ENTORNO GRÁFICO DE UN ENTRENADOR VIRTUAL DE PRÓTESIS DE MANO

JULIAN GARCÍA ORTIZ
DAVID ALEJANDRO VALLEJOS ORTIZ

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
INGENIERÍA EN AUTOMÁTICA INDUSTRIAL

POPAYÁN, julio de 2009

ENTORNO GRÁFICO DE UN ENTRENADOR VIRTUAL DE PRÓTESIS DE MANO

JULIAN GARCÍA ORTIZ
DAVID ALEJANDRO VALLEJOS ORTIZ

Proyecto de grado presentado como requisito para optar al título de Ingenieros en Automática Industrial

Director:
ING. ELENA MUÑOZ ESPAÑA

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
INGENIERÍA EN AUTOMÁTICA INDUSTRIAL

POPAYÁN, julio de 2009

Nota de aceptación:

_____
_____
_____
_____
_____
_____

_____
Firma del presidente del jurado

_____
Firma del jurado

_____
Firma del jurado

Popayán, noviembre de 2022

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

RESUMEN

El proyecto “Entorno gráfico de un entrenador virtual de prótesis de mano”, representa en 3D el modelo dinámico de una prótesis de mano robótica de nueve grados de libertad y reproduce unas trayectorias articulares dadas con el fin de desarrollar el agarre de un objeto presente en el entorno tridimensional. La herramienta desarrollada permite analizar, identificar y visualizar las habilidades de la mano considerada para el desarrollo de tareas de agarre de objetos.

El entorno gráfico se implementó en herramientas software totalmente libres con características de tiempo real, utilizando el lenguaje de programación Ada y su entorno de desarrollo (GNAT). En este trabajo se crea un vinculo entre *OpenGL* una de las interfaces 3D estándar más conocidas y el lenguaje de programación Ada, dando así la posibilidad de tener un entorno virtual con características de tiempo real, multiplataforma y libre.

A partir del entorno gráfico desarrollado se podrá construir un entrenador virtual libre con el fin de que el paciente pueda desde cualquier computador y solo con la instalación de un hardware de acondicionamiento de señales de bajo costo, entrenarse y mantener sus conexiones nerviosas en buen estado, preparándolas para la implantación de la prótesis real.

1 INTRODUCCIÓN.

La tarea de sujetar y trasladar objetos es algo que desde los primeros años de vida los seres humanos realizan con precisión. Esto que parece simple, implica un complejo proceso de transmisión de información desde el cerebro hasta el órgano ejecutor. La mano representa una de las estructuras anatómicas más sorprendentes del ser humano, es una estructura compleja, tanto en construcción como en funcionamiento. Es un órgano de prensión y de movimientos finos, de gran sensibilidad, de discriminación precisa y destreza exquisita (1).

En la actualidad se están desarrollando prótesis robóticas cuyo enfoque está basado en las señales electromiográficas (EMGs - Señales eléctricas producidas por un músculo durante el proceso de contracción y relajación) captadas en los músculos de los miembros superiores del paciente y que son utilizadas para enviar órdenes hacia la prótesis (2).

Los amputados pueden aprender rápidamente a ejecutar varios movimientos imaginarios en una mano virtual de un computador, lo que hace bastante prominente el futuro del control de prótesis mioeléctricas (3). El entrenamiento en el manejo de prótesis mioeléctricas sigue siendo uno de los factores determinantes en la aceptación de las mismas (4), (5), (6). Un estudio sobre rehabilitación de pacientes amputados (7) encontró que la tasa de aceptación en el uso de éstas se duplicaba si los pacientes eran entrenados apropiadamente. Esto ha originado que muchas investigaciones se enfoquen en el desarrollo de prótesis virtuales (8), (9) debido a que permiten el entrenamiento y valoración a un costo mucho más bajo del requerido que cuando se emplean prótesis reales.

Los trabajos de modelación y simulación 3D del cuerpo humano son pocos y más aún los referentes a manos humanas, éstos han tenido su auge en los últimos años debido al avance de las computadoras con más capacidad para poder realizar simulaciones mecánicas (10). En general, los trabajos de simulación son la primera aproximación para buscar soluciones que tratan de recuperar la funcionalidad de la mano humana mediante la implantación de prótesis activas. Actualmente existen entornos virtuales con este fin pero desarrollados con herramientas comerciales, lo cual implica al paciente un costo extra de adquisición (2). En ese sentido este trabajo desarrolló la aplicación con herramientas software totalmente libres con características de tiempo real y con interfaces amigables para el usuario. Es así, que para el desarrollo de la aplicación se escogió el

lenguaje de programación Ada debido a su gran flexibilidad en cuanto a tiempo real se refiere, ya que su entorno de desarrollo (GNAT) es uno de los más potentes y completos entre los IDE libres disponibles. Otro aspecto relevante es que actualmente existen pocos proyectos relacionados con aplicaciones 3D escritos en lenguaje Ada; este trabajo pretende además crear un vinculo entre *OpenGL* una de las interfaces 3D estándar más conocidas y el lenguaje de programación Ada, dando así la posibilidad de tener un entorno virtual con características de tiempo real, multiplataforma y libre.

Este proyecto, se enmarca dentro del área de desarrollo del Grupo en Automática Industrial, que actualmente trabaja en el proyecto macro denominado "Prótesis mioeléctrica para amputados de mano" en el cual se pretende en su primera fase desarrollar una prótesis robotizada virtual que represente con la mayor precisión las capacidades motoras de la mano natural, la cual será controlada por las intenciones de movimiento que son determinadas a partir de señales electromiográficas tomadas de pacientes.

En el presente proyecto, se desarrolla la parte correspondiente al entorno gráfico del entrenador virtual de prótesis de mano, la cual representa en 3D el modelo dinámico de una prótesis de mano robótica y reproduce unas trayectorias articulares dadas con el fin de desarrollar el agarre de un objeto presente en el entorno tridimensional, de esta forma se obtiene una herramienta que permite analizar, identificar y visualizar las habilidades de la mano considerada para el desarrollo de tareas de agarre de objetos. La Figura 1.1 ilustra los elementos de entrada y salida del entorno gráfico.

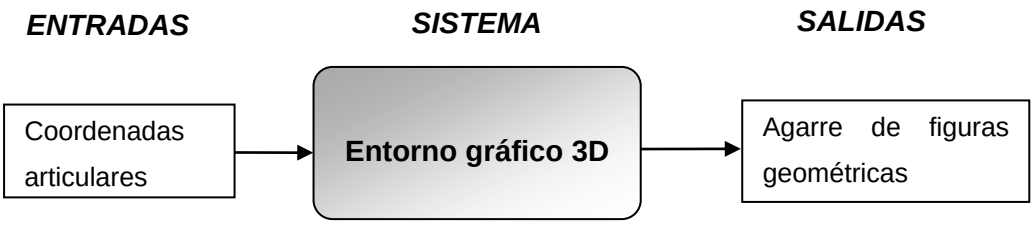


Figura 1.1 Entradas y salidas del sistema.
Fuente: Elaboración Propia.

El proyecto busca afianzar los aportes que generan la robótica y los sistemas computacionales en el campo médico mediante el desarrollo de herramientas que permiten brindar solución a problemas de discapacidad, en este caso por amputación de algún miembro en la persona. Igualmente con este proyecto se fortalece el trabajo interdisciplinario al proponer soluciones a problemas de la medicina con calidad y proyección social, ofreciendo alternativas accesibles, confiables y a bajo costo de acuerdo con las necesidades del entorno.

El presente documento se encuentra organizado en siete capítulos de la siguiente manera:

En el capítulo 2, se realiza el estudio y análisis de las diferentes herramientas de modelado para aplicaciones tridimensionales tanto libres como comerciales, e igualmente se da a conocer el software de modelado y simulación 3D empleado para el desarrollo del proyecto.

En el capítulo 3, se trata el tema correspondiente a la anatomía de la mano humana y su capacidad de aprehensión de objetos, se describen las variadas clasificaciones de agarres que puede llevar a cabo la mano humana, se da a conocer además la utilidad de las prótesis como sustituto de la extremidad perdida y para finalizar el tema se describe la técnica de agarre de objetos empleada en la herramienta software desarrollada.

El capitulo 4 se enfoca en la arquitectura, diseño e implementación del sistema software, dándose a conocer el concepto de tiempo real y describiendo su uso en la aplicación, se describe una de las técnicas empleadas para el diseño de sistemas en tiempo real y se muestra en forma breve el empleo de dicha técnica para el diseño de la aplicación.

El capitulo 5 muestra los resultados obtenidos en el proyecto, básicamente se indica la forma en la cual se validó el entorno virtual desarrollado mediante la ejecución de diversas simulaciones para diferentes tipos de agarre.

En el capítulo 6, se dan a conocer las conclusiones respecto al trabajo realizado, las contribuciones del mismo y el trabajo futuro que se puede llevar a cabo.

Finalmente se incluyen todas las referencias del presente trabajo.

Los anexos son los siguientes:

Anexo I: Ejemplos de valores articulares simulados.
Anexo II: Desarrollo de una interfaz gráfica mediante los programas GPS (GNAT Programming Studio), GtkAda y Glade.
Anexo III: Desarrollo de una aplicación basada en OpenGL escrita en lenguaje Ada.
Anexo IV: Guía de usuario del entorno gráfico de un entrenador virtual de prótesis de mano.
Anexo V: Glosario.

