,y,z)>0* no existe contacto.

- *Bounding Volumes*

  Conocidos también como algoritmos de volúmenes delimitados. Tienen como meta proporcionar una aproximación del espacio que se encuentre ocupando algún objeto compuesto por mallas con el fin de conocer si se encuentra en contacto con el avatar del sistema. Es necesario recalcar que dicho objeto debe estar contenido en su totalidad dentro del volumen utilizado por el algoritmo. Entre los más comunes se encuentran [10]:

  - *Esfera.* Es la comprobación más sencilla ya que compara la distancia entre los centros de ambas figuras, la cual debe ser mayor que la suma de sus radios. Se puede visualizar en la Figura 2.16-A.

- *AABB (Axis Aligned Boundig Box)*. Utiliza cajas como superficies envolventes que delimitan a los objetos, abarcando todo el espacio que estén ocupando dentro del sistema coordenado. Se puede observar en la Figura 2.16-B.
- *OBB (Object Bounding Box)*. Hace uso de cajas que delimitan a los objetos tomando en cuenta su orientación en el espacio. Se puede visualizar en la Figura 2.16-C.
- *Cascos convexos*. Son utilizados cuando se tiene un conjunto de puntos definidos en dos dimensiones como un único polígono convexo el cual contiene todos sus puntos y vértices.
- *k-DOP (Discrete Oriented Politopes)*. Es utilizado para figuras que se encuentren compuestas por poliedros de k dimensiones.

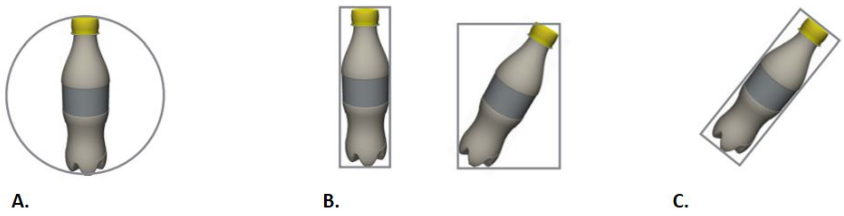


Figura 2.16: Algoritmos de volúmenes envolventes.

2.2.2. Renderizado de fuerzas

Una vez detectada alguna colisión se deben calcular las fuerzas generadas dada la interacción entre el *avatar* y el objeto de contacto. Estas fuerzas buscan aproximarse tanto como sea posible a las que normalmente surgirían durante el contacto entre objetos reales. Por lo tanto, es necesario tener en cuenta los diversos algoritmos de respuesta que permitan conocer dichas fuerzas, ya que algunos operan utilizando las posiciones tanto del avatar como de los objetos virtuales mientras que otros utilizan los valores de las fuerzas ejercidas por el efector final de la interfaz háptica sobre el sistema [1]. Para fines de esta tesis se utilizó un método en el que las posiciones son los parámetros que permiten el cálculo de la fuerza resultante y dado que se busca mostrar una visión general tanto del ámbito de la computación gráfica como del de la teoría de control, a continuación se presentará dicho método visto desde ambas perspectivas.

2.2.2.1. Renderizado de fuerzas desde la perspectiva gráfica

Tal como se mencionó, es necesario hacer una distinción desde el punto de vista de las dos áreas involucradas en este proyecto de tesis. En el caso del área gráfica se debe recalcar que las posiciones del efector final al momento de hacer contacto con el objeto virtual son las responsables de definir la función que permite encontrar el valor

de la fuerza de contacto. Además son sometidas a un proceso de cómputo en donde son utilizadas tanto en la detección de colisiones como en el algoritmo de renderizado de fuerzas denominado *método de penalización* [4], el cual se define a través de la función

$$F(x) = kx + b\dot{x} + m\ddot{x} \tag{2.11}$$

en donde x es la diferencia entre la posición del efector final de la interfaz háptica y el punto de contacto con la superficie virtual. Por otro lado k es una constante de rigidez que permite modelar la fuerza como un resorte de acuerdo a la *Ley de Hooke* [20]

$$F = kx. \tag{2.12}$$

Los extremos de dicho resorte se anclan a un punto del objeto virtual y en la posición del efector final respectivamente.

Adicionalmente b representa una constante de amortiguamiento que permite reducir las vibraciones presentes en el resorte, teniendo como resultado que la fuerza calculada es proporcional a la velocidad del efector final de acuerdo a

$$F = -bv. \tag{2.13}$$

Además, si se conoce la trayectoria de un objeto, se puede calcular la fuerza que se experimentaría durante su movimiento utilizando la Segunda Ley de Newton

$$F = m \cdot a. \tag{2.14}$$

Otro método que también puede implementarse es el del *proxy virtual* [20] en el que se establece un punto virtual denominado *God-Object*, el cual en ciertos casos no es capaz de penetrar objetos sólidos. Sin embargo, si el efector final penetra un objeto sólido, el movimiento del *God-Object* es restringido a la superficie tal como se visualiza en la Figura 2.17, generando así una fuerza que puede ser calculada simulando un resorte ideal sin masa. Esto resulta ser igual al método anterior ya que también se utiliza la *Ley de Hooke* (2.12) para generar la fuerza que finalmente se verá reflejada en el dispositivo háptico.

Una vez obtenido el renderizado de fuerza ésta debe ser transmitida de forma paralela tanto en el objeto simulado gráficamente como en el dispositivo háptico, teniendo así una retroalimentación de dos vías en el sistema.

2.2.2.2. Renderizado de fuerzas desde la perspectiva de control

Así como en el área gráfica se denota la importancia del renderizado de fuerzas, la parte de control también juega un papel muy importante ya que dentro del sistema existe una retroalimentación que permite al usuario interactuar con la aplicación. Por lo tanto resulta conveniente visualizar este tipo de desarrollos como un sistema de control

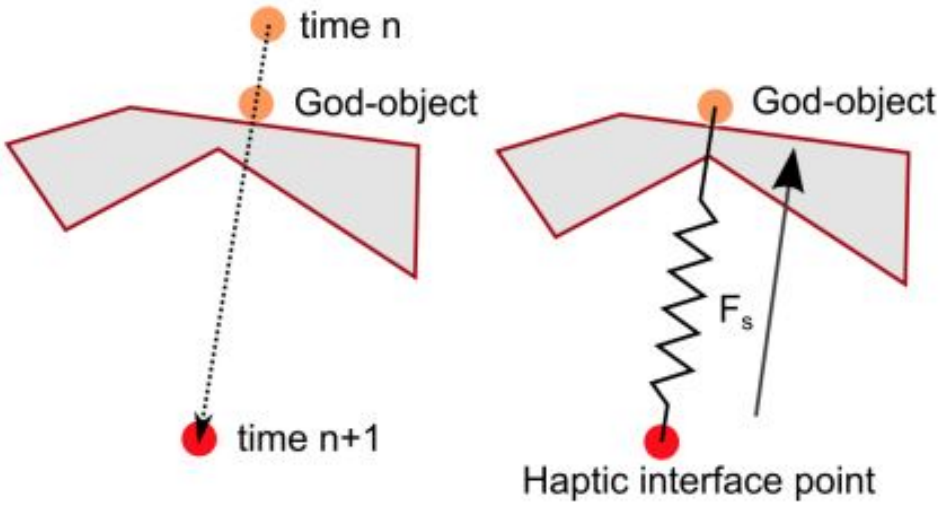


Figura 2.17: Esquema del sistema Proxy virtual [20].

clásico igual al que se muestra en la Figura 2.18. En este sistema se tiene cuatro señales: la señal de salida *y(t)* que representa el estado actual de la planta, la señal de referencia $y_d(t)$ que define el estado deseado, la señal de error *e(t)* que refleja la diferencia entre el estado actual y el deseado tal como se define en la ecuación

$$e(t) = y_d(t) - y(t) \tag{2.15}$$

y por último la señal de control *u(t)* que se aplica directamente a la planta con el fin de que el error tienda a cero de forma asintótica y en tiempo finito [19].

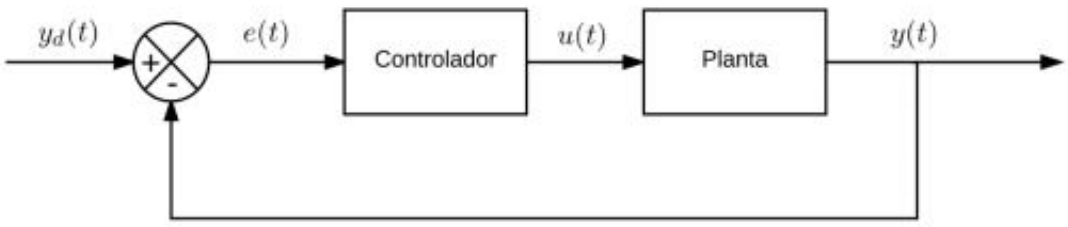


Figura 2.18: Esquema de un sistema de control [19].

Desde el enfoque de control automático se hace uso de un *modelo de impedancia* en el cual la interfaz háptica mide el desplazamiento y la simulación virtual calcula la

fuerza correspondiente. Para lograrlo se necesita contar con un *control por impedancia*, como el que se visualiza en el diagrama de bloques de la Figura 2.19, en donde:

- x es el vector de posición del efector final de la interfaz.
- F es el vector de fuerzas del efector final de la interfaz.
- τ es el vector de par de cada articulación con la que cuenta la interfaz.
- q es el vector de variables articulares.
- F_d es el vector de fuerzas deseadas.
- τ_d es el vector de par deseado.
- J^T es el Jacobiano transpuesto de la interfaz.

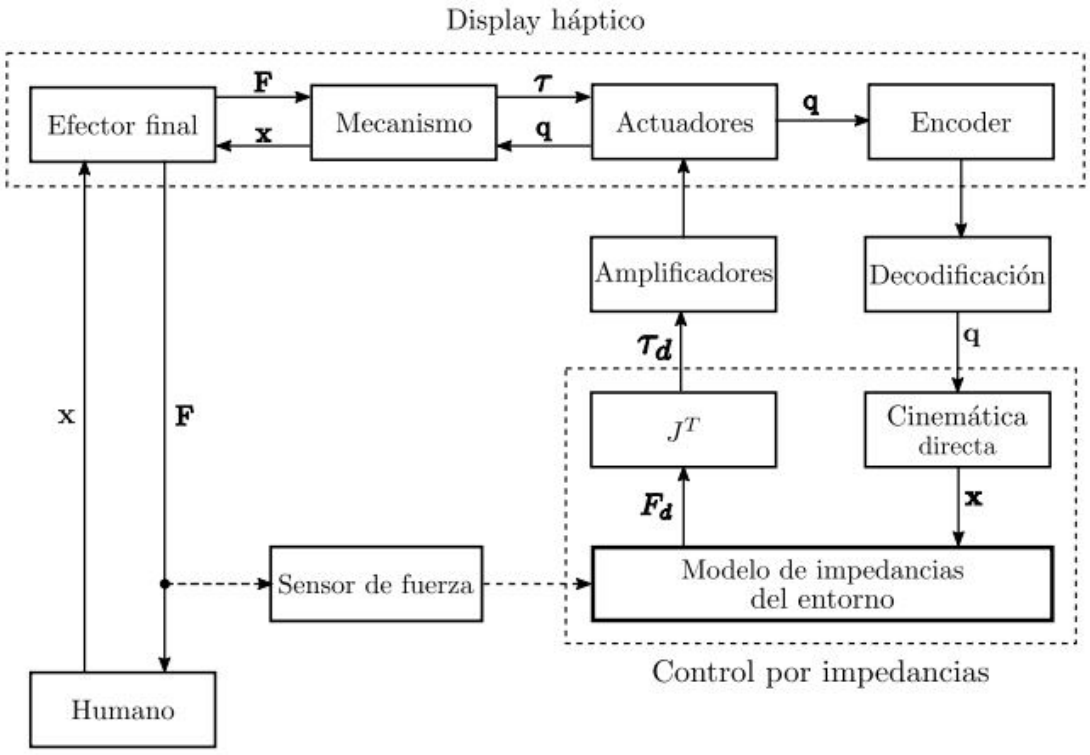


Figura 2.19: Control por impedancias [8] (las flechas punteadas representan información complementaria).

Este controlador se asemeja a un sistema masa-resorte-amortiguador, el cual trabaja de la misma forma que el método de penalización que fue visto en la parte gráfica

(2.11), teniendo como señal de entrada el desplazamiento realizado y como señal de salida una fuerza. Sin embargo dicha respuesta se encuentra basada en un controlador Proporcional-Integral-Derivativo (PID) el cual cuenta con tres acciones de control que describen su comportamiento y que es definido por la ecuación

$$u(t) = k_p \left(e(t) + \frac{1}{T_i} \int_0^t e(t)dt + T_d \frac{de(t)}{dt} \right). \tag{2.16}$$

Variando el valor de la ganancia k_p se pueden definir los siguientes controladores:

- El primero se trata de un control P, el cual recrea el comportamiento de un resorte con una constante elástica k_p.
- El segundo es un control PD que emula el comportamiento de un sistema resorte-amortiguador y donde el producto $k_p T_d$ representa la constante de amortiguamiento.
- El último es un control PID el cual agrega un término integral al sistema resorte-amortiguador y que físicamente no tiene representación.

2.3. Simulación

Esta etapa es la encargada de recrear las fuerzas establecidas en la fase de renderizado háptico así como vincularlas correctamente con los gráficos creados en el renderizado visual. Por lo tanto se puede decir que es el punto de unión entre ambas partes del sistema. Para poder realizar con éxito dicha tarea se debe tomar en cuenta tanto la frecuencia usada por los algoritmos hápticos al realizar sus cálculos así como la que el sistema de cómputo proporciona al momento de generar el renderizado visual. Uno de los principales puntos a considerar al momento de la simulación es el tipo de objetos presentes en el sistema ya que dependiendo de la dinámica con la que cuenten *(rígida o deformable)* éstos deberán ser modelados a una determinada frecuencia. Cuando la estructura de un cuerpo no se altera al momento de interactuar con una fuerza se dice que es un cuerpo rígido, sin embargo, si presenta un ligero cambio en su estructura se trata de un objeto deformable. Es de suma importancia aclarar esta distinción ya que usualmente un objeto rígido se modela a una frecuencia de 1 [KHz], mientras que uno deformable requiere aproximadamente 50 [Hz] para ser generado [2].

Tomando en cuenta lo anterior es necesario realizar un proceso denominado *virtual coupling* en el cual se ajustan las frecuencias utilizadas en los distintos objetos con la generada por el procesador de la computadora, buscando establecer la diferencia de tiempo necesaria en el renderizado háptico utilizando la ecuación

$$\Delta_h = \frac{f_o \Delta_v}{f_v} \tag{2.17}$$

en donde Δ_h es la diferencia de tiempo del renderizado háptico que se requiere para cada objeto presente en el sistema, f_o es su frecuencia háptica, Δ_v es la diferencia de

tiempo necesaria para generar el renderizado visual y f_v es la frecuencia en que éste es realizado.

Este procedimiento resulta ser una herramienta de gran importancia ya que gracias a un correcto acoplamiento de las posiciones de contacto entre el *avatar* (que se encuentra asociado a la posición del efector final del robot) y los objetos virtuales se obtiene el valor de la fuerza a simular. Por lo tanto dichas posiciones deben ser procesadas correctamente desde la fase háptica donde son obtenidas hasta la parte gráfica en donde se visualizan, pudiendo retornar al usuario una fuerza resultante en cuestión de segundos tal como se muestra en el diagrama de bloques de la Figura 2.20.

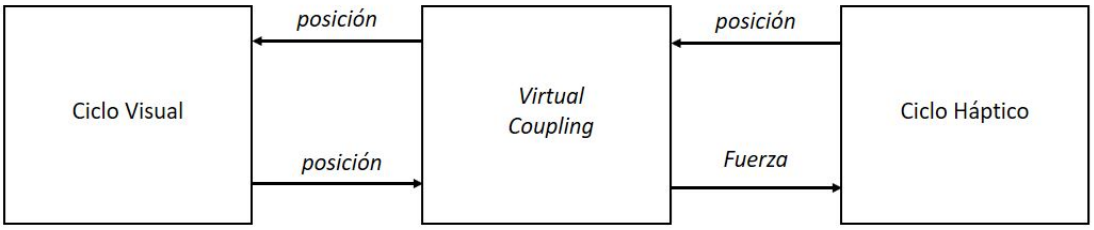


Figura 2.20: Proceso de virtual coupling.

2.4. Comentarios del capítulo

De manera general se puede resumir que el desarrollo de un sistema de RV con interacción háptica necesita de tres etapas escenciales: *renderizado visual*, *renderizado háptico* y *acoplamiento*, así mismo cada una cuenta con características que permiten al usuario determinar el alcance de sus desarrollo, desde las figuras o entornos que se busquen renderizar gráficamente hasta las fuerzas de interacción que se quieran generar.

Capítulo 3

Los robots *Geomagic Touch* y *Novint Falcon* como interfaces hápticas

Así como se mostró detalladamente el análisis de cada etapa que conforma a un sistema de RV con retroalimentación háptica, es de suma importancia conocer a fondo los dispositivos que permiten realizar dicha interacción. En el caso de este trabajo de tesis se ocuparon dos robots hápticos: *Geomagic Touch* y *Novint Falcon*, los cuales cuentan con una serie de características tanto generales como particulares que son trascendentales en la búsqueda de un resultado eficiente.

3.1. Características generales

Para comprender mejor el funcionamiento de los robots hápticos es necesario tener en cuenta una serie de características y conceptos [11]:

- *Eslabón.* Es un cuerpo que posee al menos dos nodos (puntos de unión con otros cuerpos). La mayoría de los robots actuales están construidos con eslabones rígidos, lo cual permite realizar análisis geométricos con parámetros constantes. Si los eslabones son flexibles, la dificultad para estudiar matemáticamente al robot se incrementa.

- *Mecanismo.* Es una cadena cinemática en la cual al menos un eslabón está fijo a tierra.

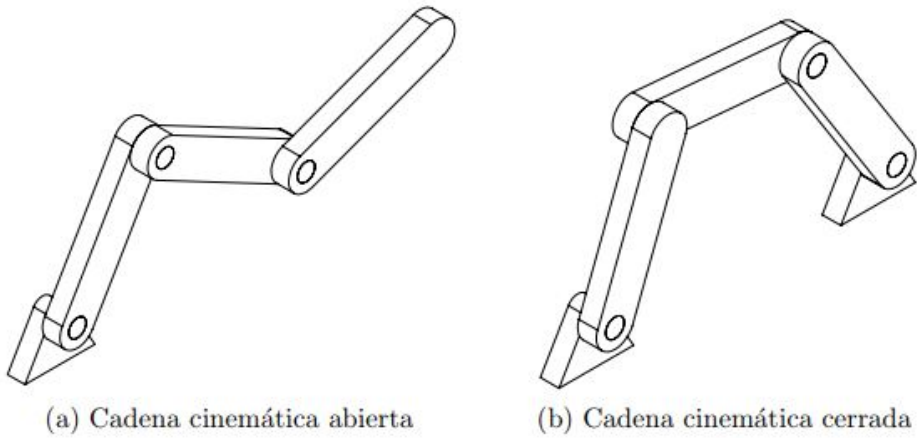
- *Articulación* (también llamada junta o par cinemático). Es la unión de dos o más eslabones en sus nodos. Existen varios tipos de juntas y una forma de clasificarlas es por los movimientos que permiten realizar a los cuerpos que unen. Las más importantes en robótica son la prismática y la de revolución. La primera permite la rotación de ambos cuerpos alrededor de un eje fijo mientras que la segunda per-

mite traslación. Pares cinemáticos más complejos son equivalentes a una sucesión de varias juntas de revolución y prismáticas.

- *Cadena cinemática.* Se obtiene al unir un conjunto de eslabones con juntas.

3.1.1. Arquitectura

Existen dos tipos de arquitecturas para clasificar a los robots manipuladores, las cuales se muestran en la Figura 3.1. El inciso (a) representa una cadena cinemática abierta que está conformada por eslabones conectados de manera secuencial desde la base y hasta llegar al efector final. Esta arquitectura es característica de un tipo de robots denominados *seriales*. Por otro lado, el inciso (b) corresponde a una cadena cinemática cerrada que es propia de otro tipo de robots conocidos como *paralelos*. Éstos cuentan con una plataforma móvil, la cual se encuentra unida a una base por medio de una serie de cadenas cinemáticas independientes.



(a) Cadena cinemática abierta (b) Cadena cinemática cerrada

Figura 3.1: Tipos de cadenas cinemáticas: (a) abierta, (b) cerrada [7].

Tanto los robots seriales como los paralelos pueden lograr un mejor desempeño si logran cumplir con una serie de parámetros y son seleccionado para tareas específicas.

En el caso de los robots seriales se necesita un espacio de trabajo amplio ya que los movimientos a realizar no deben ser muy rápidos o bruscos, así mismo, no importa si el manipulador cuenta con una inercia muy grande. Además la precisión de posición no debe ser crítica ya que las fuerzas que generan no son muy grandes. Por otro lado los robots paralelos no necesitan un amplio espacio de trabajo, sin embargo sí requieren moverse de una forma más rápida así como contar con una inercia baja ya que se busca mayor precisión de posición; además cuentan con la capacidad de generar fuerzas de mayor magnitud [7].

Característica	Robot Serial	Robot Paralelo
Espacio de trabajo	Grande	Chico
Cinemática directa	Sencilla	Complicada
Cinemática inversa	Complicada	Sencilla
Errores de posición	Acumulativos	Promediados
Errores de fuerza	Promediados	Acumulativos
Rigidez	Baja	Alta
Inercia	Alta	Baja
Velocidad/Aceleración	Lenta	Rápida
Precisión	Baja	Alta
Calibración	Sencilla	Complicada

Tabla 3.1: Características de las arquitecturas serial y paralela - Comparación entre algunas de las características más importantes con las que cuentan los robots seriales y paralelos.

La Tabla 3.1 muestra las diferencias entre ambos modelos con el fin de comprender de forma comparativa las características de cada robot.

3.1.2. Espacio de trabajo

Representa el volumen total del entorno en el que el efector final del robot puede desplazarse. La forma de este espacio depende de diversos factores, como son: la dimensión de los eslabones, el tipo y número de juntas o las limitantes mecánicas. En pocas palabras, de la estructura general del robot. La Figura 3.2 muestra el espacio de trabajo que ocupa un robot *antropomórfico*, como lo es el *Geomagic Touch*.

3.1.3. Grados de libertad

Representan el número de parámetros independientes que se requieren para definir la posición y orientación de un cuerpo de forma única en cualquier instante de tiempo. El número de eslabones que tenga el robot y además el tipo de juntas con las que se unen establecen el número de grados de libertad con los que cuenta el mecanismo. Este hecho resulta ser trascendental ya que dependiendo del número de grados de libertad con los que se cuenten, se establecerán los tipos de tareas que el robot podrá realizar.

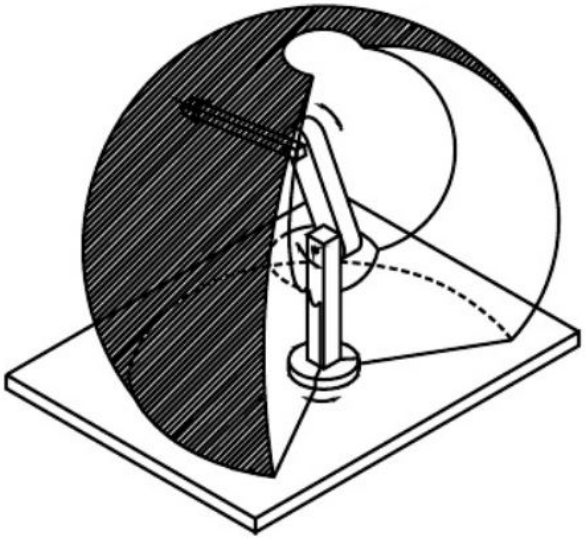


Figura 3.2: Espacio de trabajo de un robot antropomórfico [20].

Para que el mecanismo pueda alcanzar cualquier punto dentro de un espacio tridimensional, debe contar con al menos tres grados de libertad, permitiendo así posicionar su efector final en cualquier punto de su espacio de trabajo [7].

3.1.4. Cinemática directa

La cinemática de un robot se encarga de estudiar su movimiento basándose únicamente en la geometría que lo conforma. Ésta se divide en dos variantes: la directa y la inversa. La cinemática directa tiene como meta obtener la posición y orientación del efector final del robot con respecto al sistema de referencia base a partir de las variables articulares representadas con un vector de la forma

$$\mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix} \tag{3.1}$$

donde n representa el número de articulaciones, mientras que q_i puede ser un ángulo o una distancia ya sea que se trate de una articulación de revolución o prismática respectivamente.

Para resolver el problema de forma sistemática, es posible utilizar la convención de *Denavit-Hartenberg* [17], la cual trata de un algoritmo que permite seleccionar los sistemas de referencia en cada articulación de tal forma que se pueda obtener la posición y orientación del efector final respecto al sistema base por medio de un producto

de matrices. Estos arreglos se pueden ver reflejados en una ecuación como la que se muestra a continuación [7]:

$$ {}^{i-1}\mathbf{A_i} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\alpha_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2} $$

donde ${}^{i-1}\mathbf{A}_i$ denota la transformación de coordenadas del sistema de referencia i al sistema de referencia $i-1$. Los parámetros α_i, θ_i, a_i y d_i se definen mediante el algoritmo anteriormente comentado. Se debe tomar en cuenta que tres de los cuatro parámetros pueden ser constantes dependiendo del tipo de articulación con el que se esté trabajando. Si se trata de una articulación de revolución θ_i será variable. Mientras que si se trata de una articulación prismática d_i será variable.

Al realizar la multiplicación

$$ \mathbf{T} = {}^{0}\mathbf{A}_1^1\mathbf{A}_2\ldots\mathbf{A}_n \tag{3.3} $$

se obtiene la matriz $\boldsymbol{T}$ que representa la acumulación de transformaciones necesarias para obtener tanto la posición como la orientación del efector final con respecto a un sistema de referencia base.

3.1.5. Cinemática inversa

Resulta ser la operación contraria a la que realiza la cinemática directa ya que su objetivo es hallar las variables articulares conociendo la orientación y posición del efector final. Matemáticamente, el problema se puede plantear como la búsqueda de los valores q_i que cumplan la igualdad

$$ \mathbf{A}_1\mathbf{A}_2\ldots\mathbf{A}_n = \mathbf{T} \tag{3.4} $$

donde:

$$ \mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{d} \\ 0 & 1 \end{bmatrix}, \tag{3.5} $$

siendo $\boldsymbol{T}$ una matriz de transformación homogénea que representa la posición y orientación deseadas del efector final, mientras que $\boldsymbol{R}$ la matriz de rotación del mismo efector y $\boldsymbol{d}$ su vector de posición.

3.1.6. Interfaz de programación de aplicaciones (API)

Para poder realizar sistemas virtuales con interacción háptica se debe tener en cuenta las herramientas necesarias para su programación. Como ya ha sido mencionado anteriormente, en la presente tesis se ocuparon los robots *Geomagic Touch* y *Novint*

Falcon para realizar dicha interacción háptica. Dichos robots fueron fabricados por empresas diferentes, por lo que cuentan con características únicas como sus interfaces de programación o *APIs*. Se puede decir que una *API* es un conjunto de funciones, subrutinas y métodos de programación que ofrece cierta biblioteca o conjunto de librerías para ser usadas por otro software como una capa de abstracción [20]. El robot *Geomagic Touch* cuenta con una *API* conocida como *Open Haptics* mientras que el *Novint Falcon* utiliza la *API* denominada *Haptic Device Abstraction Layer (HDAL)* [19]. Ambas interfaces cuentan con una serie de librerías que permiten a cada dispositivo comunicarse con los sistemas de realidad virtual generados por computadora y realizar las correspondientes interacciones hápticas.

3.2. Robot háptico *Geomagic Touch*

Conocido anteriormente como *Sensable Phantom Omni* fue desarrollado por la empresa *3D Systems* con el fin de permitirle al usuario sentir objetos virtuales y producir sensaciones táctiles reales a medida que manipula objetos en 3D visualizados a través de una pantalla. Dicho robot, que puede verse en la Figura 3.3, cuenta con seis grados de libertad, lo que le permite ubicar la pluma o *stylus* con la que cuenta en su efector final en cualquier posición y orientación de su espacio de trabajo, permitiendo así generar un rango de movimientos más amplio y por ende darle una mayor destreza al mecanismo [20].

Figura 3.3: Robot *Geomagic Touch*.

Otro aspecto a considerar de este dispositivo es su arquitectura, dado que se trata de un robot serial, cuenta con las características convencionales para cierto tipo de tareas, tal como se explica en la sección 3.1.1 de este capítulo. De igual forma, es pertinente

comentar que cuenta con una interfaz *Ethernet* compatible con terminales *RJ45* o *USB* así como dos botones programables integrados en la pluma. En adición a lo anterior, en la tabla de la Figura 3.4 se pueden observar otras características técnicas con las que cuenta el dispositivo.

Espacio de trabajo con realimentación de fuerzas	~6.4 W x 4.8 H x 2.8 D in. = 160 W x 120 H x 70 D mm
Footprint (Area física que ocupa la base del dispositivo)	6 5/8 W in x 8 D in. 168 W mm x 203 D mm
Peso (solo el dispositivo)	3 lbs. 15 oz
Rango de movimiento	Movimiento manual que pivotea con la muñeca
Resolución Nominal de Posición	> 450 dpi 0.055 mm
Esfuerzo máximo en la posición Nominal (brazos ortogonales)	0.75 lbf (3.3N)
Esfuerzo Continuo (24 hrs)	> 0.2 lbf (0.88 N)
Dureza	X axis > 7.3 lbs/in (1.26 N/mm) Y axis > 13.4 lbs/in (2.31 N/mm) Z axis > 5.9 lbs/in (1.02 N/mm)
Inercia (masa aparente del joystick)	0.101 lbm (45 g)
Ejes de realimentación de fuerzas	x,y,z
Sensado de Posición (Tipo gimbal)	x, y, z (encoders digitales) Pitch, roll, yaw (±5 % linealidad de los potenciómetros)
Interfaz	IEEE-1394 FireWire® port: 6-pin to 6-pin
Plataformas Soportadas	Intel o PCs AMD-based
Compatibilidad con OpenHaptics Toolkit® SDK	OpenHaptics para Windows, Linux y Mac OS X

Figura 3.4: Características técnicas del robot *Geomagic Touch* [12].

Teniendo ya una percepción mas clara de las características generales de este mecanismo, es momento de conocer a mayor detalle la *API* con la que cuenta.

3.2.1. Librería Open Haptics

Tal como se citó previamente, *Open Haptics* es la *API* utilizada por este robot, la cual trata de simplificar su programación encapsulando los pasos básicos de todas las aplicaciones gráficas que cuenten con interacción háptica. Se encuentra conformada por tres librerías que son: *Quick Haptics micro API*, *Haptic Device API (HDAPI-HD)* y *Haptic Library API (HLAPI-HL)*, las cuales se relacionan entre sí tal como se muestra en la Figura 3.5. En ésta se puede ver que en su nivel superior se sitúa a *QuickHaptics* como la responsable de crear prototipos con escenas relativamente complejas ya que es donde se encuentra localizada la ejecución tanto del renderizado gráfico como el del háptico, siendo así el nivel que representa el mayor porcentaje de funcionalidad. En el segundo nivel se encuentran funciones que proporcionan efectos de renderizado de fuerza personalizados así como la interacción que tiene con las otras librerías. Por último, en el nivel inferior de la pirámide se muestra que la base de este funcionamiento son las librerías *HD* y *HL* ya que la primera provee acceso de bajo nivel a los *drivers* del dispositivo háptico, mientras que la segunda funge como intermediario entre *Quick Haptics micro API* y *HD*, puesto que ésta última cuenta con funciones de alto nivel necesarias para el renderizado háptico y que pueden integrarse con *OpenGL* en un conjunto de librerías gráficas de código abierto. Dado lo anterior, es conveniente conocer

a mayor fondo cada una de estas librerías.

Figura 3.5: Relación entre las librerías de *Open Haptics* [18].

3.2.2. Librería Quick Haptics

Esta librería se implementa en *C++* y se define en cuatro clases principales [18]:

- *Device Space.* Define el espacio de trabajo a través del cual el avatar asociado al efector final del dispositivo háptico puede moverse. Los métodos que se gestionan son:
  - *Efectos de fuerza.* Dentro de los que se encuentran la fricción, amortiguamiento (grado de dificultad al moverse por el espacio) y fuerza constante.
  - *Devolución de llamadas de usuario.* Son las peticiones de función que se producen como resultado de un evento. El movimiento denominado toque háptico o presión de un botón son ejemplos de eventos que pueden desencadenar una devolución de llamada.
- *QHRenderer*: Es una ventana en pantalla que representa los objetos virtuales desde el punto de vista de una cámara y permite al usuario tener interacción con ellos utilizando un dispositivo háptico. Cuenta con dos variantes: *QHWin32* y *QHGLUT*, las cuales son clases de creación de ventanas desarrolladas específicamente para su uso con la *API* de *Microsoft Win32* o el kit de herramientas de utilidad *OpenGL (GLUT)* respectivamente.

- *Forma (Shape).* Ejerce como clase base para la creación de uno o más objetos geométricos que se lleguen a representar tanto gráfica como hápticamente. Se pueden definir todas las primitivas de geometría a incluir en el espacio de trabajo como son:

  - Línea
  - Plano
  - Cono
  - Cilindro
  - Esfera
  - Caja
  - TriMesh
  - Texto

  Las propiedades que se pueden aplicar a cualquiera de estas primitivas incluyen: textura, color, giro y posición. Por otro lado, la clase *TriMesh* representa un modelo 3D producido por los programas de modelado estándar como *Blender*, por lo cual, se pueden incluir objetos *STL*, *OBJ*, *3DS* y *PLY*. Debido a que la geometría de esta clase naturalmente vincula vértices, bordes y caras, *OpenHaptics* ha implementado una forma sencilla de usarla a través de redes conformadas por resortes deformables.

- *Cursor.* Describe la representación gráfica del efector final con el que cuenta el dispositivo háptico es también conocido como *punto de interfaz háptico (HIP).* Esta clase es la encargada de recopilar información de todas las demás ya que calcula la ubicación final del *avatar* y requiere interacción con todos los componentes en una escena, por ejemplo:

  - Información de la clase *Device Space.* Es utilizada para determinar la ubicación del punto de interfaz háptica a partir del *Geomagic Touch Device Driver (GTDD)* ya que puede haber más de un dispositivo háptico, y por consiguiente más de un *avatar.*
  - Información espacial de la clase *QHRenderer.* Define la transformación que determina la posición del *avatar* en el espacio de trabajo del dispositivo con el fin de poder ser dibujado en pantalla.
  - Información de la clase *Forma (Shape).* Permite conocer detalles sobre los objetos a interactuar con el *avatar* así como la forma que éste tendrá al ser representado en pantalla.