2 ESTUDIO Y ANÁLISIS DE LAS HERRAMIENTAS DE MODELAMIENTO 3D.

En el mundo de los gráficos por computador, existen dos líneas de investigación que se encuentran en estados muy avanzados, pero han seguido caminos divergentes: de un lado la Animación 3D que ha centrado toda su atención en la calidad visual de las imágenes, y de otro, la Realidad Virtual o Informática Gráfica en Tiempo Real, que ha intentando conseguir la mayor calidad posible, pero dando prioridad a la rapidez en la generación de imágenes (10).

Pero tanto la Animación 3D como la Informática Gráfica en Tiempo Real siempre han estado complementadas por herramientas específicas o interfaces de edición. Estas herramientas gráficas han facilitado al desarrollador de entornos modelar objetos, generar escenarios o realizar secuencias de movimientos de manera manual mediante series de comandos o introduciendo las posiciones de la geometría manualmente. Estas herramientas también han ido evolucionando al mismo tiempo que lo han ido haciendo los motores gráficos, permitiendo en algunas de ellas una visualización en tiempo real del resultado final de la imagen. Además, gracias a la modularidad de estas aplicaciones, cada vez son más las aplicaciones que admiten extensiones, desarrolladas por terceras personas, que se pueden ir añadiendo a las mismas para mejorar su capacidad y flexibilidad (10).

Normalmente las tareas de realizar gráficas suelen presentar una complejidad computacional tan elevada que hace que el tiempo empleado en generar cada imagen sea habitualmente de varios minutos o incluso varias horas. Frente a este tipo de aplicaciones en las que prima la calidad de la imagen (a costa del tiempo de cálculo que sea necesario), existen otro tipo de aplicaciones, en las que es necesario que la generación de imágenes se produzca con una rapidez tal que posibilite la interactividad (por encima de 8 imágenes/segundo), o la percepción continua del movimiento (tasas de generación de imágenes en torno a los 50 imágenes/segundo) (10). Con este objetivo, los fabricantes de hardware han diseñado tarjetas gráficas especialmente orientadas a descargar a la CPU de las tareas relacionadas con el renderizado (proceso de generar una imagen en 3D o una animación en 3D a partir de un modelo, usando una aplicación de computadora) de las imágenes tridimensionales (10). Los métodos empleados para generar estas imágenes en tiempo real están fuertemente condicionados por las características del hardware gráfico existente, por consiguiente cabe anotar que el trabajo

desarrollado en este proyecto requiere de un ordenador con características hardware que permitan llevar a cabo todo el procesamiento requerido.

2.1 SOFTWARE DE MODELADO Y SIMULACIÓN 3D COMERCIAL.

A continuación se relaciona el software disponible en el mercado, que permite realizar simulación dinámica y que son compatibles con Matlab y Solid Edge (software disponible en el Departamento de Electrónica, Instrumentación y Control de la Facultad).

2.1.1 VisualNastran 4D.

Software de simulación comercial, que permite realizar el simulado del movimiento en 3D con el análisis de elementos finitos dinámicos en Windows. Se utiliza para cuatro etapas del ciclo del diseño: en la etapa de dibujo de piezas, modela sólidos en 3D en forma nativa y además permite importar geometrías en una variedad de CAD como Inventor de Autodesk, SolidWorks, Solid Edge, y Pro/ENGINEER.En la etapa de movimiento, permite la integración de las piezas y luego la generación del movimiento a través de la programación de grados de libertad en las juntas; el análisis de movimiento mide las fuerzas, esfuerzos de torsión, posiciones, las velocidades, la aceleración y la energía cinética de, los cuerpos. En la etapa de desintegración, se puede realizar el análisis de elementos finitos, siendo necesario componer la pieza mediante pequeños elementos, es decir, analiza el mecanismo de manera independiente para cada una de las piezas e identifica áreas de mayor tensión. En la etapa de control, se realiza la integración con MATLAB/Simulink para realizar el control de los elementos del sistema o mecanismo (11).

2.1.2 Software COSMOSmotion.

COSMOSMotion es un software robusto de simulación y análisis cinemático y dinámico de mecanismos. Éste permite asegurar el funcionamiento correcto de un diseño antes de su construcción, esto significa una reducción importante en el número de prototipos físicos a construir y acelera el ciclo de desarrollo del producto. COSMOSMotion permite entender el funcionamiento de un diseño sin necesidad de crear ningún prototipo físico (12).

2.1.3 3D GameStudio.

3D GameStudio es un kit de desarrollo comercial de juegos de ordenador. Consta de un motor 3D, un motor 2D, un editor de niveles y modelos, un compilador de scripts y librerías de modelos, texturas, objetos, etcétera. Manipula con igual rendimiento escenas de interior y exterior. Tiene un motor de iluminación que soporta sombras verdaderas y fuentes de luz en movimiento. El principal objetivo que esta aplicación persigue es que el desarrollador del juego no necesite ser un programador experimentado. Se reduce al máximo el esfuerzo de desarrollo a costa de perder flexibilidad en el diseño del juego.
Ofrece tres posibilidades para crear un juego:

1) Juegos diseñados a base únicamente de controles, para usuarios con pocos conocimientos de programación.
2) Juegos o efectos diseñados con algo de programación utilizando C-scripts.
3) Juegos o efectos programados en C++ o Delphi, para programadores con experiencia.

Incorpora un núcleo de videojuegos, llamado A5. Pero lo que aquí se entiende por núcleo es un sistema de desarrollo que se encarga de generar efectos 3D y controlar la inteligencia artificial del juego. En la Figura 2.1 se pueden observar ejemplos de entornos virtuales 3D que han sido creados con 3D GameStudio y en la Figura 2.2 se muestra la interfaz del editor de dicha herramienta software (13).

Características que implementa:

- Modelado de terreno.
- Fuentes de luz y sombras estáticas y dinámicas.
- Mapeado de texturas.
- Efectos de partículas, niebla coloreada y transparencia.
- Transformación y deformación suave de mallas.
- Efecto de movimiento lento/rápido.
- Detección de colisiones.
- Plug-In para crear modelos animados con 3D Studio Max.
- Editor de niveles, modelos y terreno.
- Compilador y depurador de C-scripts.

Figura 2.1 Capturas de Imagen de 3D GameStudio.
Fuente: (14).

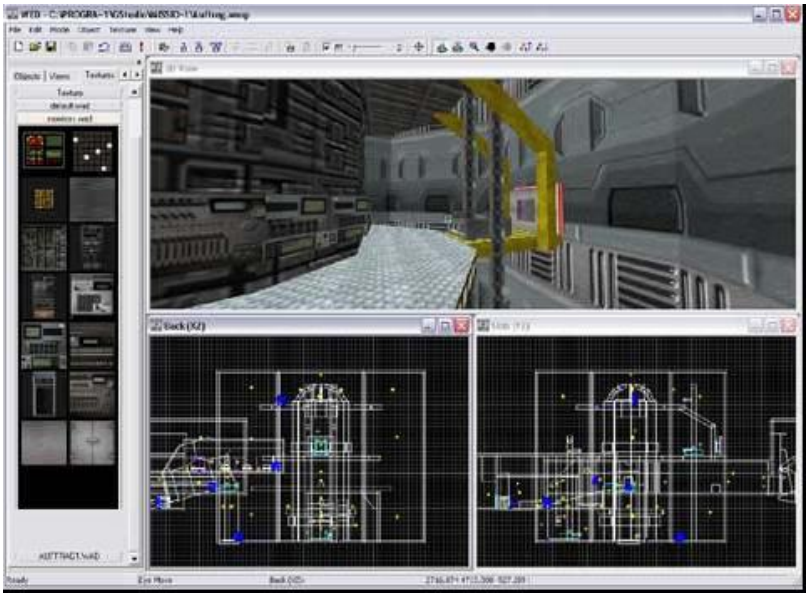


Figura 2.2 Interfaz del editor de 3D GamStudio.
Fuente: (14).

2.1.4 Torque Game Engine.

Torque Game Engine (TGE) es el motor desarrollado por Dinamix para su juego Tribes2. Enfocado a la simulación de misiones militares, incluye utilidades para la creación de terrenos, superficies acuáticas, interiores estilo portal y sistemas de partículas. También incluye soporte multiplataforma (Windows, Mac OS y Linux), soporte para red, creación de interfaces de usuario y lenguaje de script estilo C++. Permite importar objetos desde 3D Studio MAX y dispone de librerías matemáticas, de detección de colisiones, de física de

vehículos y una base de datos especial. La Figura 2.3 brinda unos ejemplos de gráficos con Torque Game Engine (15).

Características que implementa:

- Mallas poligonales progresivas.
- Edición externa mediante 3D Studio MAX o MilkShape.
- Editor de interiores basado en Portal.
- Modelado de terreno y de superficies acuáticas.
- Nivel de detalle continúo.
- Iluminación por vértices.
- Sombras arrojadas con recortado sobre el entorno.
- Niebla esférica por distancia y volumétrica por capas.
- Mapeo de texturas.
- Mapas de entorno.
- Animación de esqueletos.
- Deformación de vértices en mallas.
- Efecto de Partículas.
- Generador de Interfaces de Usuario.
- Multiplataforma.
- Driver SFX/Music con OpenAL.
- Conversor de 3D Studio MAX y milkshape.
- Editor de terrenos, de superficies acuáticas y de interiores.
- Editor de misiones.