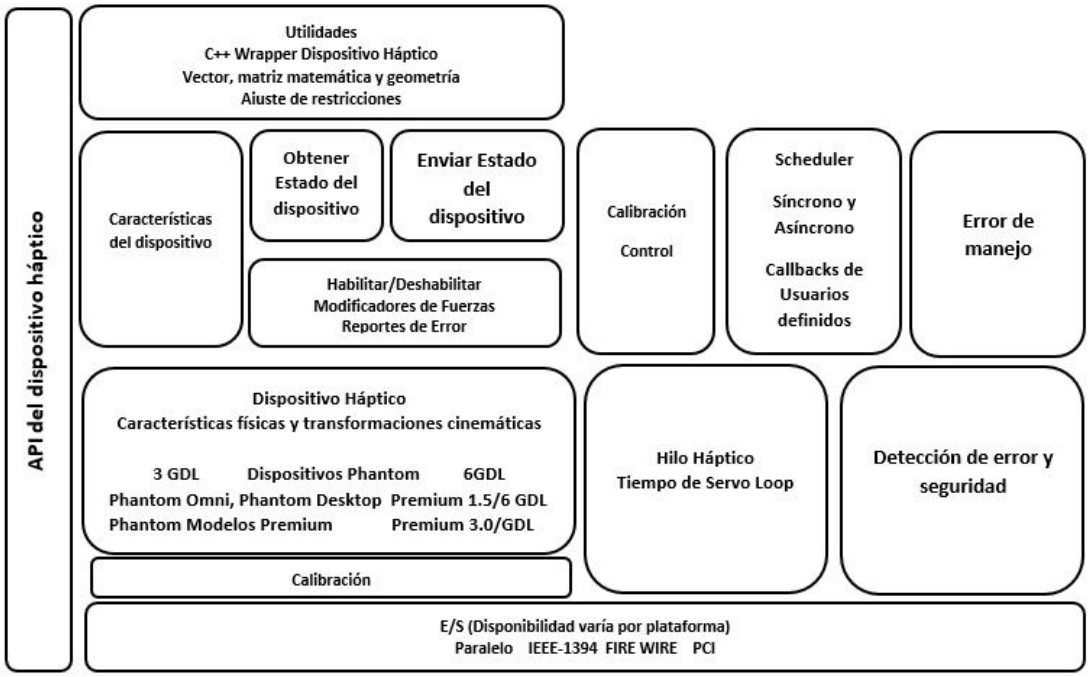


Figura 3.6: Arquitectura de HDAPI [12].

3.2.3. Haptic Device API (HDAPI-HD)

Así como se citó anteriormente, esta librería permite accesar a bajo nivel del dispositivo, ofreciendo así la posibilidad de renderizar fuerzas directamente; es decir, configurar los controladores en tiempo de ejecución y leer el estado en que se encuentra el dispositivo ya sea posición, velocidad o fuerza aplicada. Gracias a *HD* se puede integrar el dispositivo háptico en diferentes aplicaciones desarrolladas sobre lenguaje *C* o *C++*, con el fin de manipular sólidos generados a partir de imágenes de otros paquetes gráficos o de librerías especializadas que incluyan propiedades de deformación o colisión.

La arquitectura en la que está basada *HDAPI* se puede visualizar en la Figura 3.6, mientras que la Figura 3.7 muestra el diagrama de flujo que se debe seguir cuando se busca crear una aplicación de renderizado de fuerzas con objetos virtuales utilizando esta herramienta. De forma general el procedimiento consiste en [12]:

1. Inicializar el dispositivo.
2. Crear el *Scheduler*, que es la función encargada de repartir el tiempo disponible del microprocesador entre todos los procesos del sistema así como los *callback* necesarios para definir los efectos de fuerza.
3. Habilitar la salida de fuerzas.
4. Iniciar el *Scheduler*.

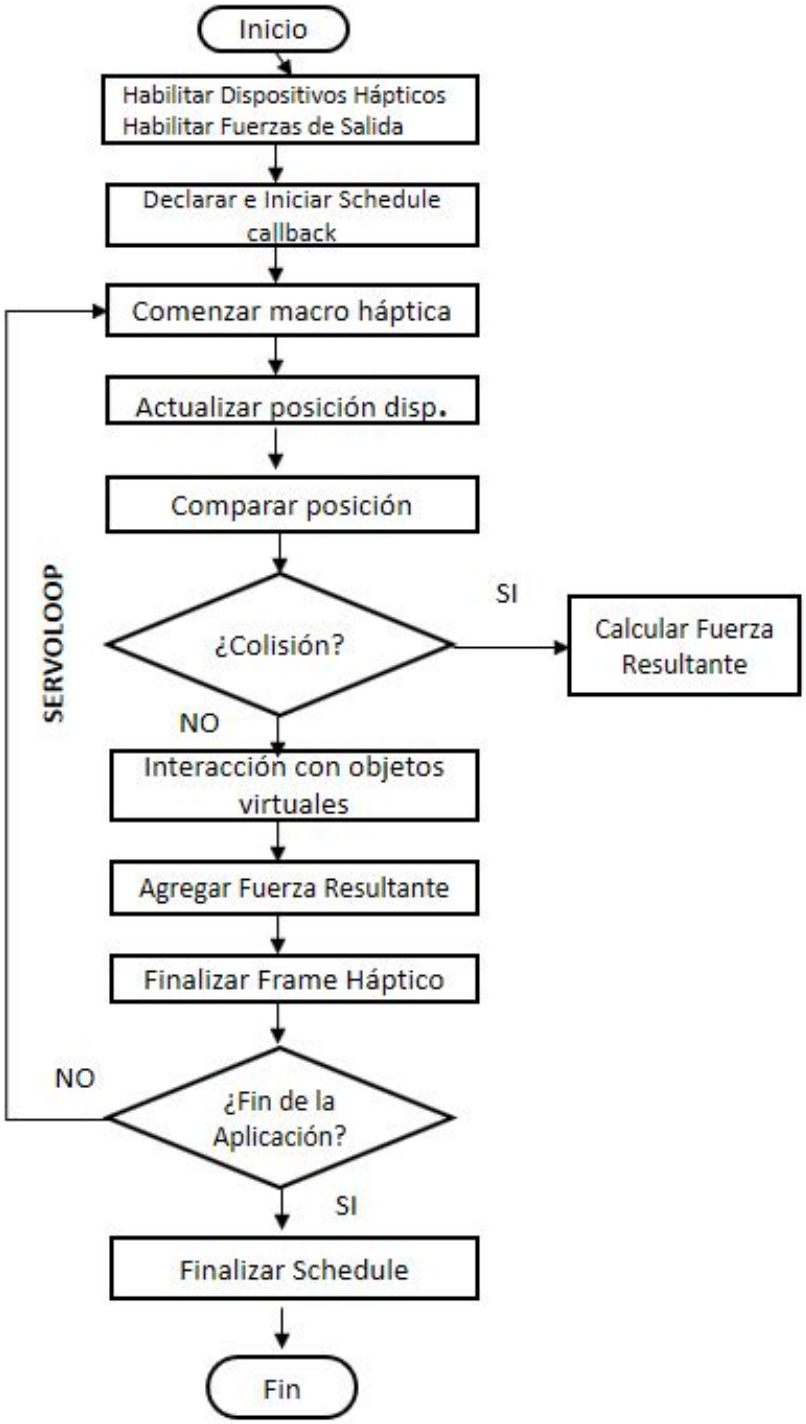


Figura 3.7: Diagrama de flujo del renderizado de fuerzas usando HDAPI [12].

5. Actualizar posiciones e iniciar efectos de fuerza.

En donde el *Servoloop* se refiere al lazo o bucle de control encargado de calcular las fuerzas y enviarlas el dispositivo háptico.

3.2.4. Haptic Library API (HLAPI-HL)

A diferencia de *HDAPI*, *HLAPI* es una librería que proporciona programación a alto nivel, por lo que incluye funciones de renderizado háptico para aplicaciones gráficas desarrolladas con *OpenGL*. De esta forma permite especificar sobre primitivas geométricas tales como triángulos, líneas y puntos así como diversas propiedades físicas como rigidez o fricción, de esta forma se facilita la integración con entornos virtuales cuyo código ya ha sido implementado [12].

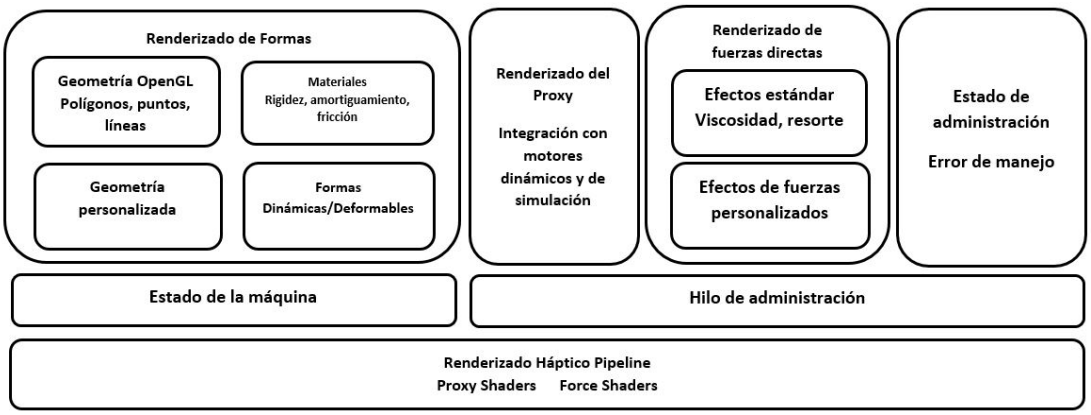


Figura 3.8: Arquitectura HLAPI [12].

En la Figura 3.8 se puede apreciar la arquitectura *HLAPI*, la cual oculta operaciones de control del entorno como son: la sincronización entre los hilos hápticos y gráficos, la detección de colisiones así como el cálculo y la generación de fuerzas sobre un punto de contacto virtual (*proxy*). Mientras que el diagrama representado en la Figura 3.9 muestra el procedimiento que se debe seguir si se desea obtener una aplicación gráfica con interacción háptica utilizando *HLAPI* cuyos pasos son [12]:

1. Inicializar el dispositivo.
2. Configuración e integración con *OpenGL* a través de la creación un contexto gráfico.
3. Inicialización de la aplicación con *HLAPI* mediante la creación de un contexto adicional asociado al comportamiento del dispositivo háptico.
4. Transformación de coordenadas físicas, que son las que está recibiendo directamente el dispositivo, a las coordenadas locales, las cuales serán las utilizadas dentro de la aplicación.

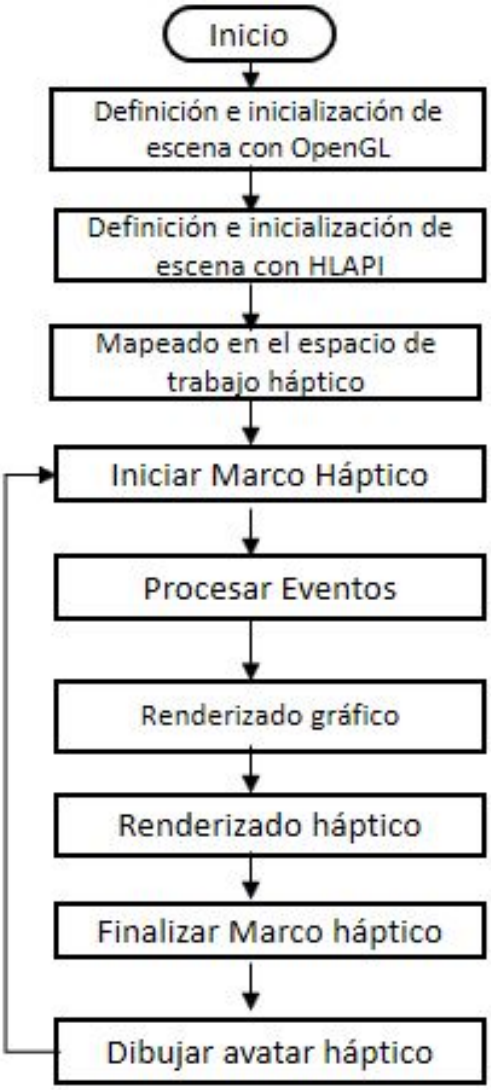


Figura 3.9: Diagrama de flujo del renderizado de fuerzas usando HLAPI [12].

5. Representación gráfica de la escena con *OpenGL* y renderizado háptico con *HL* y *HDAPI*.

6. Ejecución y atención al reporte de eventos generados en el hilo háptico.

Recapitulando, las diferencias entre *HLAPI* y *HDAPI* son el nivel de acceso que tienen al ser implementadas así como su utilidad ya que la primera es utilizada para el desarrollo gráfico mientras que la segunda se encarga de la integración háptica.

3.3. Robot háptico *Novint Falcon*

Es un dispositivo fabricado por la empresa *Novint Technologies*, el cual pretende interactuar con escenarios en 3D utilizando retroalimentación de fuerzas. En un principio fue diseñado para aplicaciones de entretenimiento (videojuegos) o para fungir como sustituto de periféricos más comunes ya que fue lanzado con costo relativamente accesible, por lo que fue convirtiéndose en un pionero dentro del mercado comercial de los productos hápticos [12].

Este dispositivo, que se observa en la Figura 3.10, a diferencia del *Geomagic Touch* cuenta con una arquitectura en paralelo de tres grados de libertad con los que es capaz de moverse en todas las direcciones espaciales: ancho, largo y profundidad, siendo su espacio de trabajo el conjunto de todos los puntos que puede alcanzar su efector final. Dicho efector está representado por una esfera que se encuentra al final del mecanismo y puede ser manipulada libremente por el usuario con la limitante de que no puede modificarse su orientación como sí ocurre con el *stylus* del *Geomagic Touch*.

Figura 3.10: Robot *Novint Falcon*.

Así mismo en la tabla de la Figura 3.11 se pueden observar otras características técnicas con las que cuenta el mecanismo.

3.3.1. Haptic Device Abstraction Layer (HDAL)

De manera similar a *Open Haptics*, *HDAL* es una *API* que contiene toda la documentación y archivos de software necesarios para desarrollar aplicaciones con el dispositivo *Novint Falcon*. Utiliza los lenguajes de programación *C* y *C++* garantizando compatibilidad con *APIS* gráficas como *DirectX* y *OpenGL*.

Los niveles de abstracción con los que cuenta *HDAL* se encuentran establecidos de una forma jerárquica en la que se permite una correcta sincronización entre la

Especificaciones de Hardware	
Tamaño	9"x 9"x 9"
Peso	6 lbs
Resolución de posición	> 400 dpi
Interfaz de Comunicación	USB 2.0
Alimentación	30W, 100V- 240V, 50Hz-60Hz
Grado de Libertad	3DOF input, 3DOF output Additional DOF possible through enabled grips
Especificaciones Software	
Plataformas soportadas	Windows XP y Vista Se anticipa migración ato PS3 and XBox
Mínimos requerimientos del sistema	Procesador 1 GHz, targeta gráfica de 128MB, DirectX Version: DirectX 9.0c 1.5 GB de espacio libre del disco duro Memoria RAM 512MB Conexión USB 2.0
API	C++ SDK overview Niveles de API: DHDLC (low-level drivers), HDAL (mid-level device communication) Falcon API (high-level programming toolset)
Especificaciones Hapticas	
Espacio de trabajo háptico	4"x 4"x 4"
Capacidad de fuerzas	> 2 lbf
Hilos separados para render háptico y gráfico	
3 grados de libertad con realimentación de fuerzas	

Figura 3.11: Características técnicas del robot Novint Falcon [12]

aplicación gráfica y la retroalimentación de fuerzas tal como se puede ver en la Figura 3.12. Cabe destacar que la comunicación existente entre la capa de simulación háptica y las funciones de la *API* se realiza a través de una función *callback*. Dentro de dicha función el usuario lee la posición del efector final del dispositivo y se calculan los niveles de fuerza que posteriormente serán aplicados. Así mismo, se cuenta con una serie de consideraciones para trabajar con las librerías *HDAL*, las cuales son [12]:

- La *API* está representada por dos archivos *include hdl/hdl.h* e *include hdlu/hdlu.h.* El primero representa la interfaz primaria para las funcionalidades básicas, mientras que el segundo es la interfaz de utilidades.
- La cadena de caracteres *Tool* con la que cuenta la librería se refiere al *Proxy* o cursor háptico del instrumento virtual ya que define la representación del efector final en la escena. De tal manera que es común encontrar parámetros como *hdlToolPosition*, *hdlSetToolForce*, *hdlToolButton*, *hdlToolButtons* los cuales se encuentran relacionados con la posición del cursor, el punto sobre el cual es aplicada la fuerza y las entradas digitales, respectivamente.
- Las unidades que maneja la interfaz son: distancia [m], fuerza [N], tiempo [s].
- El sistema coordenado base que utiliza sigue la regla de la mano derecha la cual

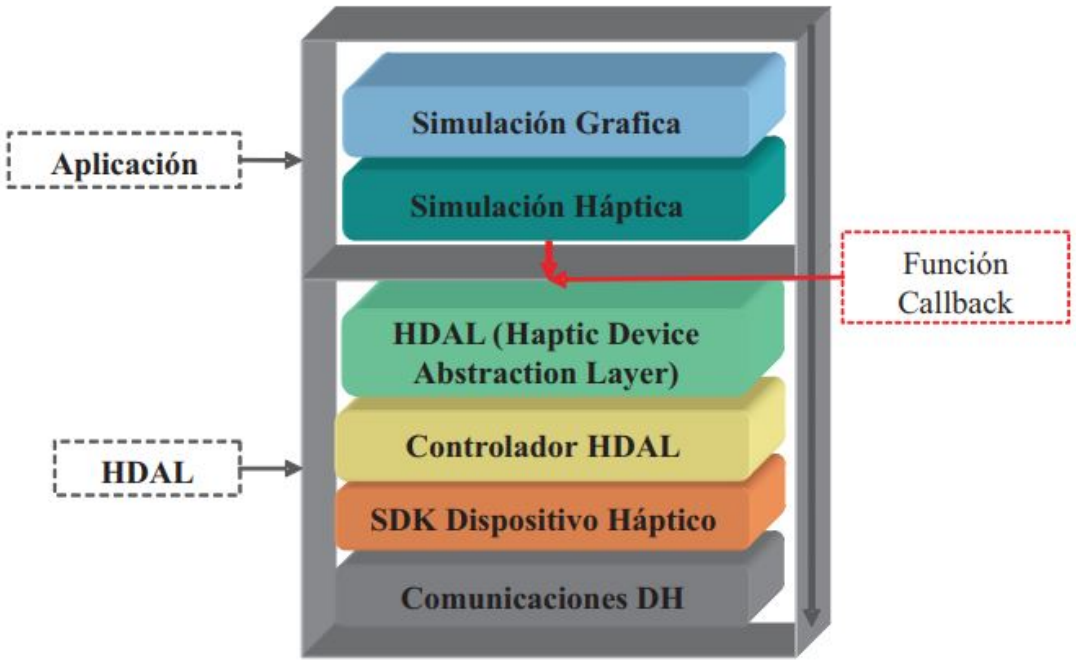


Figura 3.12: Arquitectura de HDAL [12].

define que:

- X incrementa hacia la derecha
- Y incrementa hacia arriba
- Z incrementa hacia el usuario

- El origen ($X = 0$, $Y = 0$, $Z = 0$) es aproximadamente el centro del espacio de trabajo del dispositivo háptico.
- El hilo *Servo Thread* maneja la comunicación con el dispositivo y hace el llamado de las funciones *callback* en las que se calculan y aplican las fuerzas con el fin de actualizar la posición y así intercambiar datos entre la aplicación y el usuario.
- Para que no se vea afectada la escena gráfica, el acceso y aplicación de datos hápticos (posición, tiempo, fuerzas) se realiza a través de una sincronización de *callbacks* con comunicación directa hacia el dispositivo.
- Se cuenta con la clase *HapticClass* y una función que inicia la transferencia de datos.
- La rutina de inicialización del dispositivo se compone de una secuencia de pasos, la cual se puede ver reflejada en la Figura 3.13.