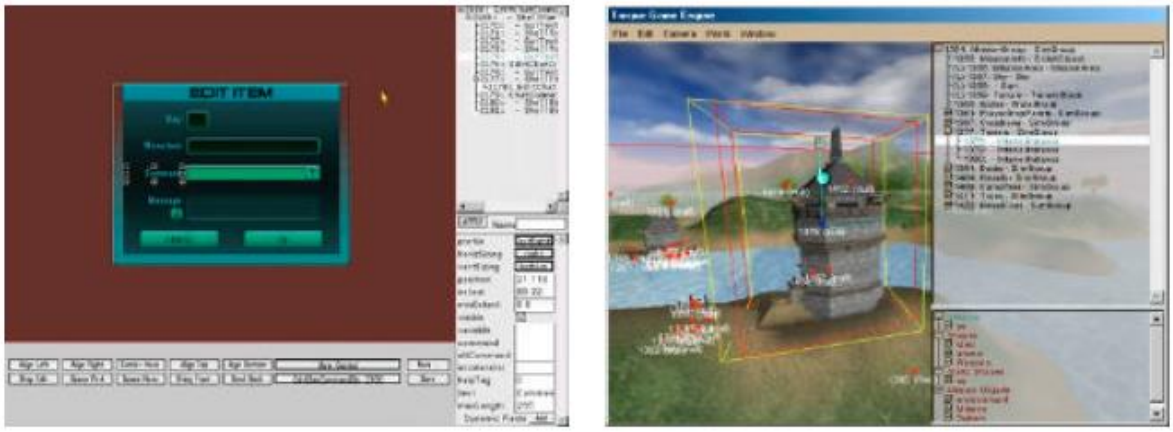


Figura 2.3 Capturas de Imagen del motor Torque.
Fuente: (14).

2.2 SOFTWARE DE MODELADO Y SIMULACIÓN 3D LIBRE.

2.2.1 Globe_3D.

Es una herramienta libre con código abierto escrita en lenguaje de programación Ada y funciona en tiempo real, su estructura está basada en OpenGL (Open Graphics Library), por lo cual se puede desarrollar cualquier tipo de aplicación gráfica tridimensional que sea multiplataforma (Windows, Mac OS y Linux) mediante programación en lenguaje Ada. En la Figura 2.4 se pueden observar varios ejemplos de gráficos tridimensionales creados con Globe_3D (16).

Sus características son:

- Completamente desarrollado en lenguaje de programación Ada.
- Estructura basada en la librería gráfica estándar OpenGL, permite crear cualquier aplicación gráfica 3D.
- Multiplataforma, libre, gratuito y con un tamaño de origen realmente pequeño.
- Renderizado (proceso de generar una imagen en 3D o una animación en 3D a partir de un modelo, usando una aplicación de computadora) en tiempo real, es muy rápido si se utiliza tarjeta gráfica aceleradora 3D.
- Completo movimiento visual.
- Permite aplicar todo tipo de transformaciones sobre los objetos (rotaciones, traslaciones, escalado).
- Puede mostrar combinaciones de colores, materiales y texturas.
- Programación de luces estáticas de colores con sombras e igualmente luces dinámicas de colores con sombras.
- Permite aplicar transparencia a los objetos.
- Partición binaria del espacio.
- Fácil administración de texturas almacenadas en archivos .zip.
- Salida-entrada de objetos 3D o grupos de objetos enlazados.
- Captura de pantalla y de video.
- Soporta múltiples vistas.
- Permite aplicar la técnica de Portales.

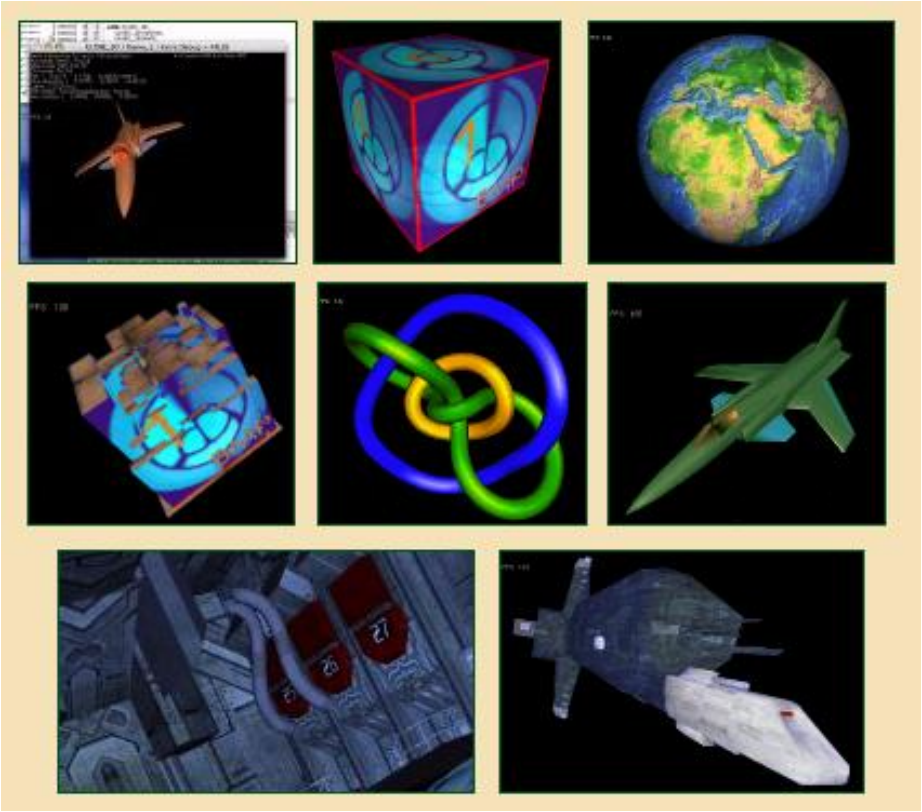


Figura 2.4 Capturas de imagen de gráficos creados con Globe_3D.
Fuente: (16).

2.2.2 Blender.

Blender es un software de animación 3D multiplataforma (Compatible con Mac Os x, Windows, Linux, Solaris, IRIX y freeBSD). Con él se pueden crear escenas y vídeos generados por computadora, con muchas prestaciones y facilidades. En la parte izquierda de la Figura 2.5 se puede observar la interfaz de usuario de Blender en la cual se crea el modelo 3D y en la parte derecha de dicha figura se muestra un ejemplo gráfico creado con Blender (17).

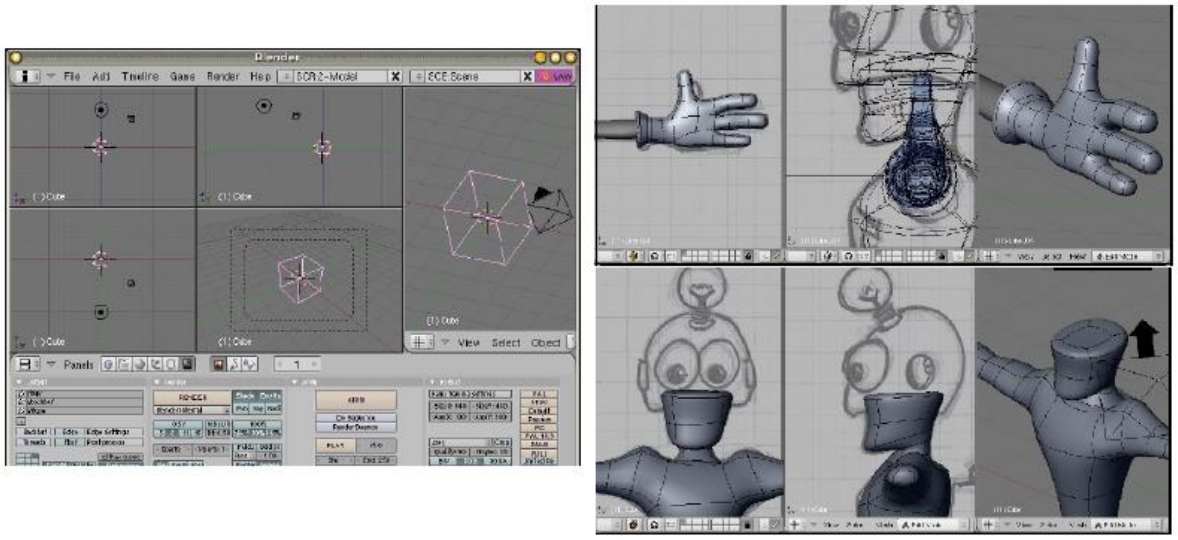


Figura 2.5 Capturas de imagen del software Blender.
Fuente: (17).

Sus principales características son las siguientes:

- Multiplataforma, libre, gratuito y con un tamaño de origen realmente pequeño comparado con otros paquetes de 3D, dependiendo del sistema operativo en el que se ejecuta.
- Capacidad para una gran variedad de primitivas geométricas, incluyendo curvas, mallas poligonales, vacíos, etc.
- Junto a las herramientas de animación se incluyen cinemática inversa, deformaciones por armadura o cuadrícula, vértices de carga y partículas estáticas y dinámicas.
- Edición de audio y sincronización de video.
- Características interactivas para juegos como detección de colisiones, recreaciones dinámicas y lógica.
- Lenguaje de programación Python para automatizar o controlar varias tareas.
- Blender acepta varios formatos gráficos.
- Motor de juegos 3D integrado. Para más control se usa programación en lenguaje Python.
- Sistema de partículas estáticas para simular cabellos y pelajes, al que se han agregado nuevas propiedades entre las opciones de shaders (Conjunto de instrucciones gráficas destinadas para el acelerador grafico, estas instrucciones dan el aspecto final de un objeto. Los shaders determinan materiales, efectos, color, luz, sombra y etc.) para lograr texturas realistas.

2.2.3 Ogre.

OGRE (Object-Oriented Graphics Rendering Engine) es un motor escrito en C++ flexible, orientado a escenas, y diseñado para hacer más simple e intuitiva a los desarrolladores la producción de juegos utilizando hardware de aceleración 3D. La librería de clases permite abstraer los detalles asociados a las librerías de bajo nivel (OpenGL o Direct3D), proporcionando una interfaz basada en objetos. La Figura 2.6 da a conocer varios ejemplos de gráficos tridimensionales creados con el motor Ogre (18).

Características que implementa:

- Graficado jerárquico de escena.
- Permite la edición externa mediante 3D Studio MAX.
- Soporta la técnica de portales.
- Permite luces puntuales, direccionales y de foco.
- Admite texturas animadas y realiza cálculo de coordenadas de textura.
- Brinda efecto de generación de Partículas.
- Se pueden realizar objetos transparentes.
- Su Interfaz es orientada a objetos.
- Admite scripts mediante Python.

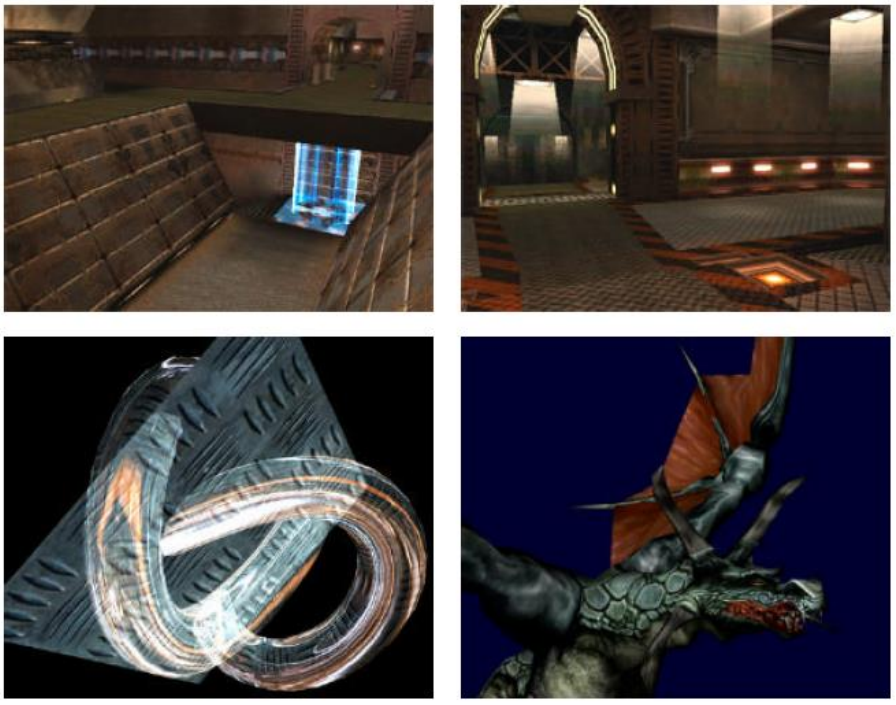


Figura 2.6 Capturas de imagen de gráficos creados con el motor Ogre.
Fuente: (14).

2.2.4 Crystal Space.

Cristal Space es un kit de desarrollo de juegos 3D libre (LGPL) y portable escrito en C++. Soporta seis grados de libertad, luces de colores, mipmapping (técnica de colecciones de imágenes de mapas de bits que acompañan a una textura principal para aumentar la velocidad de renderizado y reducir sus artefactos.), portales, espejos, transparencias, superficies reflectivas, sprites 3D (basados en frames o animaciones de esqueleto), texturas de procedimiento, radiosidad (conjunto de técnicas para el cálculo de la iluminación global de la forma más realista posible en el campo de los gráficos 3D), sistemas de partículas, halos, niebla volumétrica, lenguaje de script (Python y otros), soporte para visualización a 8-bits, 16-bits y 32-bits, Direct3D, OpenGL, Glide, y render (término que se refiere a la generación de una imagen individual. El render consume potencia de computo debido a los cálculos necesarios según el número de objetos, luces y efectos en la escena) por software, soporte para fuentes, transformaciones jerárquicas, etc. Actualmente Crystal Space puede ejecutarse sobre GNU/ Linux, Windows, BeOS, NextStep, OpenStep, MacOS/X y DOS. En la Figura 2.7 se pueden apreciar algunos ejemplos de gráficos creados con Cristal Space (19).

Características que implementa:

- Mallas 3D con animación. Importación de objetos 3DS, MDL, MD2, OBJ, POV, y ASE. Las mallas son multi-resolución “progressive meshes”.
- Sistema de visibilidad basado en la combinación de portales.
- Cielo iluminado dinámicamente, sol en movimiento.
- Superficies brillantes y reflejos con espejos.
- Luces estáticas de colores con sombras reales (sombras pre-calculadas).
- Luces dinámicas, de colores con sobras suaves.
- Texturas de cualquier dimensión y formatos GIF, TGA, PNG, BMP, JPG y otros.
- Pueden aplicarse con transformaciones (escalado, rotación).
- Soporte para texturas dinámicas.
- Plugins para otros archivos fuente.
- Brinda sistema de detección de colisiones jerárquico.
- Soporte para sonido 3D (DS3D, EAX, A3D.) en varios formatos: WAV, MP3, Ogg/Vorbis, AU, AIFF, IFF, and MOD.
- Se incluyen varios scripts de Blender para exportar modelos y mapas al crystal space.

Figura 2.7 Capturas de imagen de gráficos creados con el motor Cristal Space. Fuente: (14).

2.2.5 Genesis3D.

Genesis3D es un motor para la visualización de escenas tridimensionales en tiempo real y que permite construir aplicaciones gráficas 3D de altas prestaciones. Ha sido diseñado principalmente para la visualización de escenas de interior logrando un alto 'frame-rate' siempre y cuando estén compuestas por una cantidad moderada de polígonos. También puede ser utilizado para escenas de exterior si el diseño de las escenas se realiza tomando ciertas precauciones. Sus principales características son la detección rápida de colisiones, iluminación pre-calculada y chequeo de la visibilidad. Su principal inconveniente es la visualización de escenarios exteriores sin imponer algún tipo de restricción o límite al tamaño de la escena. La Figura 2.8 muestra dos ejemplos de gráficos creados con Genesis3D (20).

Características que implementa:

- Primitivas básicas: arco, cono, cilindro, cubo, esferoide y escalera.
- Permite definirlas sólidas y agujereadas.
- Diseñado para integrar fácilmente cualquier tipo de luz.
- Implementa radiosidad, iluminado de puntos e iluminado dinámico.
- Mapeado de texturas.
- Efectos de partículas, niebla, coronas.
- Simulación física del movimiento.

- Editor de escenas.
- Sólo soporta Microsoft Visual C++ v6 y Requiere DirectX6 o superior.
- Requiere Windows 95, 98, ME, 2000 y XP, No hay soporte para otras plataforma, poca documentación.
- No brinda soporte para OpenGL.
- Simulación física de movimiento cuando un objeto es empujado o disparado.
- Las mejoras se obtienen mediante las contribuciones que realizan los propios usuarios del motor.

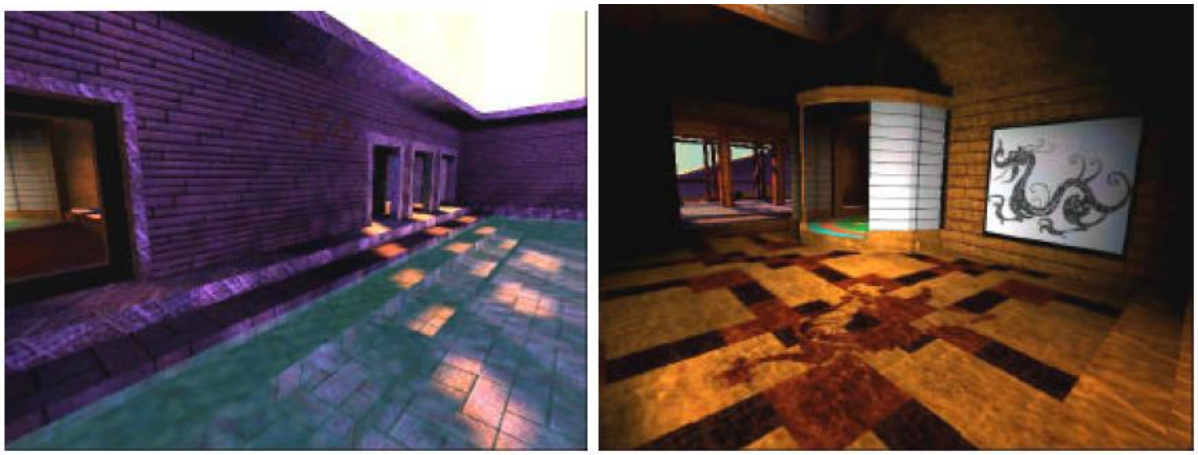


Figura 2.8 Capturas de imagen de gráficos creados con el motor Genesis3D.
Fuente: (14).

2.3 SELECCIÓN DEL SOFTWARE DE MODELADO Y SIMULACIÓN 3D PARA EL DESARROLLO DEL ENTORNO.

Después de estudiar y analizar las herramientas software que permiten realizar el modelado y la simulación del entorno en el cual se debe visualizar el modelo de la prótesis de mano 3D y tomando en consideración el cumplimiento de los requerimientos de usar herramientas de software libre, tanto en el entorno de programación como en las demás herramientas, y además que la aplicación final proporcione características de tiempo real no estricto; se consideró necesario para el presente trabajo emplear una herramienta software que estuviera basada en una de las librerías estándar más conocidas para la creación de gráficos como lo es OpenGL (21) y que además estuviese escrita en uno de los lenguajes más potentes y fiables en el desarrollo de aplicaciones de Tiempo Real como lo es el lenguaje Ada (22). Se tomó la decisión de emplear la herramienta de software Globe_3D, puesto que es el motor gráfico que además de brindar al desarrollador un buen número de características para el desarrollo de gráficos

tridimensionales, es software libre, cumple con los requerimientos de estar escrito en lenguaje de programación Ada, está basado en OpenGL y su funcionamiento es en tiempo real.