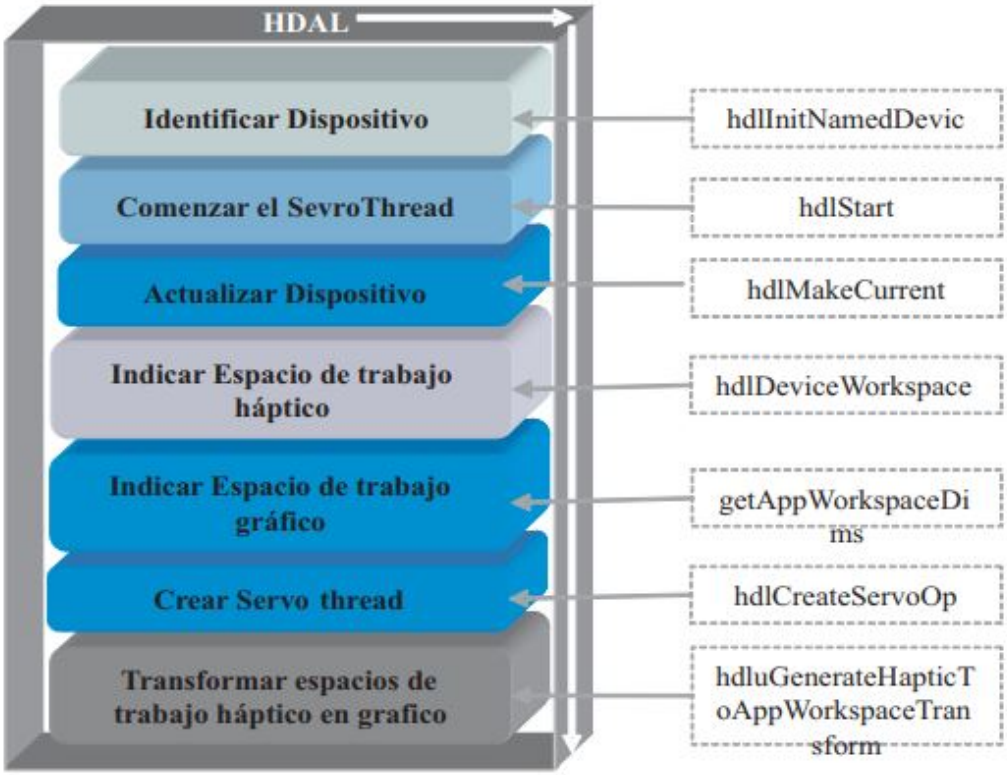


Figura 3.13: Inicialización de Novint Falcon utilizando HDAL [12].

3.4. Comentarios del capítulo

En resumen, tanto el *Geomagic Touch* como el *Novint Falcon* cuentan con características especiales que les permiten ser utilizados con distintos fines, así mismo las diferencias que existen entre ambos resultan ser de gran aporte ya que el usuario pude identificar la mejor opción dado el objetivo a lograr. Un claro ejemplo son las *APIs* de cada dispositivo, mientras que la del *Geomagic Touch* necesita de tres partes esenciales para lograr su objetivo, la utilizada por el *Novint Falcon* engloba todas las características necesarias en una sola.

Teniendo ya en cuenta el conocimiento necesario de cada dispositivo y de las herramientas necesarias para el desarrollo de aplicaciones virtuales con renderizado háptico, en el siguiente capítulo se mostrarán los pasos que se siguieron en el diseño de cada una de las aplicaciones desarrolladas en este trabajo de tesis.

Capítulo 4

Implementación y resultados

En los capítulos anteriores se sentaron las bases necesarias de cada una de las fases que componen un sistema de realidad virtual con interacción háptica, así como la descripción de los robots utilizados como interfaz y su programación. En este capítulo se describirán de manera específica las etapas de cada programa realizado durante el trabajo de tesis, es decir, la forma práctica en que se implementaron los métodos presentados en su desarrollo.

4.1. Sistema de realidad virtual para objetos rígidos utilizando el robot *Geomagic Touch*

Este primer sistema tiene como objetivo que el usuario pueda adentrarse en un ambiente virtual semi-imersivo tridimensional como el que fue presentado en la Sección 1.3.4, en el cual se puede interactuar con dos objetos rígidos, es decir, que no presentan deformaciones al momento del contacto, utilizando como medio háptico el robot *Phantom Touch*. Para representar a dichos cuerpos rígidos fueron utilizados una esfera y una caja, siendo estos dos de los objetos más comunes de representar gráficamente. Dichos objetos fueron modelados a partir de primitivas básicas como las que se indicaron en la Sección 2.1.2, además de que cuentan con un proceso de iluminación y sombreado, lo que hace que los cuerpos tengan un mayor realismo. Para comprender mejor el diseño de la aplicación, a continuación se presentarán los pasos que se siguieron en el desarrollo de cada una de las fases principales del sistema.

4.1.1. Renderizado visual

Tal como se comentó previamente, por la facilidad y sencillez de uso es muy común el uso de primitivas para el modelado gráfico en *OpenGL* y esto se pudo ver reflejado en el modelado de figuras básicas como planos, esferas y cubos. Para su modelado se

tomó como base de programación una estructura que parte de lo más sencillo (un punto) hasta lo más complejo (objeto). También es necesario contar con las librerías que permitan visualizar dicho renderizado, haciendo uso de *GLUT* en este caso. Para tener una mejor conceptualización, a continuación se presentarán tanto el pseudocódigo como la estructura general del código de programación implementado para modelar la caja utilizada en este sistema así como el de los pasos a seguir para visualizar los cuerpos en el programa principal. De igual forma en la Figura 4.1 se presenta el renderizado final del programa por lo que cuenta con una serie de planos que contienen al espacio virtual, dos objetos rígidos (esfera y caja), así como un avatar y un vector (cono azul y línea roja) respectivamente, con los cuales el usuario puede visualizar el lugar en donde se encuentra el efector final del robot dentro del ambiente virtual.

Pseudocódigo del modelado de la caja

1. Incluir las librerías necesarias (GLUT).
2. Crear la clase en la cual se recibirán las posiciones tomadas generadas en clases más sencillas(Punto, Vector, Plano).
3. Ajustar los vectores normales, definir el tamaño del objeto e indicar qué planos representan a cada cara de la caja.

Pseudocódigo del renderizado visual en el programa principal

1. Incluir las librerías y cabeceras necesarias para realizar el renderizado gráfico.
2. Definir tanto las variables globales, en caso de ser necesarias, así como las funciones en donde se realice el de renderizado de las diferentes figuras geométricas.
3. Definir y desarrollar la función que inicialice la librería *GLUT* ya que en términos de renderización gráfica es una de las que cuenta con mayor importancia, pues es la encargada de proporcionar diversas utilidades de *OpenGL*.
4. Definir cada una de las funciones que tengan como objetivo renderizar gráficamente a las figuras. En ellas se habilitarán las propiedades físicas que presente cada objeto como son: color, tamaño, luces, etc.
5. Crear la funciones que permitan visualizar el espacio de trabajo gráfico, en el cual se verán reflejadas las figuras generadas.
6. Crear la función principal del programa, la cual permite ejecutar la aplicación de manera general.

Estructura general del código utilizado en el renderizado gráfico del programa principal.

1. Se incluyen las diferentes librerías: Include "glut.h"
   Include "graficos.h"
   Include "point.h"
   Include "vector.h"
   Include "plane.h"
   Include "box.h"
   Include "figuras.h"
2. Se definen las variables que permitirán la creación de los diversos objetos. Box *box = new Box()
   Vector *vec = new Vector()
   Point *iP = new Point()
   CFiguras miFigura;
3. Definición de las funciones principales del renderizado visual.
   - void displayFunction(void);
   - void reshape(int width, int height);
   - void dibujaCursor(const DeviceDisplayState *pEstado); // Función genera el cursor
   - void dibujaLinea(const DeviceDisplayState *pState); // Función que genera vector rojo.
   - void DibujaCaja(const DeviceDisplayState *pEstado); // Función para dibujar la caja.
   - void DibujaVector(const DeviceDisplayState *pEstado);
   - void DrawPlanes(); // Función para generar los planos de la caja.
   - void drawWorkspace(); // Función para generar el espacio de trabajo.
   - void InitGL(); // Función que permite iniciar Open GL.
   - void dibujaSphere(); // Función que genera la esfera.
4. Se define la función principal del programa, en donde se mandan a llamar a las diversas funciones gráficas.
   main(int argc, char* argv[])
   - miFigura.esfera(20, 100, 100, Text1.GLindex); // Se define la esfera.
   - box− >InitializeBox(15); //Se define el objeto tipo caja

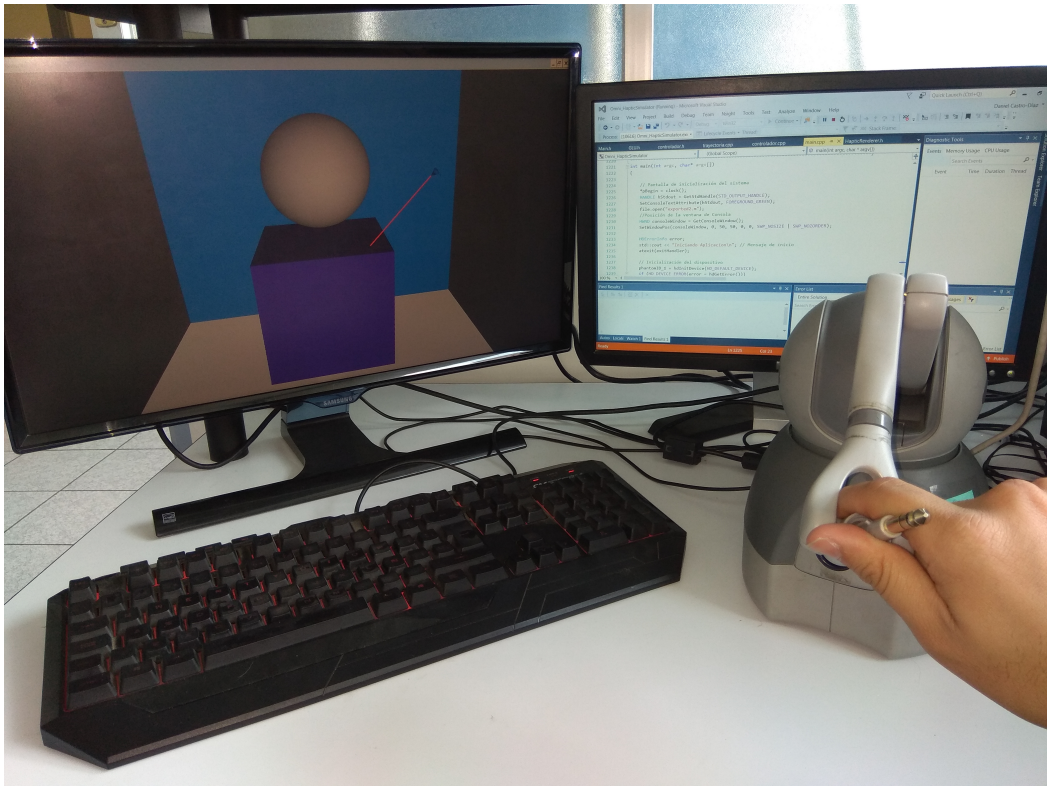


Figura 4.1: Sistema de realidad virtual con interacción háptica para objetos rígidos utilizando el robot *Geomagic Touch*.

4.1.2. Renderizado háptico

Para el desarrollo de esta etapa es importante tomar en cuenta los puntos especificados en la detección de colisiones vistos en la Sección 2.2.1 del presente trabajo ya que se tomarán en cuenta los algoritmos de *Ray Tracing*, en el caso de la caja y los planos, y el de superficie implícita para la esfera. Los dos funcionan como punto de apoyo en el proceso de renderización háptica.

Dado que el algoritmo de superficie implícita es el encargado de validar si se hace contacto con la esfera, ésta necesita estar definida por la ecuación:

$$f(x, y, z) = x^2 + y^2 + z^2 - R^2 \tag{4.1}$$

donde R es constante y representa su radio que además es el parámetro que permite comparar la región de contacto.

De igual forma es necesario tomar en cuenta lo citado en la Sección 2.2.2, en la cual se indica que los algoritmos de renderizado requieren la posición del avatar en función de los ejes coordenados X, Y y Z, por lo que si se utiliza la función del error entre la

posición del *avatar* asociada a la posición del efecto final del robot dentro del espacio de trabajo, se debe tomar en cuenta que ésta se define gracias a la siguiente ecuación:

$$e(t) = R - \sqrt{R^2 + f(x, y, z)} \tag{4.2}$$

Por otro lado las respuestas de los algoritmos de control *u(t)* representan el módulo del vector de fuerza que se está aplicando directamente al usuario mediante el dispositivo háptico. Este vector se obtiene al multiplicar su módulo por el vector unitario que parte desde el origen hacia la posición del avatar [19] y se calculan con la ecuación

$$F = \frac{u(t)}{\sqrt{x^2 + y^2 + z^2}} \begin{bmatrix} x \\ y \\ z \end{bmatrix}. \tag{4.3}$$

Así como en el etapa anterior, a continuación se presentará el pseudcódigo utilizado en el programa principal así como la estructura general del código desarrollado que permitió la renderización háptica. De igual forma en las Figuras 4.2, 4.3 y 4.4 se presenta una secuencia en donde se puede apreciar la interacción entre el usuario y los diferentes cuerpos que componen al sistema.

Pseudocódigo del renderizado háptico en el programa principal.

1. Incluir las librerías y cabeceras necesarias para el renderizado háptico que proporciona la *API* del dispositivo.
2. Definir las funciones *callback* en las cuales se realizarán los renderizados hápticos.
3. Definir los algoritmos de colisiones utilizados para cada uno de los objetos.
4. Determinar si se está haciendo o no contacto con los cuerpos y en caso de que exista, generar la fuerza que represente dicho contacto.

Estructura general del código implementado en el renderizado háptico del programa principal.

1. Se incluyen las librerías y cabeceras necesarias para el renderizado háptico:
   Include <HapticRenderer.h>
   Include <HD/hd.h>
   Include <HDU/hdu.h>
   Include <HDU/hduError.h>
   Include <HDU/hduVector.h>.
2. Definir *callbacks* del programa:
   HDCallbackCode HDCALLBACK DeviceStateCallback(void *pUserData)
   HDCallbackCode HDCALLBACK MainCallback(void *data)
3. Definir algoritmos de colisiones y renderizado de fuerzas de cada objeto dentro del *callback* principal.

   - Planos de la caja:
     IF (posiciónR[0] < PosX) // Se compara la posición obtenida por el robot con una establecida. Si la condición anterior se cumple el renderizado de fuerza es aplicado
       - pHaptic− >fuerza[0] = kp*abs((posiciónR[0]+PosX) - b*vx;
       - pHaptic− >fuerza[1] = 0;
       - pHaptic− >fuerza[2] = 0;

     EN CASO CONTRARIO: No se aplica renderizado de fuerza.

   - Renderizado Esfera: // Se delimita el área de contacto (Bounding Volumes). Se obtiene el valor del radio de la esfera dadas las posiciones cachadas por el robot.

     // Si se cumple la condición se obtiene una función de error.
     IF (radio < radioE)
     errorN = (radioE - radio);
     pHaptic− >fuerza[ii] = (1.2*errorN)*posicionR[ii] / radioE; //Se realiza renderizado de fuerza.

     EN CASO CONTRARIO: No se encuentra en contacto y por ende no se renderiza fuerza alguna.