Básicamente Globe_3D es la herramienta empleada para el desarrollo de la parte correspondiente al entorno gráfico del entrenador virtual de prótesis de mano, y puesto que está basado en OpenGL, su arquitectura se compone de aquellos elementos de dicha API(Interfaz de Programación de Aplicaciones) la cual es una librería de funciones que permite visualizar gráficos 3D en las aplicaciones, la gran ventaja que brinda esta librería es que la aplicación desarrollada es independiente del sistema operativo y hardware disponible (23); por lo tanto, si a la computadora se le instala una tarjeta aceleradora de gráficos o bien se la cambia por otra, no hará falta realizar cambios en el programa para aprovechar la potencia de dicha tarjeta (10). Con base a lo anterior se puede aclarar que la arquitectura de Globe_3D en cuanto al soporte de gráficos se divide en cuatro partes funcionales: GL, librería que proporciona todo lo necesario para acceder a las funciones de dibujado de OpenGL; GLU (OpenGL Utility Library), librería que proporciona acceso rápido las funciones más comunes de OpenGL; GLUT (OpenGL Utility Toolkit) permite crear aplicaciones de ventanas totalmente portables y GLX (para Sistemas X Windows común en plataformas Unix), WGL (para sistemas Microsoft Windows), AGL (para sistemas Apple Macintosh) que son librerías que proporcionan acceso a OpenGL para poder interactuar con el sistema de ventanas del sistema operativo empleado (24), (21), (23).

En el capítulo 4 se da una descripción más detallada respecto a la arquitectura, forma de enlazar y emplear Globe_3D en el entorno de programación en el cual se desarrolló el entorno gráfico tridimensional que representa el modelo dinámico de la prótesis virtual de mano robótica.

3 LA MANO HUMANA Y SU CAPACIDAD DE APREHENSIÓN DE OBJETOS.

La mano humana es una estructura altamente compleja que en muchas maneras desafía el entendimiento. Consiste de la palma y cinco dedos, y está hecha de una colección de huesos, músculos, ligamentos, tendones y estructuras vasculares, todos ellos encapsulados por la piel. Miles de sensores en la piel, los músculos y las articulaciones le permiten al cerebro conocer su estado actual (25).

En un principio las manos del ser humano primitivo fueron adquiriendo la capacidad de agarrar piedras, huesos y madera para modificarlos y así crear herramientas funcionalmente efectivas; las herramientas comienzan a ser una extensión muy importante de la mano, pues sin ellas la mano estaría limitada en fuerza y precisión. Por ello resulta realmente interesante resaltar que a medida que el ser humano ha ido evolucionando, ha podido, gracias a la capacidad pensante de su cerebro visionar elementos, herramientas e infinita diversidad de cosas que con sus propias manos ha podido desarrollar y emplear para diversos fines. Por tanto, la mano puede ser vista como una herramienta versátil de propósito general, la cual se puede ajustar a una gran variedad de posturas con el fin de desarrollar una amplia diversidad de funciones o tareas. Por ejemplo adecuarse a una postura para agarrar un objeto o una herramienta y así desarrollar una función o bien tomar ella misma la forma adecuada para desarrollar directamente aquella función (25).

3.1 ANATOMÍA DE LA MANO HUMANA.

La mano humana anatómicamente consiste en una palma central (metacarpo) de la que surgen cinco dedos, está unida al antebrazo por una unión llamada muñeca (carpio). Además, la mano está compuesta de varios músculos y ligamentos diferentes que permiten una gran cantidad de movimientos y destreza, es decir, son el principal órgano para la manipulación física del medio. La punta de los dedos contiene algunas de las zonas con más terminaciones nerviosas del cuerpo humano, son la principal fuente de información táctil sobre el entorno, por eso el sentido del tacto se asocia inmediatamente con las manos (26).

Existen tres tipos principales de huesos en la mano, tal como se puede apreciar en la Figura 3.1, incluyendo los siguientes:

- ***Falanges:*** son catorce huesos que se encuentran en los dedos de cada mano. Cada dedo tiene tres falanges (distal, medial y proximal); el pulgar tiene sólo dos.
- ***Huesos metacarpianos:*** los cinco huesos que componen la parte media de la mano.
- ***Huesos carpianos:*** los ocho huesos que forman la muñeca. Los huesos carpianos están conectados a dos huesos del brazo: el cúbito y el radio.

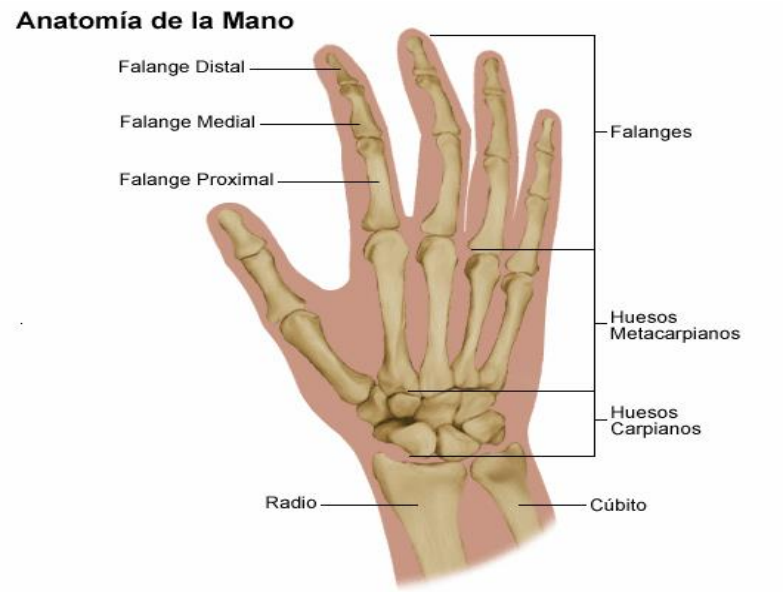


Figura 3.1 Anatomía de la Mano Humana.
Fuente: (26).

3.1.1 La mano como un mecanismo.

Si bien la mano humana es más que una máquina, existen razones por las cuales se debe pensar y considerar la mano como algo mecánico, ello con el fin de entender que los principios de la mecánica pueden ser aplicados para estudiar la mano, y ser capaces de producir manos mecánicas modificadas con sus respectivas articulaciones y tendones (27). Paul Brand consideró tres aspectos importantes de la mecánica de la mano humana: el motor (los músculos), la transmisión (tendones, huesos y articulaciones), y la aplicación (a través de los tejidos y la piel) (25). En la siguiente Figura 3.2 se muestra la disposición física de ellos.

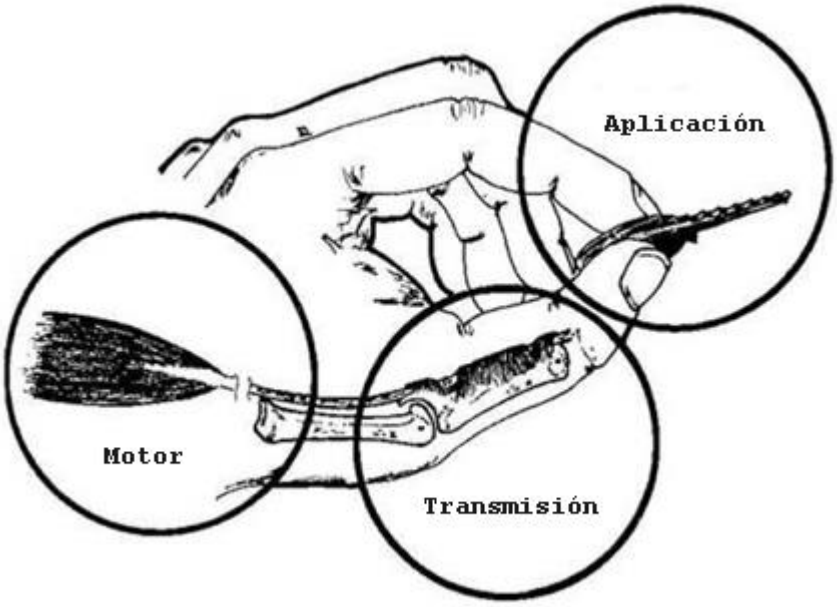


Figura 3.2 Los tres elementos principales del movimiento de la mano humana.
Fuente: (25).

3.1.2 Las articulaciones.

Las articulaciones son estructuras que tienen como propósito mantener conectados los huesos, por medio de los ligamentos y los músculos. La dirección y el grado de movimiento dependen de la forma de las superficies de la articulación y de la contracción y dilatación de los músculos (25).

Los movimientos de la mano son de tipos diferentes: cada dedo puede moverse en el plano de la mano (aducción) para ir más cerca al medio eje, puede mover lejos del eje (abducción), se puede encorvar y puede extenderse (flexión). El dedo pulgar también puede entrar en oposición con otros dedos. La Figura 3.3 muestra algunos movimientos comunes en las articulaciones de los dedos de la mano (25).