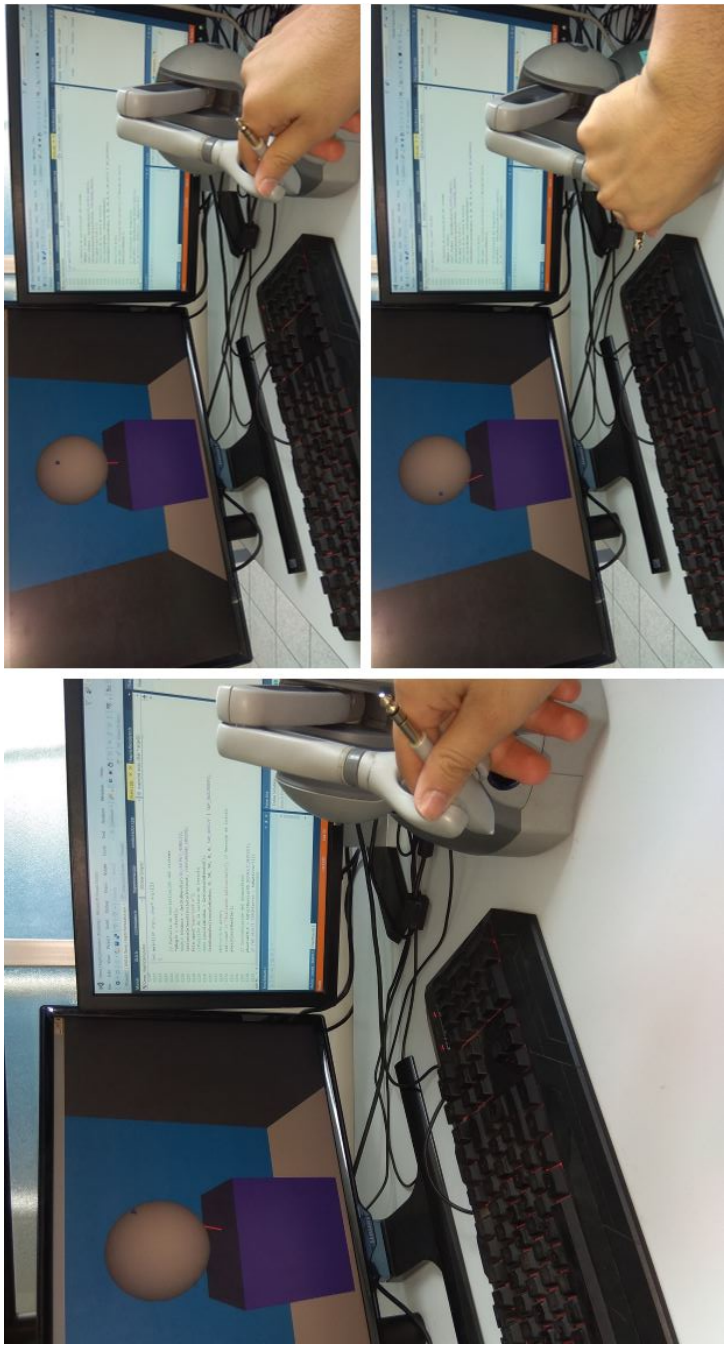


Figura 4.2: Secuencia de contacto con la esfera utilizando dispositivo *Geomagic Touch*.

.

Secuencia de contacto con la esfera utilizando dispositivo *Geomagic Touch* (visualizada en la Figura 4.2).

1. En el primer cuadro del lado izquierdo se puede apreciar que el *avatar* hace un primer contacto con la esfera del lado derecho dada la perspectiva del usuario.
2. En el cuadro superior de la derecha se puede notar que el avatar se recorrió hacia el centro de la esfera.
3. Por último, en el cuadro inferior se puede notar que se está realizando contacto del lado opuesto al que se realizó en el primer cuadro de la secuencia.

Secuencia de contacto con la caja utilizando dispositivo *Geomagic Touch* (visualizada en la Figura 4.3).

1. En la primera imagen de izquierda a derecha se puede apreciar que el usuario se encuentra a punto de realizar contacto con una de las caras de la caja.
2. En la imagen superior de la derecha se puede visualizar el contacto que se está haciendo con la cara basal superior de la caja.
3. Por último en la imagen inferior de la derecha denota contacto entre el *avatar* y la cara frontal de la caja.

Secuencia de contacto con el espacio virtual utilizando dispositivo *Geomagic Touch* (visualizada en la Figura 4.4).

1. Como inicio de la secuencia en el cuadro superior izquierdo se puede apreciar que el *avatar* está en contacto con la pared posterior del entorno virtual.
2. En segundo plano se puede visualizar en el cuadro superior derecho que el *avatar* está en contacto con la cara inferior o piso del ambiente virtual.
3. Por otro lado en el cuadro inferior de la izquierda se puede notar que se está realizando contacto con la cara lateral derecha del entorno semi-inmersivo.
4. Para concluir con esta secuencia, en el cuadro inferior derecho se visualiza la interacción entre *avatar* y la cara lateral izquierda del entorno.

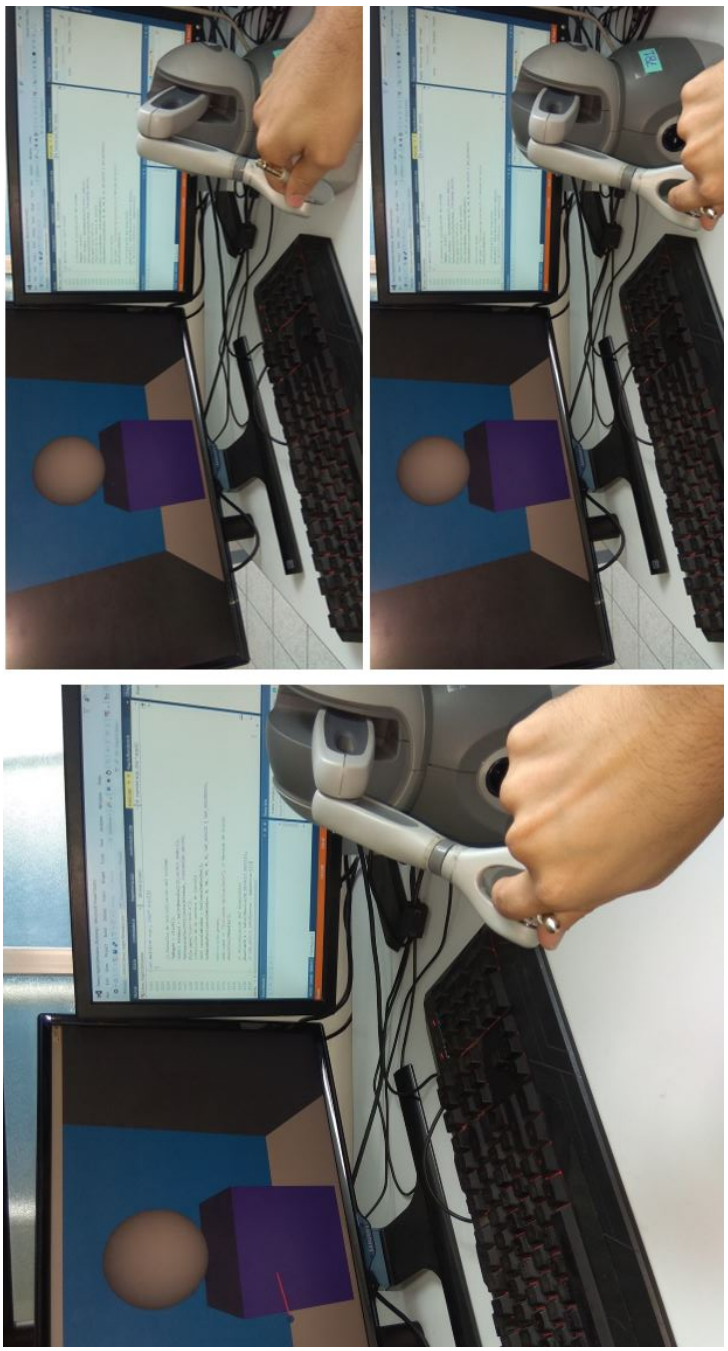


Figura 4.3: Secuencia de contacto con la caja utilizando dispositivo Geomagic Touch.

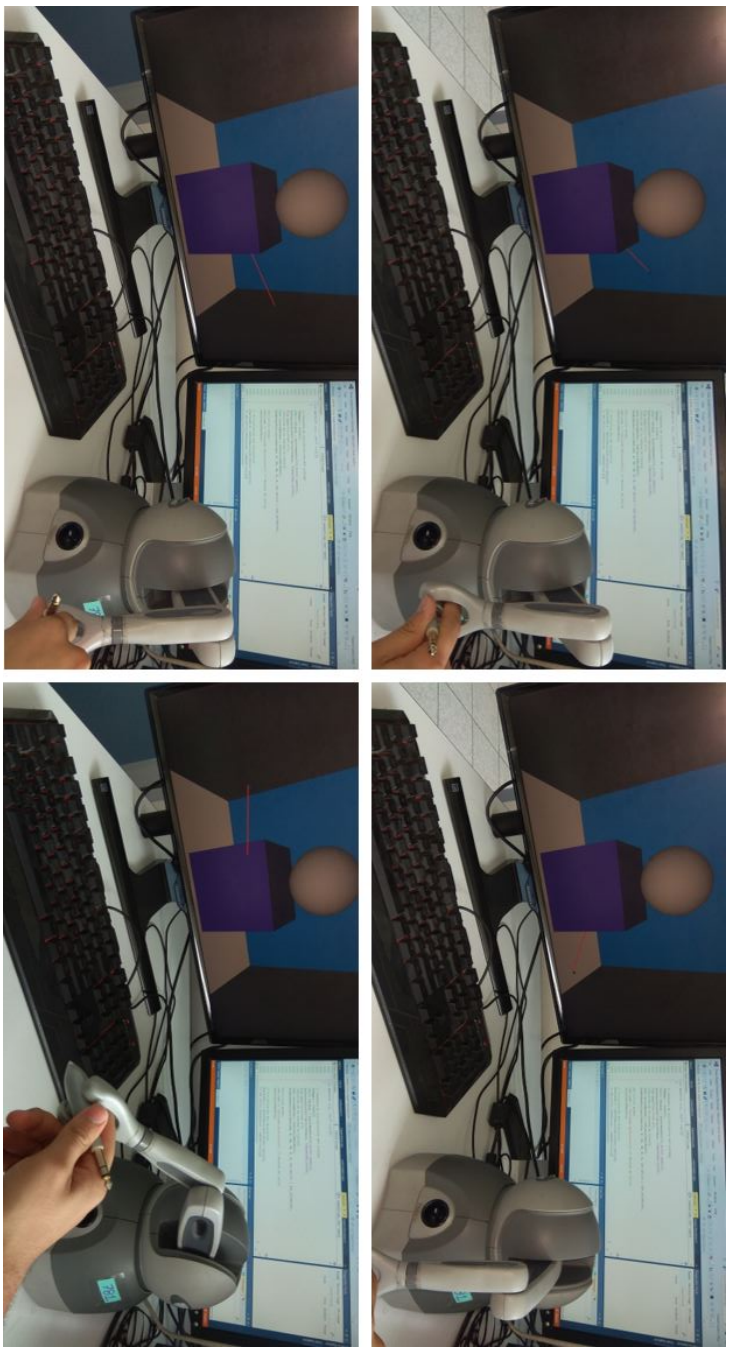


Figura 4.4: Secuencia de contacto con el espacio virtual utilizando dispositivo *Geomagic Touch*.

4.2. Sistema de realidad virtual para objetos rígidos utilizando el robot *Novint Falcon*

Al igual que en el primer sistema, este desarrollo pretende que el usuario pueda interactuar en un ambiente virtual semi-inmersivo con objetos rígidos. Sin embargo, tiene ciertas modificaciones, por ejemplo, este sistema no cuenta con un espacio tridimensional, al contrario es uno plano, es decir, en solo dos dimensiones, así mismo, los cuerpos presentados en este sistema son tres en vez de dos, esto con el fin de establecer que se pueden implementar tantos cuerpos se desee o requiera el usuario; de igual forma estos cuerpos no son exactamente los mismos que en el sistema anterior, ya que a pesar de tomar en cuenta la representación de una esfera, los demás objetos presentan modificaciones ya que al ser modelados a partir de primitivas básicas sus parámetros cambian, obteniendo formas más rectangulares que la caja presentada en el primer programa. Además se les incorpora un proceso de texturización con el fin de representarlas como cuerpos aún más realistas. De igual forma, para comprender mejor su diseño se indicarán a continuación los pasos que se siguieron para el desarrollo de cada una de las fases principales que comprenden al sistema.

4.2.1. Renderizado visual

El proceso del renderizado visual de este sistema es bastante parecido al visto en la Sección 4.1.1, ya que cuenta con parámetros y funcionalidades similares, sin embargo, a los cuerpos presentes se le agregó una particularidad: el manejo de texturas. Éste fue implementado a través de un procedimiento diferente en el modelado de figuras ya que se utilizó una nueva clase en la que se pueden englobar diversas primitivas como son esferas, cubos y cilindros entre otras y complementarlas con las propiedades físicas que pudieran presentar los objetos (color, tamaño, iluminación, texturas). Para comprender de una mejor forma el diseño de este sistema a continuación se presentarán tanto el pseudocódigo utilizado como la estructura general del código ya implementado, así mismo en la Figura 4.5 se puede apreciar el renderizado final, en donde se observa que el *avatar* es representado por una pequeña esfera debido a que resulta ser más acorde al efector final del robot *Novint Falcon*.

Pseudocódigo del renderizado visual en el programa principal

1. Incluir las librerías y cabeceras necesarias para realizar el renderizado gráfico.
2. Definir tanto las variables globales, en caso de ser necesarias, así como las funciones en donde se realice el de renderizado de las diferentes figuras.
3. Definir y desarrollar la función que inicialice la librería *GLUT* ya que en términos de renderización gráfica es una de las que cuenta con mayor importancia, pues es la encargada de proporcionar diversas utilidades de *OpenGL*.
4. Definir cada una de las funciones que tengan como objetivo renderizar gráficamente las figuras. En ellas se habilitarán las propiedades físicas que presente cada objeto como son: color, tamaño, luces, etc.
5. Crear las funciones que permitan visualizar el espacio de trabajo gráfico en el cual se verán reflejadas las figuras generadas.
6. Crear la función principal del programa.

4.2.2. Renderizado háptico

Al igual que en el caso anterior, se utilizó un algoritmo de colisiones que permitiera validar la existencia de algún contacto con los objetos. Sin embargo para este sistema en particular se utilizó un algoritmo del tipo *Bounding Values (AABB)*, el cual se puede visualizar en la Figura 2.16, con dicho algoritmo se estableció una región delimitada por seis parámetros fijos: (X_{min}, Y_{min} y Z_{min}) (X_{max}, Y_{max}, Z_{max}) los cuales representan las coordenadas de dos vértices diagonalmente opuestos de la caja. Este algoritmo fue utilizado para delimitar el área ocupada por los objetos de forma rectangular que fueron agregados en el sistema.

Así mismo, el proceso de diseño del renderizado háptico es parecido al del sistema anterior, el cual fue mostrado en la Sección 4.1.2 del Capítulo 4, puesto que solamente se adaptaron a las características y herramientas con las que cuenta el dispositivo tal cual se muestra tanto en el siguiente pseudocódigo como en la estructura general del código.

Estructura general del código utilizado en el renderizado gráfico del programa principal.

1. Se incluyen las diferentes librerías:
   Include "glut.h"
   Include "figuras.h"
   Include "SOIL.h"
   Include "texture.h"
2. Se definen las variables que permitirán la creación de los diversos objetos.
   CTexture text1 ,text2
   CFiguras Mifigura.
3. Definición de las funciones principales del Renderizado visual.
   - void glutDisplay(void);
   - void glutReshape(int width, int height);
   - void glutIdle(void);
   - void initGL();
   - void initScene();
   - void drawGraphics();
   - void drawCursor();
   - void pyramid(float x, float y, float z);
   - int LoadGLTextures();
4. Se define la función principal del programa, en donde se mandan a llamar a las diversas funciones gráficas.
   main(int argc, char* argv[])
   - Mifigura.esfera(.7, 40, 40, text2.GLindex); // Se define la esfera.
   - Mifigura.prisma(2, .5, 1, text1.GLindex); // Se definen los prismas.

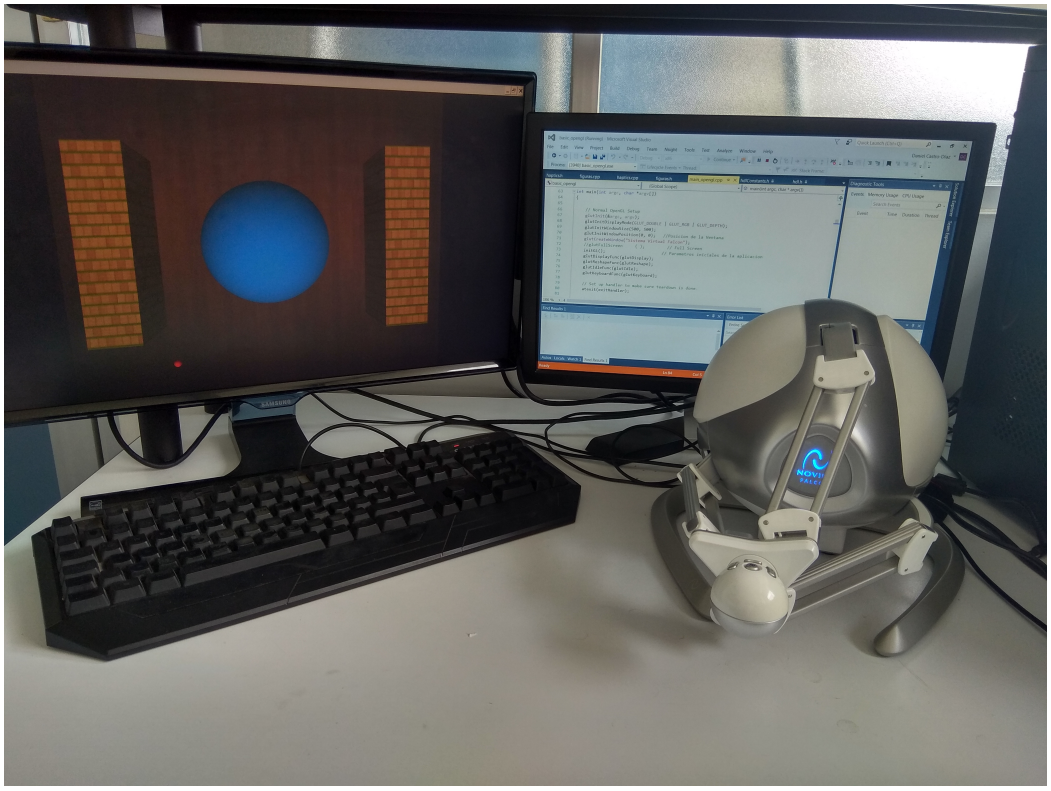


Figura 4.5: Sistema de realidad virtual con interacción háptica para objetos rígidos utilizando el robot *Novint Falcon*.

Pseudocódigo del renderizado háptico en el programa principal.

1. Incluir las librerías y cabeceras necesarias para el renderizado háptico que proporciona la *API* del dispositivo.
2. Definir las funciones *callback* en las cuales se realizarán los renderizados hápticos.
3. Definir los algoritmos de colisiones utilizados para cada uno de los objetos.
4. Determinar si se está haciendo o no contacto con los cuerpos y en caso de que exista, generar la fuerza que represente dicho contacto.