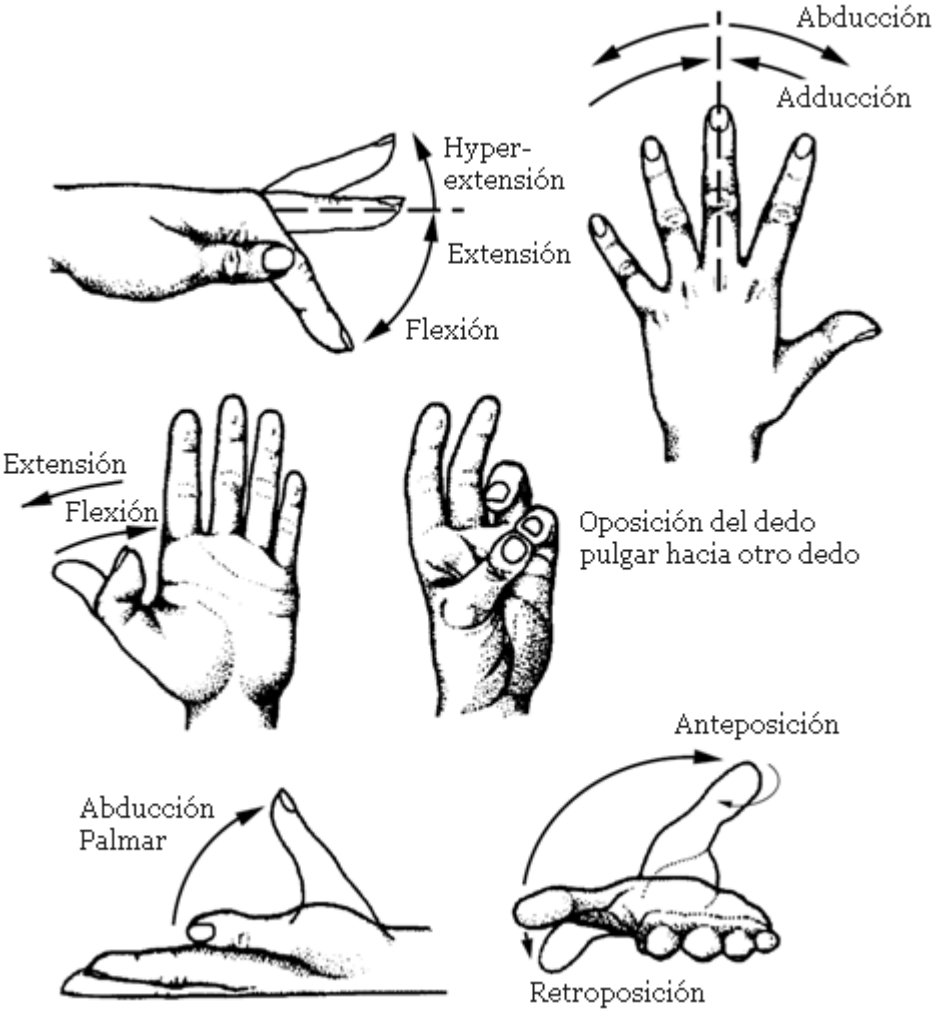


Figura 3.3 Movimientos comunes en las articulaciones de los dedos de la mano humana. Fuente: (25).

Las ligaduras en la palma conectan los huesos metacarpianos y carpianos. Ellos pueden bloquear algunos movimientos de los huesos metacarpianos para dar una estructura más rígida a las falanges de la palma (25).

Los tendones conectan músculos y huesos. Ellos son internamente hechos de colágeno, dándole propiedades de elasticidad para que puedan volver a poner los dedos en la posición original después de una flexión (25).

3.2 CLASIFICACIÓN DE AGARRES DE LA MANO.

3.2.1 Clasificación de Schlesinger.

El agarre humano posee una sorprendente flexibilidad e incomparable destreza. Tales características involucran un proceso muy complejo, por esta razón se ha deseado duplicar esta capacidad humana por ingenieros, terapeutas y científicos médicos quienes desean ayudar a los pacientes en el desarrollo funcional de la mano. G. Schlesinger (27) desarrolló una de las muchas clasificaciones que han sido creadas para las diferentes posturas de la mano. Schlesinger se enfocó en determinar cuál era la funcionalidad específica necesaria para el agarre de varios tipos de objetos, con lo cual concibió un conjunto mínimo de seis posturas de agarre (27) como se muestra en la Figura 3.4.

Figura 3.4 Seis diferentes tipos de agarre definidos por Schlesinger.
Fuente: (27)

- Agarre cilíndrico (*cylindrical grasp*): Se usa para objetos cilíndricos, tales como cilindros huecos y tasas.
- Agarre de punta (*tip grasp*): Se usa para agarrar objetos muy pequeños tales como agujas o pedazos de papel.
- Agarre de gancho (*hook grasp*): Se usa para cargar objetos pesados tales como maletas.
- Agarre palmar (*palmar grasp*): Se usa para objetos delgados y relativamente gruesos tales como un cigarrillo.
- Agarre esférico (*spherical grasp*): Se usa para objetos esféricos, tales como pelotas y manzanas.
- Agarre lateral (*lateral grasp*): Se usa para objetos planos y delgados tales como llaves y cuchillas.

3.2.2 Clasificación de Napier.

La clasificación de Napier es la primera en presentar la dicotomía entre agarres de potencia y de precisión.

- *Un agarre de potencia:* es una postura en la que un objeto se sostiene abrochado por los dedos parcialmente flexionados y la palma. Se distinguen dos variantes: con el pulgar aducido (de potencia preciso) y con el pulgar abducido (de potencia con baja precisión). Imágenes de la parte superior de la Figura 3.5.
- *Un agarre de precisión:* es un agarre en el que el objeto se pincha entre los flexores del pulgar y los dedos. Imágenes de la parte inferior de la Figura 3.5.

De acuerdo a Napier, los factores que influencian la postura de la mano son: la forma y tamaño del objeto y la naturaleza de la tarea. Aunque los dos primeros factores influencian el tipo de aprehensión (captura) que se usa, usualmente es el tipo de tarea (o naturaleza de la actividad a realizar) la que normalmente determina la postura del agarre (27).

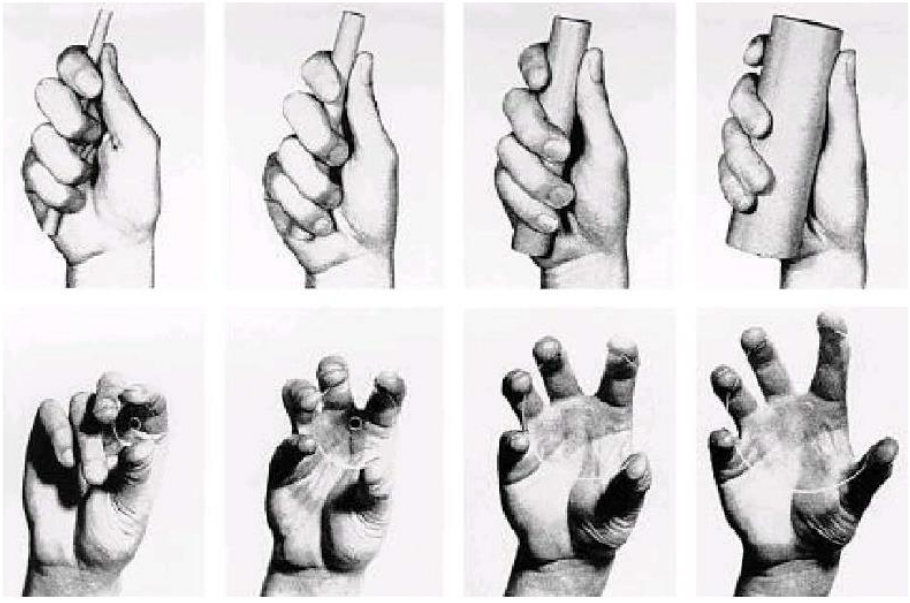


Figura 3.5 Clasificación de agarres según Napier.
Fuente: (27).

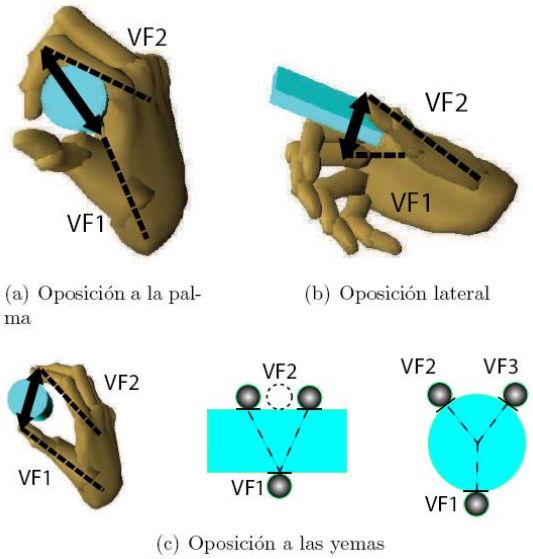
3.2.3 Clasificación de Arbib.

Un método alternativo de analizar una clasificación de agarre es la considerada por Arbib, quien considera que la aprehensión involucra al menos dos fuerzas aplicadas en oposición contra las superficies de un objeto. Según este concepto se distinguen tres tipos básicos de agarre (25):

- *Oposición a las yemas:* la fuerza se aplica a lo largo de un eje paralelo a la palma. Ofrece flexibilidad para manipulación a expensas de estabilidad y fuerza.
- *Oposición de palma:* la fuerza se aplica a lo largo de una dirección perpendicular a la palma. Sacrifica flexibilidad en favor de estabilidad.
- *Oposición lateral:* las fuerzas se aplican a lo largo de una dirección transversa a la palma. Es un compromiso entre flexibilidad y estabilidad.

Para cuantificar y describir la clasificación de Arbib, se introduce el concepto de dedo virtual. Es una representación abstracta usada para describir una colección de dedos o superficies de la mano aplicando fuerzas de oposición. Los dedos reales se agrupan en un dedo virtual (VF) (25).

Para oposición a la palma, la palma es el primer dedo virtual (VF1), y los cuatro dedos son el segundo dedo virtual (VF2). VF1 termina en cualquier lugar sobre la superficie palmar, mientras que VF2 termina sobre la unión MCP de los dedos. Para oposición a las yemas, la oposición ocurre entre el pulgar (VF1) y uno o más dedos (VF2). VF1 y VF2 varían en longitud y orientación como el pulgar y los dedos se extienden o flexionan respectivamente. Para oposición lateral, la oposición ocurre entre el pulgar (VF1) y el lado del dedo índice (VF2). Aquí el eje de oposición es ortogonal a VF1 y VF2. Los tres tipos de agarres de oposición descritos anteriormente se pueden observar claramente en la Figura 3.6. La oposición a las yemas se usa en agarre de precisión (Napier). Oposición a la palma se usa en el agarre de potencia tipo martillo " (Napier). Una combinación de oposición a la palma (poder y estabilidad) con oposición lateral (dirección) forman el agarre de precisión (Napier) (25).



(a) Oposición a la palma

(b) Oposición lateral

(c) Oposición a las yemas

Figura 3.6 Clasificación de agarres según Arbib.
Fuente: (25).

3.3 LAS PRÓTESIS COMO SUSTITUTO DE LA EXTREMIDAD PERDIDA.

El área de la robótica aplicada al ser humano es comúnmente denominada Biónica, que en una definición más exacta corresponde a los análisis del funcionamiento real de los sistemas vivos y una vez descubiertos sus secretos, materializarlos en los aparatos. Esta definición indica que el primer ingeniero biónico fue Leonardo Da Vinci, quien estudió los principios de funcionamiento de los seres vivos para aplicarlos en el diseño de máquinas (2). Parte de la biónica se encarga del diseño de prótesis, dado que un componente fundamental en el proceso de rehabilitación de un paciente con amputación de mano consiste en reemplazar el miembro faltante por un mecanismo artificial conocido como prótesis. Éstas se clasifican en prótesis pasivas y prótesis activas; las primeras tienen un objetivo puramente estético y a diferencia de las segundas no pueden ser controladas por el paciente. Las prótesis activas por el contrario permiten la movilidad de la mano artificial por parte de la persona discapacitada (28).

3.3.1 Estado del arte en prótesis para amputados de mano.

Ya en el siglo XX, el objetivo de que los amputados regresaran a su vida laboral, es alcanzado gracias a los esfuerzos del médico francés *Gripoulleau*, quien realizó diferentes accesorios que podían ser usados como unidad terminal, tales como anillos, ganchos y diversos instrumentos metálicos, que brindaban la capacidad de realizar trabajo de fuerza o de precisión (2).

En el año de 1912 Dorrance en Estados Unidos desarrolló el *Hook*, que es una unidad terminal que permite abrir activamente, mediante movimientos de la cintura escapular, además se cierra pasivamente por la acción de un tirante de goma. Casi al mismo tiempo fue desarrollado en Alemania el gancho *Fischer* cuya ventaja principal era que poseía una mayor potencia y diversidad en los tipos de prensión y sujeción de los objetos. El origen de las prótesis activadas por los músculos del muñón se da en Alemania gracias a *Sauerbruch*, el cual logra idear como conectar la musculatura flexora del antebrazo con el mecanismo de la mano artificial, mediante varillas de marfil que hacía pasar a través de túneles cutáneos, haciendo posible que la prótesis se moviera de forma activa debido a la contracción muscular (2).

Es hasta 1946 cuando se crean sistemas de propulsión asistida, dando origen a las prótesis neumáticas y eléctricas. Un sistema de propulsión asistida es aquel en el que el movimiento es activado por algún agente externo al cuerpo (2).

Las prótesis con mando mioeléctrico comienzan a surgir en el año de 1960 en Rusia. Esta opción protésica funciona con pequeños potenciales extraídos durante la contracción de las masas musculares del muñón, siendo estos conducidos y amplificados para obtener el movimiento de la misma. En sus inicios, este tipo de prótesis solo era colocada para amputados de antebrazo, logrando una fuerza prensora de dos kilos (2).

Actualmente existen numerosos grupos dedicados al desarrollo de prótesis activas, cuyo trabajo puede clasificarse en dos grandes tendencias. La primera tendencia, denominada cibernética, es más reciente y aún no se han desarrollado prótesis comerciales, y consiste en manipular la prótesis a partir de señales neuronales eferentes provenientes del sistema nervioso central (29). El primer prototipo cibernético fue desarrollado por la *Scuola Superiore Sant'Anna* (Pisa - Italia) como resultado del proyecto *Cyberhand* (30) y su validación clínica se iniciará en el presente año. El segundo enfoque está basado en las señales electromiográficas captadas en los músculos de los miembros superiores del paciente y que son utilizadas para enviar órdenes hacia la prótesis. Este enfoque provee una buena solución para las prótesis activas aunque quedan aún muchos problemas sin adecuada solución.

A pesar de lo innovador del enfoque cibernético, en el desarrollo del proyecto marco del grupo en Automática Industrial se optó por la opción electromiográfica debido a que en ésta, la utilización de la prótesis no requiere ningún tipo de intervención quirúrgica. Una de las partes del desarrollo del proyecto marco es la realización de un entrenador virtual de prótesis de mano.

El presente proyecto, junto con varios proyectos en curso del Grupo en Automática Industrial tales como: modelado y control de una mano robótica, agarre estable de objetos con una prótesis de mano robótica, extracción de características e identificación de movimiento a partir de señales electromiográficas, entre otros, permitirán la construcción de un entrenador de prótesis activa controlada a partir de señales electromiográficas captadas mediante electrodos ubicados en el antebrazo de un ser humano. Estas señales se filtrarán y amplificarán para ser enviadas al sistema que identificará la intención del movimiento. Una vez realizada la identificación, las salidas de este sistema se enviarán al computador que realizará el despliegue gráfico de la prótesis virtual. Dicho ambiente gráfico contendrá el modelo de la mano virtual así como la información del paciente.

La idea es que el software del entorno gráfico se construya con una herramienta libre con el fin de que el paciente pueda desde cualquier computador y solo con la instalación de un hardware de acondicionamiento de señales de bajo costo, entrenarse y mantener sus conexiones nerviosas en buen estado, preparándolas para la implantación de la prótesis real.

3.3.2 Agarre de objetos mediante una prótesis robótica.

El problema del agarre de objetos mediante una mano robótica es algo complejo, puesto que el sistema debe tener en cuenta la cinemática y dinámica de la mano, los contactos entre la mano y el objeto, los grados de libertad redundantes (dado el objeto, puede haber más de una configuración de postura de la mano para agarrar el objeto) y, por supuesto, las características del objeto mismo. Debido a que el problema puede llegar a ser muy complejo (31), usualmente se hacen las siguientes suposiciones sobre el problema:

- La mano y el objeto se modelan como cuerpos rígidos, y los contactos entre ellos determinan los puntos de contacto. Esto es, se asume que la mano sólo agarra objetos que no son deformables.
- Los puntos de contacto entre los objetos y los dedos son ideales (no hay parámetros de fricción o viscosidad en los cálculos).
- No hay grados de libertad redundantes (existe solo una configuración de la mano para agarrar el objeto).

Se sabe también que a partir de las señales electromiográficas captadas del antebrazo de un paciente amputado de mano es posible detectar algunos tipos de intención de agarre. El tipo de agarre deseado, como es de suponer, determina en gran medida las trayectorias de los dedos. Fundamentados en este análisis se impuso, entonces, resolver un problema concomitante importante: la clasificación de las señales electromiográficas obtenidas del paciente para determinar el tipo de agarre deseado. Gracias al apoyo del proyecto Análisis de Señales EMG Superficiales y su Aplicación en Control de Prótesis de Mano (32), en el momento este problema ha sido resuelto para el caso de cuatro tipos de agarre básicos, mano en reposo, no agarre, agarre grueso, y mano abierta. Debido al excelente desempeño de los clasificadores que se obtuvieron se espera que los demás resultados sean similares para clasificar otros tipos de agarre. Por otro lado, el objetivo de agarre estable de objetos ya ha sido estudiado como parte del trabajo de grado de maestría “Aprehensión estable con una mano robótica usando información visual". Los resultados obtenidos, en la actualidad, permiten detectar los perfiles de los objetos y determinar los mejores puntos para un agarre estable (33) (33).

La tarea de agarrar un objeto con la mano, en el ser humano es un proceso cuya información se elabora en el sistema nervioso central a partir de las experiencias adquiridas desde la niñez (33) (33). Con base a lo anterior y teniendo en cuenta que agarrar un objeto de forma estable es un proceso complejo que esta precedido por una fase previa a la manipulación del objeto en la cual el humano adopta una postura de los dedos y la palma de la mano en concordancia con el tipo de tarea a realizar y que el grupo “Agarre estable de objetos con una prótesis de mano robótica” reorientó la investigación hacia la obtención de trayectorias para una prótesis de mano robótica en

una fase de pre-agarre, investigación en la cual el grupo ha desarrollado el algoritmo de generación de trayectorias para el agarre cilíndrico, e igualmente continúa trabajando en la obtención de las trayectorias para los agarres tipo lateral y gancho; se estableció que el software desarrollado en el presente proyecto reproducirá en forma automática las trayectorias articulares dadas en un archivo de texto. El modelo 3D de mano desarrollado toma como consideración que la mano se encuentra posicionada cerca al objeto, lista para desarrollar el agarre del objeto que se encuentra presente en el entorno tridimensional, tal mano 3D posee una arquitectura y modelo matemático con el fin de poder ser representado adecuadamente a nivel computacional.