Estructura general del código implementado en el renderizado háptico del programa principal.

1. Se incluyen las librerías y cabeceras necesarias para el renderizado háptico:
   Include "haptics.h".
2. Definir funciones del renderizado háptico
   void HapticsClass::renderPID().
3. Definir algoritmos de colisiones y renderizado de fuerzas de cada objeto virtual.
   - Esfera: Se determina el valor del radio dadas las posiciones obtenidas por el robot.

     IF (radio < radioE) // Si el valor del radio obtenido es menor a uno establecido.

     errorN = (radioE - radio); //Se genera una función de error.
     mforceServo[i] = (errorN)*pointLC[i] / radio. //Se cumple la condición y se renderiza la fuerza.

     EN CASO CONTRARIO: El valor de la fuerza = 0.
   - Renderizado Prisma: Se delimita el área de contacto y valor del volumen.

     IF (volumen < VolumenE) // Si se cumple la condición se genera una función de error.

     errorN = (VolumenE - volumen);
     mforceServo[i] = ((Función de error*K)*pointLC[i])* -1; // Se renderiza valor de la fuerza.

     EN CASO CONTRARIO: No se encuentra en contacto y por ende no se renderiza fuerza alguna.

Por último, al igual que en el desarrollo anterior, en las Figuras 4.6 y 4.7 se presenta una secuencia de cuadros en las que se puede visualizar la interacción háptica entre el usuario y los diversos objetos que componen al sistema.

Figura 4.6: Secuencia de contacto con esfera utilizando dispositivo Novint Falcon.

Figura 4.7: Interacción con objetos rígidos texturizados utilizando el dispositivo Novint Falcon.

Secuencia de contacto con la esfera utilizando dispositivo *Novint Falcon* (visualizada en la Figura 4.6).

1. En el primer cuadro de la izquierda se puede notar la primera interacción entre el *avatar* y la esfera.
2. Mientras que en el cuadro superior de la derecha se visualiza que el *avatar* sigue realizando contacto con la esfera pero ahora en otro punto.
3. Por último en el cuadro inferior de la derecha se puede notar al *avatar* dentro del espacio virtual sin hacer contacto con ningún objeto.

Secuencia de contacto con los objetos rígidos texturizados utilizando dispositivo *Novint Falcon* (visualizada en la Figura 4.7).

1. En el cuadro de la izquierda se visualiza que el *avatar* está haciendo contacto con uno de los objetos texturizados.
2. Así mismo en el cuadro de la derecha se puede apreciar la misma acción con el otro objeto.

4.3. Sistema de realidad virtual con interacción háptica para objetos deformables utilizando el robot *Geomagic Touch*

A diferencia de los sistemas anteriores, en este último se presenta un método experimental que busca un objetivo diferente ya que en vez de adentrarse en un ambiente virtual e interactuar con objetos rígidos, se desea hacer contacto con un objeto deformable con el fin de poder experimentar los cambios físicos que llegan a sufrir éstos al momento de la interacción. Para este sistema, se utilizó nuevamente el dispositivo háptico *Geomagic Touch* como interfaz háptica ya que se contaba con las librerías necesarias para habilitar dicha característica. A comparación de los sistemas anteriores, éste cuenta con un desarrollo gráfico más simple, en parte porque se trata de una primera prueba experimental y además de que las librerías utilizadas no permiten realizar un desarrollo más complejo como en los casos anteriores. Sin embargo, de igual forma se presentarán los pasos utilizados en cada una de las fases que comprenden dicho desarrollo.

4.3.1. Renderizado visual

Tal como se citó previamente, este sistema tiene ciertas discrepancias a comparación de sus predecesores, una de ellas es la forma en que se genera el renderizado visual de los cuerpos, ya que no utiliza primitivas para generar los objetos. En cambio, hace el uso del modelado de objetos en 3D utilizando mallas tridimensionales. Para lograrlo se utilizó *Blender*, el cual es un programa especializado en el modelado, iluminación, renderizado y creación de gráficos tridimensionales. Para este sistema se buscó mostrar la interacción con un cuerpo sencillo por lo que se utilizó una esfera, la cual fue modelada tal como se puede ver en la Figuras 4.8 y 4.9. Se puede apreciar tanto la malla que comprende dicha figura como la representación que tiene en su renderizado final.

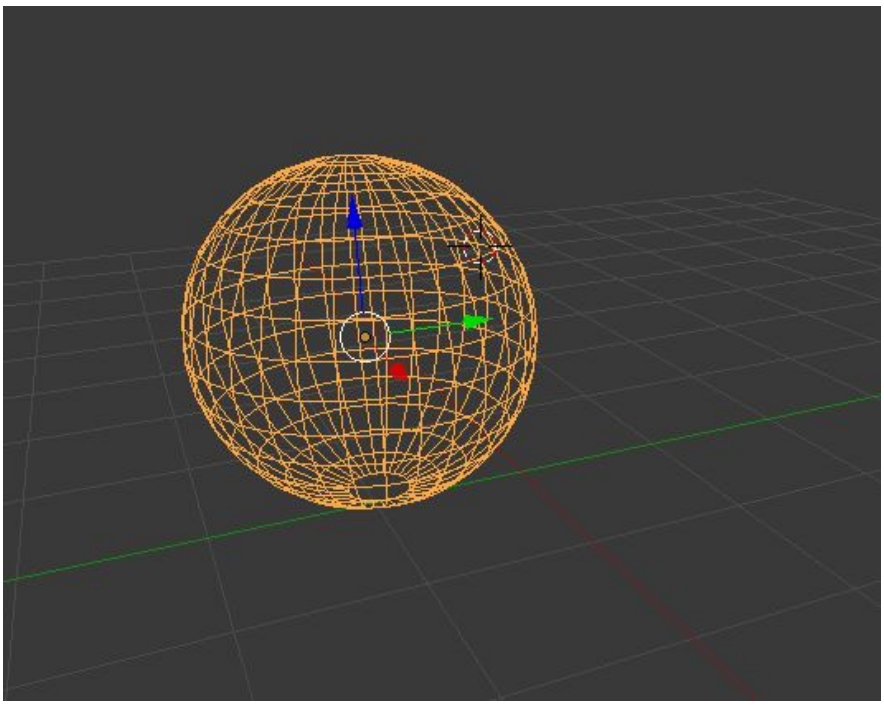


Figura 4.8: Esfera modelada a través de mallas usando Blender.

Una vez que se tiene el modelo, éste se extrae en un archivo *OBJ* o *3DS* tal como se indica en la Sección 3.2.2 ya que son formatos utilizados para objetos tridimensionales y que además son compatibles con las librerías utilizadas en la *API* del dispositivo y por *OpenGL*. A continuación se presenta el pseudocódigo utilizado para la renderización gráfica del sistema, así como el resultado del renderizado final, el cual puede apreciarse en la Figura 4.10.

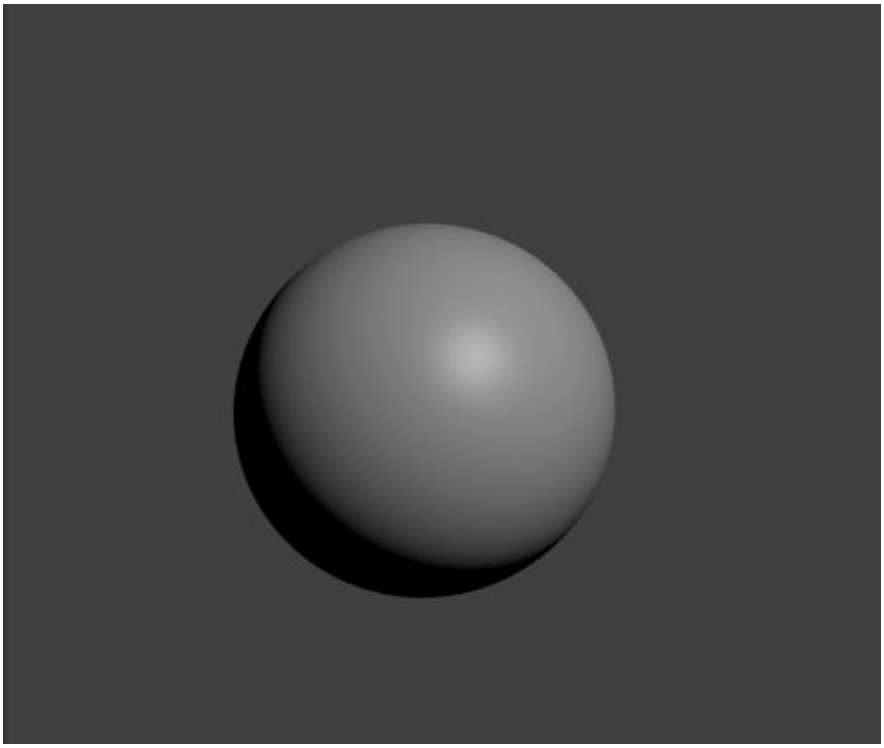


Figura 4.9: Renderización de esfera en 3D.

Pseudocódigo del renderizado gráfico

1. Incluir las librerías y cabeceras necesarias para el renderizado gráfico que proporciona la *API* del dispositivo.
2. Definir tanto las función principal del programa así como los *callback* utilizados en el renderizado gráfico.
3. Se crea la ventana en donde se verá reflejado el sistema.
4. Se crea el objeto utilizando el archivo creado en *Blender* y la librería *Trimesh*, la cual contiene todas las características necesarias para utilizar objetos 3D.
5. Una vez creado el objeto tipo *Trimesh*, se le agregan parámetros físicos que le de un mayor realismo, como iluminación y color.
6. En el *callback* principal se manda a llamar al objeto creado con el fin de poder visualizarlo en pantalla.

Estructura general del código utilizado en el renderizado gráfico del programa.

1. Se incluyen las librerías necesarias.
   Include <QHHeadersGLUT.h>
2. Se definen las funciones de *callback*.
   void GraphicsCallback(void).
3. Dentro de la función principal se crean los objetos gráficos.
   int main(int argc, char *argv[])
   - Se crea la ventana GLUT
     QHGLUT* DisplayObject = new QHGLUT(argc,argv);
   - Se crea el espacio de trabajo.
     DeviceSpace* OmniSpace = new DeviceSpace;
     DisplayObject− >tell(OmniSpace);
   - Se le da color al espacio de tabajo.
     DisplayObject− >setBackgroundColor(0.0, 1.0, 1.0);
   - Se carga el objeto tipo esfera.
     TriMesh* Esfera= new TriMesh(”Models/ese.3ds”);
   - Se le dan características físicas y nombre al objeto.
     Esfera− >setTranslation(3.0, 0.0, 0.0);
     Esfera− >setScale(.7);
     Esfera− >setName(.Esfera”);
     Esfera− >setTexture(”Models/y.jpg”);
   - Se manda a renderizar el objeto.
     DisplayObject− >tell(Esfera);

4.3.2. Renderizado háptico

A diferencia de los sistemas anteriores, el renderizado háptico en este caso depende en gran medida del renderizado gráfico ya que se utilizan propiedades físicas presentes en la clase *Trimesh* como son rigidez y amortiguamiento, los cuales son parámetros que permiten evaluar la fuerza en un sistema de masa-resorte-amortiguador, que tal como se vio en la Sección 2.2.2 de esta tesis, es uno de los modelos más utilizados en cuestión de renderizado háptico. Así mismo, para que el objeto en cuestión pueda presentar deformaciones al momento de la interacción, se debe habilitar otro parámetro denominado como *dinamismo*, siendo éste el responsable de que el objeto pase de ser rígido

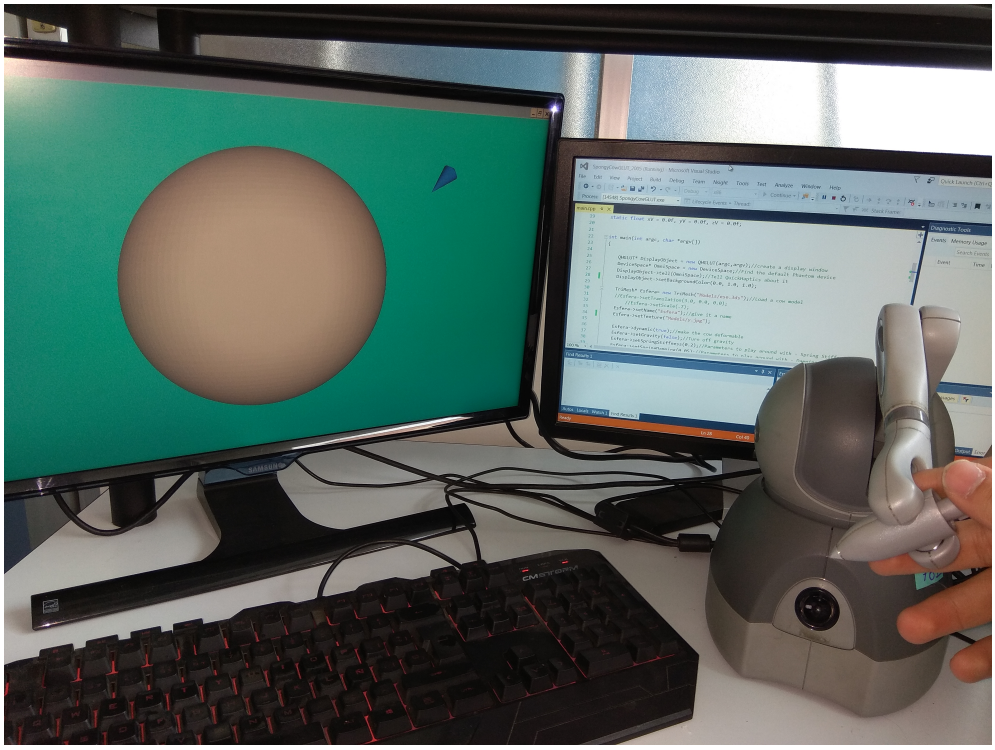


Figura 4.10: Sistema de realidad virtual con interacción háptica para objetos deformables utilizando el robot *Geomagic Touch*.

a deformable. Para visualizar de mejor forma el desarrollo del sistema, a continuación se presentará tanto el pseudocódigo como la estructura general del código implementado. Así mismo en la Figura 4.10, se puede observar el renderizado final mientras que en la Figura 4.11 se puede apreciar la secuencia de contacto que el usuario tiene con dicho objeto, así como una ligera perturbación en su superficie debido a la deformación que el efector final causa en ella, la cual no se presentaba en los sistemas anteriores.

Pseudocódigo del renderizado háptico en el programa principal.

1. Incluir las librerías y cabeceras necesarias para el renderizado háptico que proporciona la *API* del dispositivo.
2. Definir las funciones *callback* en las cuales se realizarán los renderizados hápticos.
3. Definir los objetos a través del renderizado gráfico.
4. Habilitar la característica de dinamismo, la cual permitirá que el objeto pueda ser deformable.

Estructura general del código utilizado en el renderizado háptico del programa.

1. Se incluyen las librerías necesarias
   Include <QHHeadersGLUT.h>
2. Se definen las funciones de *callback*
   void MotionCallback(unsigned int ShapeID);
3. Dentro de la función principal donde se crean los objetos gráficos, se habilita que el objeto sea deformable.
   int main(int argc, char *argv[])
   - Se carga el objeto tipo esfera.
     TriMesh* Esfera= new TriMesh("Models/ese.3ds");
   - Se habilita el dinamismo del objeto y se agregan parámetros que permitan el renderizado de fuerzas.
     Esfera− >dynamic(true);
     Esfera− >setGravity(false);
     Esfera− >setSpringStiffness(0.2);
     Esfera− >setSpringDamping(0.05);
     Esfera− >setMass(0.01);
   - Se manda a renderizar el objeto.
     OmniSpace− >motionCallback(MotionCallback, Esfera);

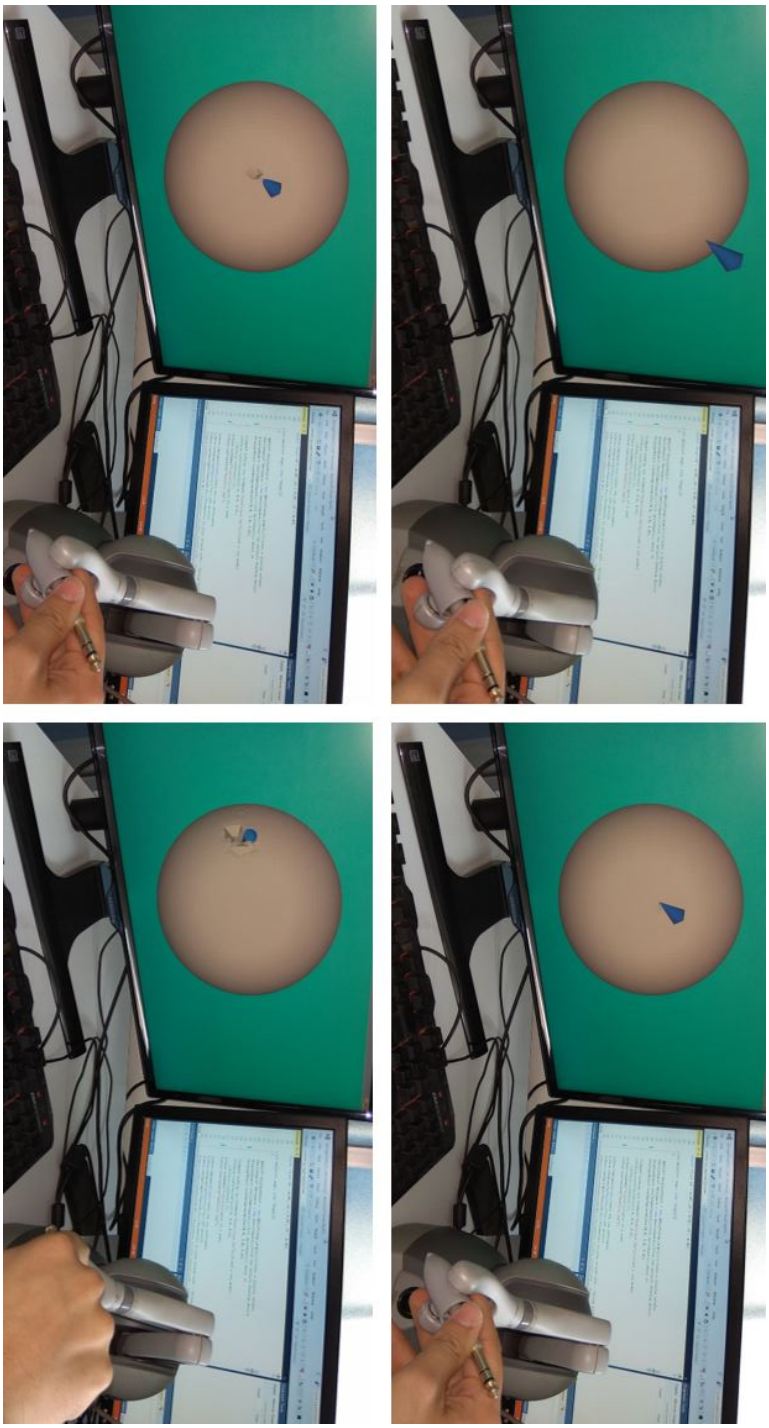


Figura 4.11: Secuencia de contacto con esfera deformable utilizando el dispositivo *Geomagic Touch*.

Secuencia de contacto con esfera deformable utilizando el dispositivo *Geomagic Touch* (visualizada en la Figura 4.11).

1. En el cuadro superior derecho se puede apreciar que el *avatar* está apunto de interaccionar con la esfera.
2. En el cuadro consiguiente, el *avatar* cuenta con una nueva posición de contacto con la esfera.
3. Ya en el cuadro inferior izquierdo se aprecia el contacto entre la esfera y el *avatar* así como la deformidad que se genera en la superficie del objeto.
4. Por último en el cuadro inferior derecho se aprecia que el *avatar* se encuentra en otra posición ejerciendo una fuerza mayor en la esfera, por lo que se genera una deformación más evidente en su superficie.

4.4. Comentarios del capítulo

De manera general las aplicaciones obtenidas cumplen correctamente los objetivos propuestos. Así mismo tanto el robot *Geomagic Touch* como el robot *Novint Falcon* funcionan correctamente como interfaces hápticas al momento de interactuar con los sistemas. De igual forma se puede apreciar que la implementación final cuenta con una estructura sólida en cuanto a programación por lo que dichos sistemas pueden servir de base en busca de desarrollos mucho más complejos.

Capítulo 5

Conclusiones

Uno de los principales objetivos de esta tesis era el lograr relacionar dos áreas de desarrollo (Realidad Virtual y Háptica) en un solo procedimiento, creando así una brecha entre ellas con el fin de implementar aplicaciones mucho más complejas e interactivas a las que con anterioridad se habían desarrollado dentro del Laboratorio de Robótica. Para obtener dichos resultados este proyecto se trabajó desde dos frentes. El primero se encargó de la parte gráfica, enfocándose en su totalidad en el desarrollo visual de los objetos a interactuar, para esto se tomaron en cuenta las distintas fases que necesita un sistema de este tipo, las cuales van desde el modelado hasta la renderización final. Mientras tanto, el segundo frente de trabajo se enfocó en lograr la interacción entre el sistema gráfico y el usuario, para esto se utilizaron bases matemáticas enfocadas en los principios básicos de la teoría control, los cuales fueron pieza clave al momento de simular las fuerzas de interacción entre el usuario y los objetos virtuales. No obstante es conveniente señalar que dicha interacción no se hubiera podido lograr sin el uso de algún dispositivo háptico que permitiera simular las fuerzas de contacto, en este caso, los robots *Geomagic Touch* y *Novint Falcon*.

Teniendo en cuenta lo anterior, es necesario recalcar que otro punto importante en el desarrollo de este trabajo fue entender de manera clara y concisa el funcionamiento de cada robot, así mismo sus diversas características y la forma en como enlazarlos con las herramientas que permitieron desarrollar la parte gráfica del sistema. Sin embargo, las complicaciones que se tuvieron fueron la incompatibilidad entre los robots ya que cuentan con licencias diferentes, por lo que no se pudo realizar una aplicación que pudiera interactuar con ambos dispositivos. Sin embargo, dicho obstáculo se podría contrarrestar si se creara un librería compatible para los dos robots.

Finalmente, con este trabajo no solo se logró presentar un sistema con el cual se pueda ejemplificar la interacción háptica dentro de un entorno virtual, si no que se obtuvieron tres desarrollos con características diferentes, en los cuales se pueden apreciar y experimentar esta nueva herramienta tecnológica que cuenta con un gran potencial por explorar.

5.1. Trabajo futuro

Teniendo en cuenta que esta serie de sistemas virtuales con interacción háptica se lograron obtener de manera exitosa, sabiendo que está nueva área tecnológica se encuentra en pleno desarrollo y que se espera un crecimiento e impacto de gran importancia, algunas propuestas a implementar en un futuro son:

- Utilizar los programas obtenidos como base de nuevos proyectos con mayor complejidad, ya sea con mayor cantidad o diversos tipos de objetos virtuales con los cuales interactuar.
- Adentrarse y desarrollar de una forma más compleja la interacción háptica con objetos deformables.
- Realizar una aplicación con un objetivo específico o área de trabajo especializada tal como simuladores de medicina, programas educativos para niños, videojuegos, entre otras.

Particularmente en el Laboratorio de Robótica, donde se trabaja de forma esencial con esquemas de control avanzados, se pretende explorar más a fondo la manera de incluir realidad virtual en las aplicaciones desarrolladas así como los retos que ello implica. Este trabajo de tesis es un primer esfuerzo para unir dos áreas del conocimiento aparentemente distintas que al unirse conllevan un potencial enorme.

Apéndice A

Manual de configuraciones entre el robot *Geomagic Touch* y *Visual Studio*

Antes de mostrar la configuración de cada robot es importante comentar las características con las que cuenta el equipo en donde se realizaron los desarrollo, las cuales son:

- Sistema operativo *Windows 10* de 64 bits.
- Procesador *Intel core i7*.
- Memoria RAM de 16 GB.

Así mismo se debe recalcar que se utilizaron las versiones *Microsoft Visual Studio 2015* y *OpenGL 2.0* para el desarrollo de los sistemas.

A.1. Configuración *Visual Studio*

Para poder utilizar de forma correcta esta herramienta es necesario tener en cuenta una serie de características que se deben cumplir.

1. Como primer punto se deben conocer las propiedades generales del *software* por ejemplo la versión que se está utilizando o la plataforma en la que se realiza el desarrollo. Dichos puntos se pueden notar en un cuadro de propiedades generales, como el que se muestra en la Figura A.1.
2. Una vez que se conocen las características generales de la herramienta se procede a enlazar los archivos que permitan generar las diversas características del sistema como la renderización háptica o el renderizado visual, entre otras. Dichos archivos deben estar juntos en una ruta específica ya que es mucho más sencillo agregarlos en conjunto. Para lograrlo se debe hacer uso del mismo cuadro descrito en el

punto anterior, sin embargo se debe configurar tal como se muestra en la Figura A.2.

3. Así mismo se deben agregar dependencias adicionales que se utilizan en algunas librerías. Del mismo modo que en los puntos anteriores se hace uso del cuadro de propiedades del programa tal como se puede ver en la Figura A.3.

4. Una vez realizado estos pasos se obtiene una configuración óptima del *software*, sin embargo, para lograr visualizar al sistema como en la Figura A.4 se deben configurar y calibrar el dispositivo háptico tal como se muestra a continuación

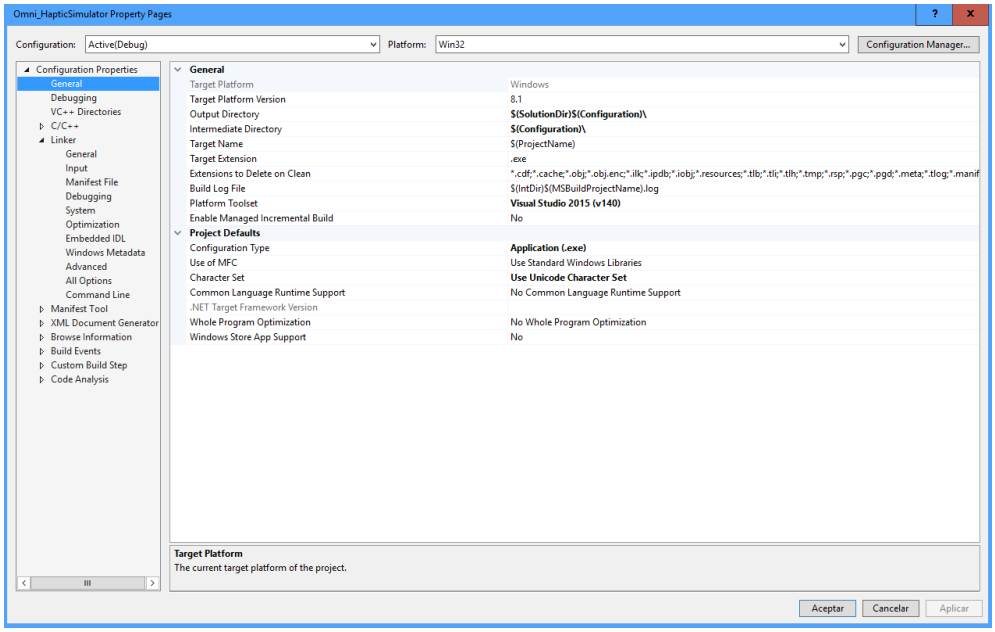


Figura A.1: Propiedades generales de *Visual Studio*.

A.2. Configuración *Geomagic Touch*

Para hacer uso de este robot se debe realizar una configuración inicial, la cual se muestra a continuación:

1. Primeramente se debe conectar el robot tanto a la corriente eléctrica como a la computadora por medio de un cable *Ethernet*.

2. Una vez conectado el robot se debe abrir una aplicación propia del robot llamada *Touch Setup*, la cual permitirá vincular exitosamente el dispositivo háptico con el equipo de computo tal como se puede ver en la Figura A.5.

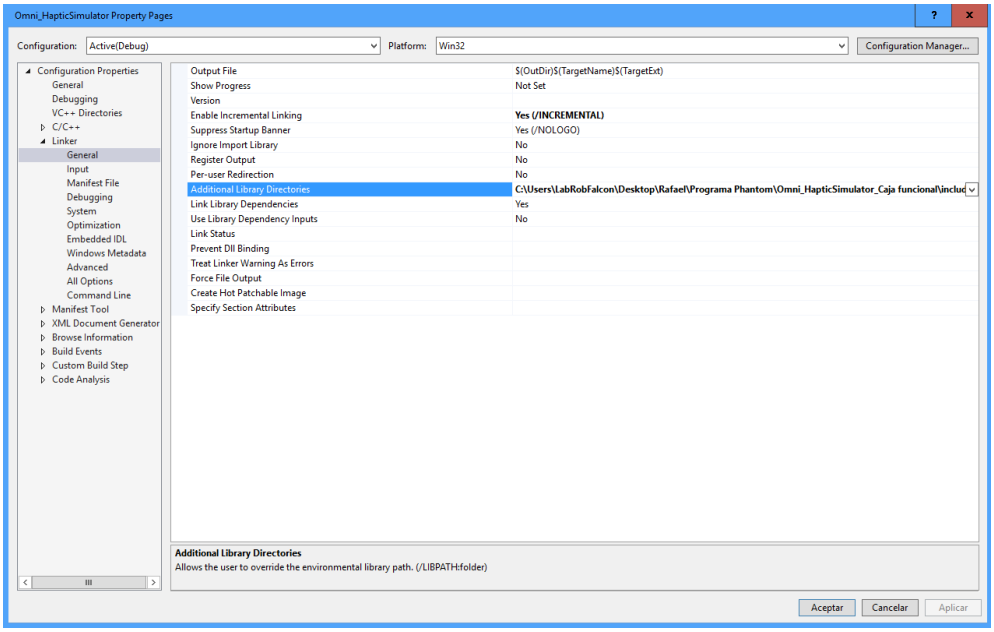


Figura A.2: Selección de la ruta de archivos.

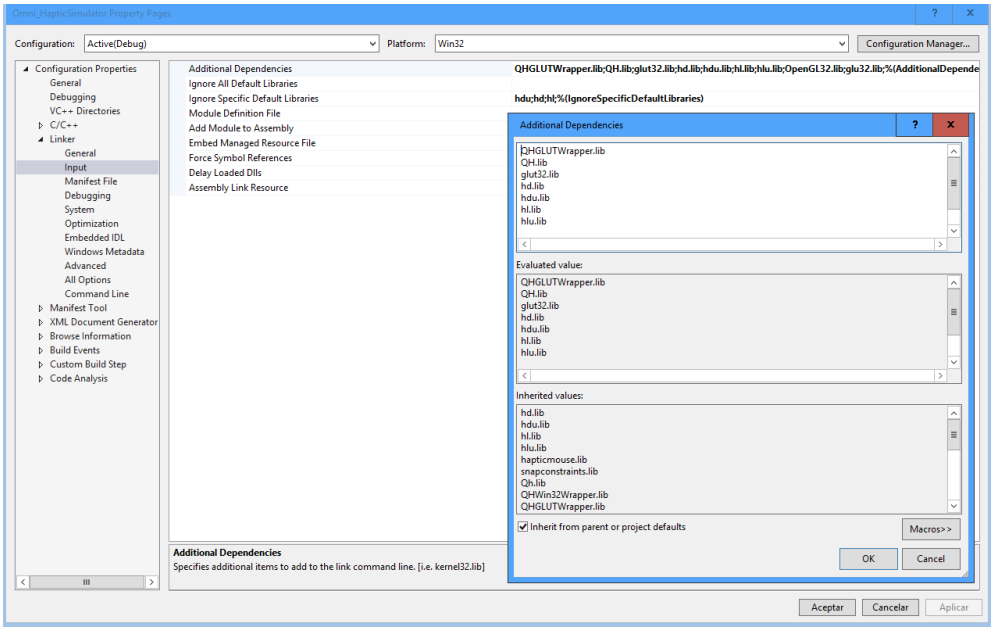


Figura A.3: Inclusión de las dependencias adicionales.

3. Ya emparejado el robot éste debe ser calibrado antes de utilizarlo como interfaz háptica, para esto se hace uso de otra aplicación propia del dispositivo denominada

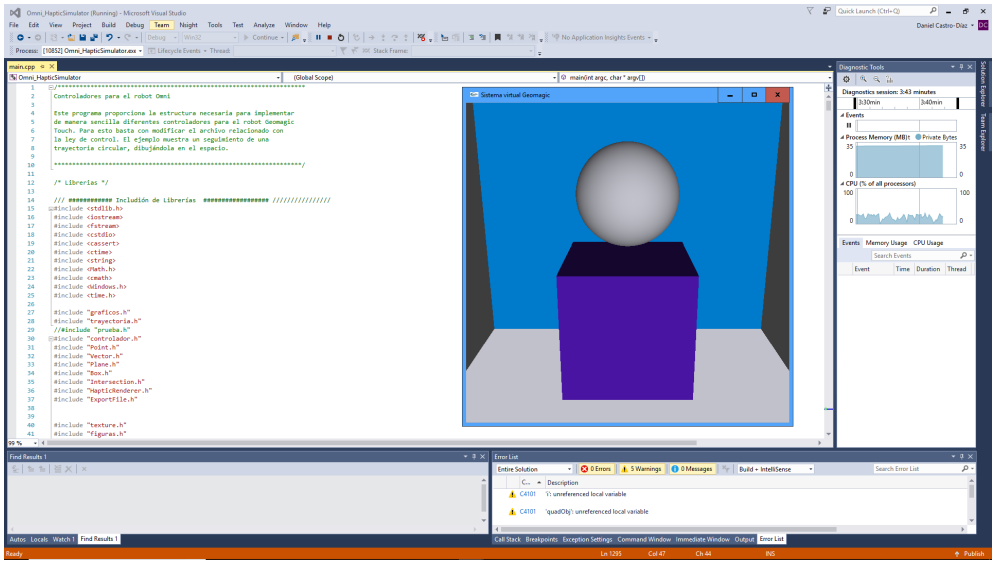


Figura A.4: Ejecución del sistema virtual.

Touch Diagnostic tal como se muestra en la Figura A.6.

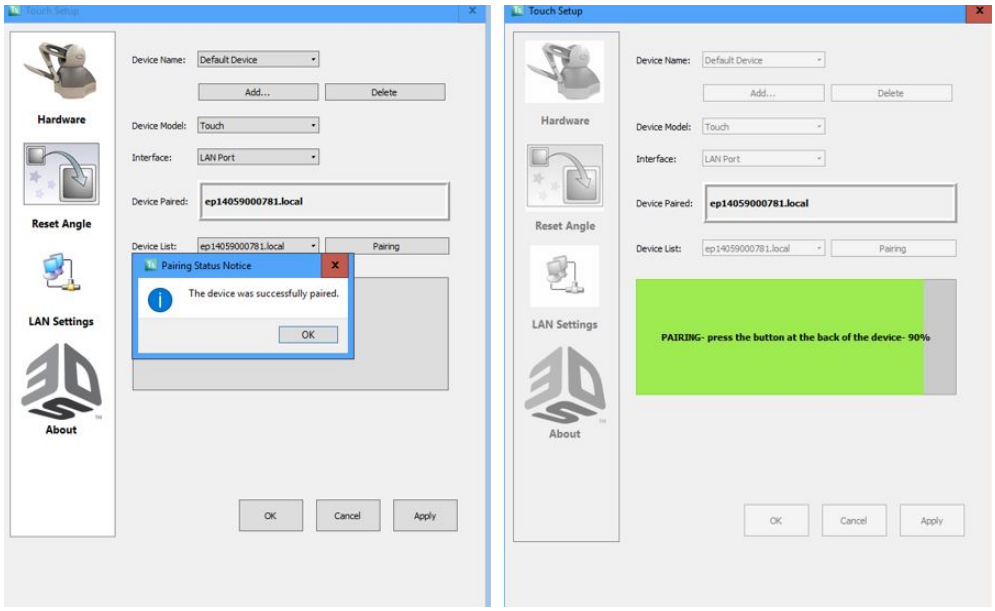


Figura A.5: Vinculación del robot *Geomagic Touch* utilizando *Touch Diagnostic*

Secuencia de vinculación del equipo de cómputo y el robot *Geomagic Touch* (visualizada en la Figura A.5).

1. En el cuadro de la izquierda se puede visualizar el proceso de conexión entre el dispositivo háptico y el equipo de cómputo.
2. Mientras que en el cuadro de la derecha se observa que el robot ha sido conectado correctamente.

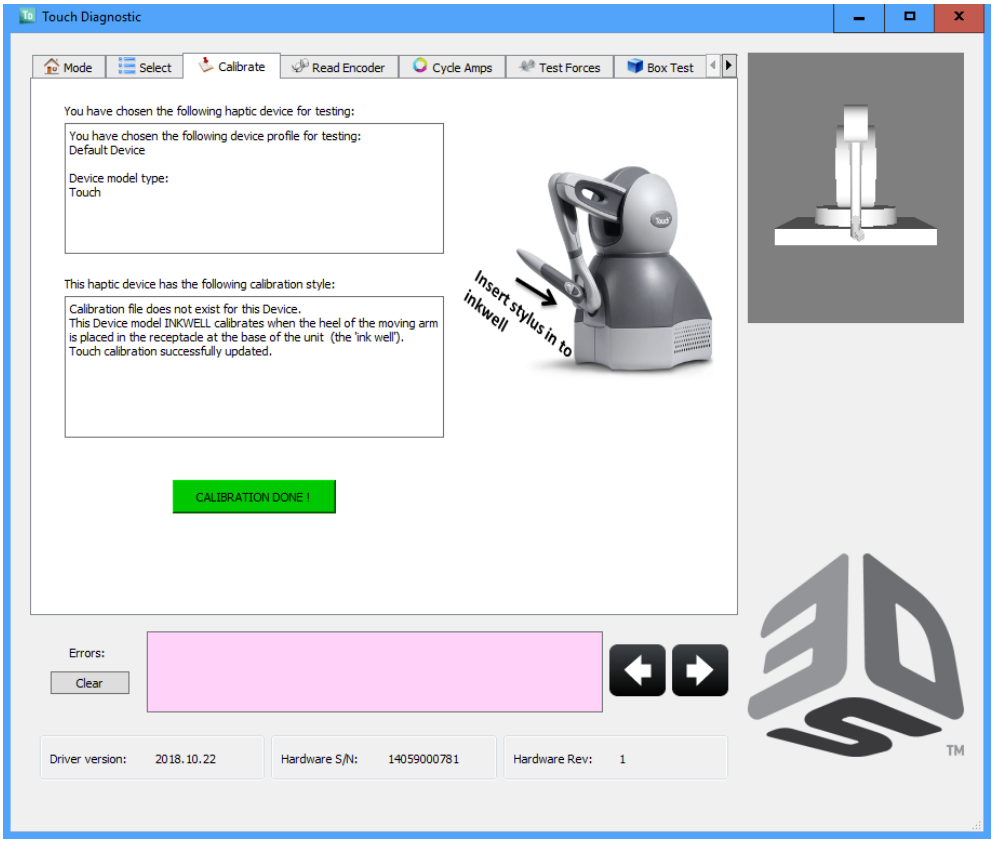


Figura A.6: Calibración del dispositivo háptico utilizando *Geomagic Diagnostic*.

Apéndice B

Manual de configuraciones entre el robot *Novint Falcon* y *Visual Studio*

Al igual que en el manual anterior, se presentarán las configuraciones necesarias para utilizar el dispositivo *Novint Falcon* con *Visual Studio*, sin embargo en este caso no es necesaria una configuración y calibración del robot.

B.1. Configuración de *Visual Studio*

1. Así como en el primer punto del manual anterior, se deben conocer las propiedades generales del *software* por lo que se debe hacer uso del cuadro de propiedades, el cual se puede ver en la Figura B.1.

2. Ya que se conocen las características se procede a enlazar los archivos que permitan generar las características del sistema tal como se muestra en la Figura B.2.

3. Así mismo se agregan las dependencias adicionales que se lleguen a utilizar, tal como se puede visualizar en la Figura B.3.

4. Una vez realizado estos pasos se obtiene una configuración óptima del *software*, tal como se muestra en la Figura B.4. Para lograr dicho resultado es de suma importancia que el dispositivo *Novint Falcon* ya se encuentre conectado tanto a la corriente eléctrica como al equipo de cómputo.

5. Una vez que se ejecutan las aplicaciones, el robot debe colocarse en una posición inicial la cual se obtiene empujando hacia el centro el efector final del dispositivo y posteriormente extenderlo hacia el lado contrario.

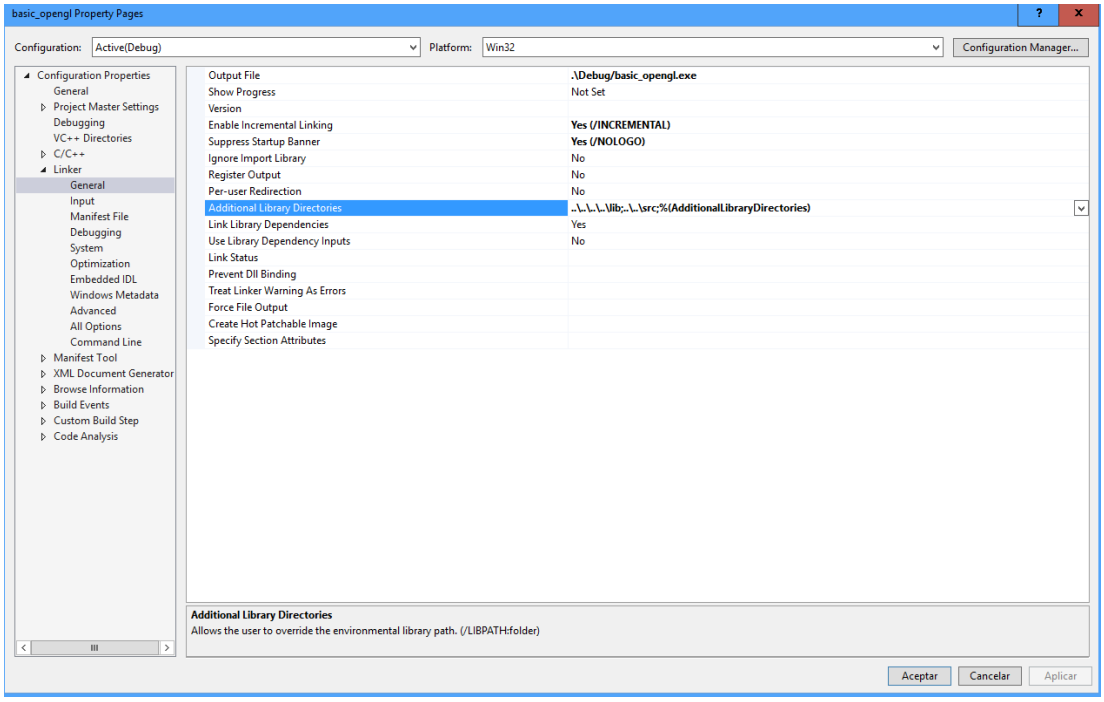


Figura B.1: Propiedades generales de *Visual Studio*.

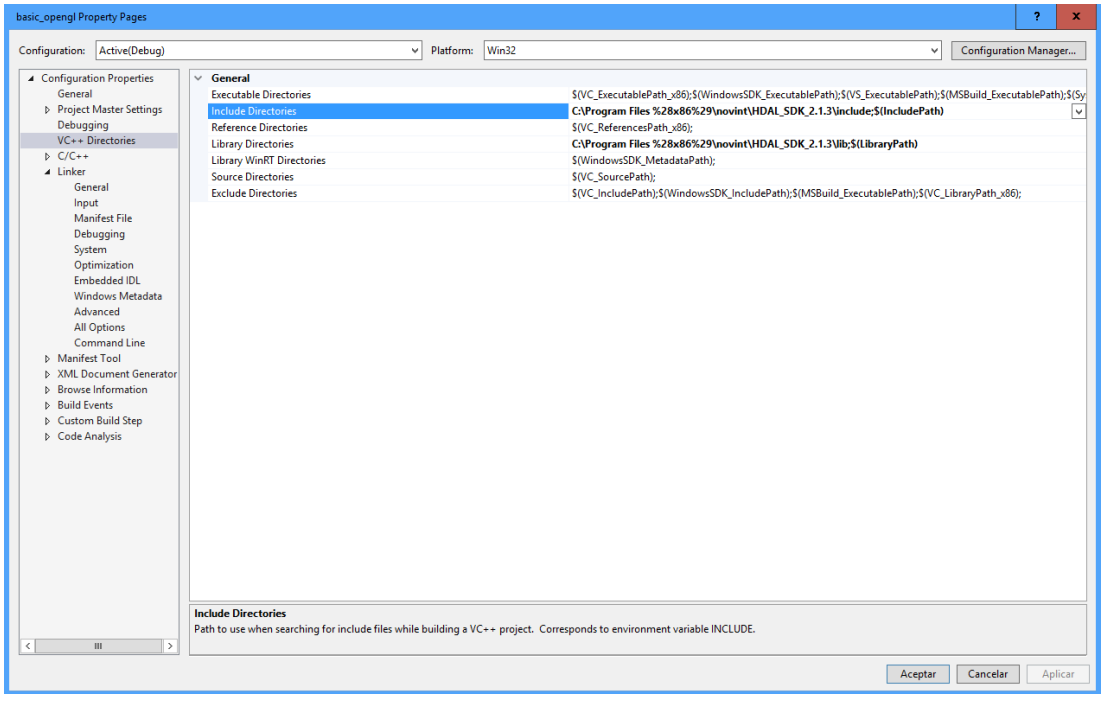


Figura B.2: Propiedades generales de *Visual Studio*.

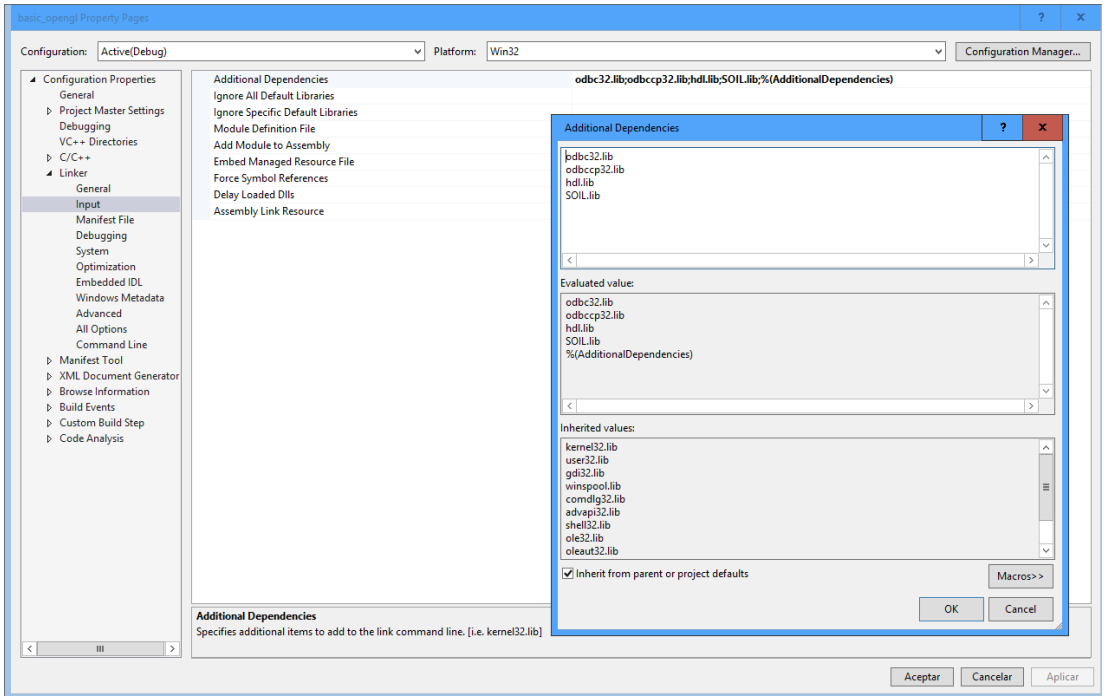


Figura B.3: Propiedades generales de *Visual Studio*.

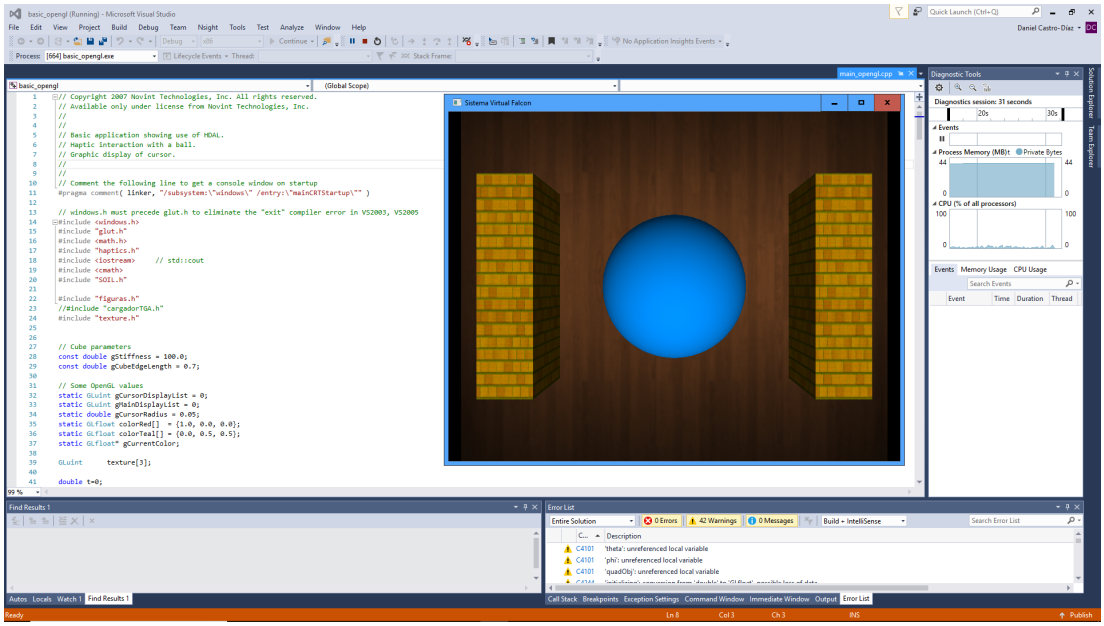


Figura B.4: Propiedades generales de *Visual Studio*.

Bibliografía

[1] Barbagli, F., Salisbury, K., and Conti, F. (2004). *Haptic Rendering: Introductory Concepts*. Standford University and University of Siena. 14, 16, 26, 29

[2] Dervaux, F., Peterlik, I., Dequidt, J., Cotin, S., and Duriez, C. (2015). *Haptic rendering of interacting dynamic deformable objects simulated in real-time at different frequencies*. HAL. 33

[3] Ewatt, D. M. (2015). *Oculus Rift: desafiando a la realidad*. Forbes México. 8

[4] García, C. L. (2014). *Control de fuerza sobre superficies Virtuales. Tesis de Maestría*. UNAM. 30

[5] González, C. M., Gracia, M. A. B., Grasa, L. S., and Romero, D. S. M. (2015). *Análisis: Motores gráficos y su aplicación en la industria*. Instituto Tecnológico de Aragón. 17

[6] Govil-Pa, S. (2004). *Principles of Computer Graphics Theory and Practice Using OpenGL and Maya*. Springer. 17, 18, 19, 20, 21, 22, 23, 24, 25, 26

[7] Martínez, E. F. (2018). *Diseño e Implementación de un mecanismo en configuración de muñeca esférica para experimentos de control de fuerza en superficies virtuales. Tesis de Licenciatura*. UNAM. 36, 38, 39

[8] Mihelj, M. and PodobnikMark, J. (2012). *Haptics for Virtual Reality and Teleoperation*. Springer. 32

[9] Montero, R. (1996). *Realidad Virtual*. Autores Científicos Técnicos y Académicos (ACTA). 4

[10] Mukundan, R. (2012). *Advanced Methods in Computer Graphics with examples in OpenGL*. Springer. 26, 27, 28

[11] Norton, R. L. (2009). *Diseño de maquinaria*. McGrawHill. 35

[12] Pinto, M. L. S. (2009). *Análisis e implementación de una interfaz háptica en entornos virtuales. Tesis de Maestría*. UNAL. 12, 13, 41, 44, 45, 46, 47, 48, 49, 50, 51

[13] Pérez, F. J. M. (2011). *Presente y Futuro de la Tecnología de la Realidad Virtual.* Revista Creatividad y Sociedad. 2, 9, 12

[14] Romero, F. G. (2005). *Técnicas para la simulación de objetos deformables. Tesis de Licenciatura.* Universidad de Málaga. 4, 10, 11, 12

[15] Sherman, W. R. and Craig, A. B. (2002). *Understanding Virtual Reality: Interface, aplication and design.* Elsevier. 4, 5, 6, 7, 8, 9

[16] Sánchez, M. V. (2002). *La Realidad Virtual como herramienta en la enseñanza de la anatomía humana para el 4to y 5to grado de nivel primario.* UDLAP. 11

[17] Spong, M. W., Hutchinson, S., and Vidyasagar, M. (2006). *Robot modeling and control.* Springer. 38

[18] Systems, D. (2015). *Open Haptics Toolkit version 3.3.0: Programmers Guide.* 3D Systems Inc. 42

[19] Torres, I. R. (2017). *Puesta en funcionamiento del robot paralelo Novint Falcon. Tesis de Licenciatura.* UNAM. 28, 31, 40, 57

[20] Villavicencio, J. A. C. (2015). *Display háptico de sistemas dinámicos virtuales y teleoperados sujetos a restricciones holonomónicas. Tesis de Licenciatura.* UNAM. 30, 31, 38, 40