3.3.3 Arquitectura de la mano y sus dimensiones.

La prótesis de mano que se considerará en este proyecto, ha sido definida por el proyecto "Modelado y control de una mano robótica" donde se obtuvo el modelo dinámico y cinemático de una mano de tres dedos (medio, índice y pulgar). Sin embargo, el prototipo final que se modelará llevará también los dedos anular y meñique, que seguirán fielmente los movimientos del dedo medio, por lo tanto el número total de grados de libertad es de nueve, tres por cada dedo, utilizando articulaciones rotoides en cada caso. Esto hace que el diseño matemático sea más simple, proporcionando sin embargo las funcionalidades básicas de una mano humana. La Figura 3.7 muestra la arquitectura general de la prótesis propuesta (34).

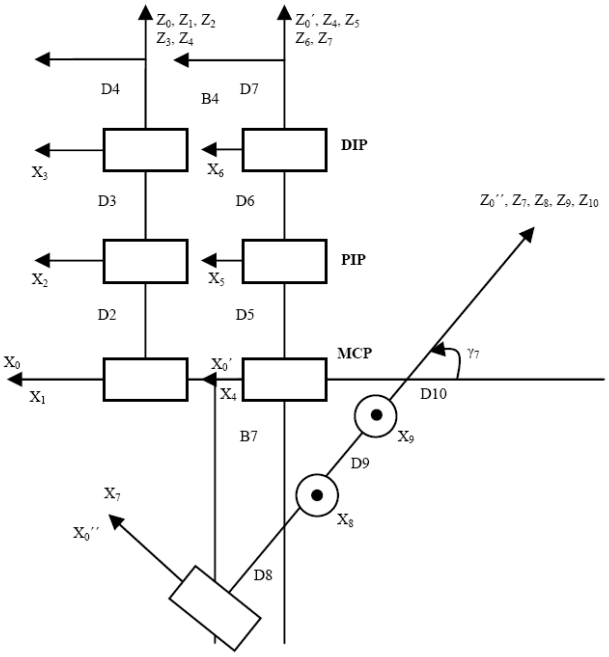


Figura 3.7 Arquitectura de la prótesis de mano.
Fuente: (34).

Los nombres de las falanges son: proximal (MCP), media (PIP) y distal (DIP). Los ejes Xi representan los ejes de movimiento de las articulaciones rotoides. La tabla de parámetros geométricos se muestra a continuación (Tabla 3.1), teniendo en cuenta que se trata de una estructura tipo arborescente (35).

j	σ_j	γ_j	b_j	α_j	d_j	θ_j	r_j
1	0	0	0	0	0	θ1	0
2	0	0	0	0	D2	θ2	0
3	0	0	0	0	D3	θ3	0
4	0	0	B4	0	D4	θ4	0
5	0	0	0	0	D5	θ5	0
6	0	0	0	0	D6	θ6	0
7	0	γ7	B7	0	D7	θ7	0
8	0	0	0	90º	D8	θ8	0
9	0	0	0	0	D9	θ9	0

Tabla 3.1 Parámetros Geométricos.
Fuente: (34).

Los parámetros θj hacen referencia a las variables de cada articulación rotoide; dj hace referencia a la longitud de cada falange; αj al ángulo entre ejes Xj; bj la distancia entre los ejes de cada dedo; rj representa la distancia entre ejes Zj; σj determina que el tipo de articulaciones utilizadas son todas rotoides; y γj representa el ángulo del dedo pulgar respecto a los otros dedos (definido igual a 45º) (34).

En cuanto a las dimensiones de la mano los datos fueron obtenidos del proyecto “Diseño de una prótesis de mano en un ambiente de diseño asistido por computador” (36) en el que se tuvo como referencia a 10 personas, a las cuales se midió las longitudes básicas para realizar el diseño de la mano, los datos se pueden observar en la Tabla 3.2, a dicha tabla se adicionó la última columna que indica las dimensiones conforme a dimensiones del modelo 3D en OpenGL de la mano virtual. Esta conversión se hace necesaria pues OpenGL trabaja con diferentes escalas, ya sea para los colores, las sombras, las dimensiones de radio, longitudes, en algunos casos píxeles, es así que es mejor escalar los valores proporcionalmente de tal manera que se obtengan valores con la misma relación que los obtenidos en la muestra real, muchos de los valores que maneja OpenGL podrían considerarse como escalares que deben adaptarse a las medidas que se manejen como longitudes, intensidad de luz, etc. Para adaptar las medidas del promedio muestral en mm se hizo una regla de tres simple, se tomó una medida como base donde se obtuvieran buenos resultados en cuestión de visibilidad, manejabilidad de la vista y perspectiva, esta medida fue 4.73 la cual corresponde a 180 mm y de ahí en adelante se parte de esta relación para las demás longitudes de la mano, la ecuación que permite dicha conversión será:

$$X = \frac{(m \times 4.73)}{180}$$

Donde: X es la muestra convertida a dimensiones OpenGL.

m: es la muestra (en la tabla 3.2 Longitud escogida) que se desea convertir.

Dimensión	Promedio Muestral (mm)	Porcentaje	Longitud escogida (mm)	Dimensiones conforme a dimensiones OpenGL (ptos)
Longitud de la mano	175	100%	180	4,73
Longitud de la palma	103	59%	106	2,78
Anchura metacarpiana	80	46%	82	2,15
Falanje Inf Dedo Índice	51	29%	52	1,36
Falanje med Dedo Índice	35	20%	36	0,94
Falanje Sup Dedo Índice	24	14%	25	0,65
Falanje Inf Dedo Medio	55	31%	57	1,5
Falanje med Dedo Medio	38	22%	39	1,02
Falanje Sup Dedo Medio	26	15%	27	0,61
Falanje Inf Dedo Anular	52	30%	53	1,39
Falanje med Dedo Anular	35	20%	36	0,94
Falanje Sup Dedo Anular	24	14%	25	0,65
Falanje Inf Dedo Meñique	45	26%	46	1,21
Falanje med Dedo Meñique	30	17%	31	0,81
Falanje Sup Dedo Meñique	20	11%	21	0,55
Pulgar Falanje sup	32	18%	32	0,84
Pulgar Falanje Prox	38	22%	39	1,02
Pulgar Metacarpio	43	25%	44	1,15

Tabla 3.2 Dimensiones de las partes que conforman la Mano.

3.4 TÉCNICA DE AGARRE DE OBJETOS EMPLEADA EN EL SOFTWARE DESARROLLADO.

Para que el agarre de objetos sea desarrollado adecuadamente en el modelo de gráfico computacional se debe implementar en la aplicación una o varias técnicas mediante algoritmos de detección de colisiones; una colisión se produce como resultado del movimiento de los objetos que ocupan parte de una misma porción del espacio al mismo tiempo; cuando esto ocurre con dos objetos estáticos se dice que existe interferencia

entre los dos objetos. Hay muchos algoritmos para la detección de colisiones y la elección de uno u otro dependerá de las características de la aplicación, pues unos son más eficientes en cuanto al tiempo de respuesta y otros son más eficientes en cuanto a la exactitud de la detección de la colisión a costa del tiempo de respuesta (37).

Para este caso el tiempo de respuesta es muy importante por lo cual se opto por un algoritmo sencillo que a su vez garantiza un buen tiempo de respuesta. El algoritmo usado para el agarre de objetos con la mano virtual fue llevado a cabo con el principio de esferas envolventes pero a otro nivel. La elección de este método es debido a que la aplicación 3D tiene fuertes especificaciones en cuanto a tiempo real se refiere y básicamente este algoritmo es muy rápido debido a que solamente se limita a calcular distancias. En este caso no se elige una esfera envolvente para cada objeto, sino que se sitúan tantas esferas sean necesarios en cada objeto de acuerdo al radio de los cilindros o diferentes espacios en la mano. De esta manera no se alterará su geometría cuando la mano adquiere su forma sólida.

Con esta técnica se evita calcular la colisión píxel a píxel una vez detectada la colisión con la técnica de esferas envolventes tradicional. Simplemente se limita a calcular distancias entre esferas, cabe anotar que esta técnica fue factible por la anatomía de la mano y el hecho de que pueda rellenarse con esferas teniendo una buena exactitud. En el ítem 4.5.4 del capítulo 4 se describe en forma más detallada éste método.

4 ARQUITECTURA, DISEÑO E IMPLEMENTACIÓN DEL SISTEMA.

En el presente capítulo se describe en forma detallada la arquitectura de diseño e implementación de la cual se compone y organiza la aplicación 3D que representa en forma gráfica el modelo matemático de la prótesis de mano robótica de 9GDL (Grados de libertad) y la cual opera bajo ciertas condiciones en Tiempo-Real No Estricto. Es así que para el desarrollo de la aplicación se escogió el lenguaje de programación Ada debido a su gran flexibilidad en cuanto a tiempo real se refiere ya que dichas características se encuentran definidas en el lenguaje y además su entorno de desarrollo (GNAT Programming Studio) es uno de los más potentes y completos entre los IDE libres disponibles (38). Otro aspecto relevante en el proyecto es que actualmente existen pocos proyectos relacionados con aplicaciones 3D escritos en lenguaje Ada; por ello este trabajo pretende además crear un vinculo entre OpenGL (a través del empleo de la herramienta de software libre llamada Globe_3D la cual está basada en OpenGL) una de las interfaces 3D estándar más conocidas y el lenguaje de programación Ada, dando así la posibilidad de tener un entorno virtual con características de tiempo real, versátil y libre.

El proyecto es por lo tanto enfocado en el desarrollo de la parte correspondiente al entorno gráfico de un entrenador virtual que representará en forma gráfica tridimensional el modelo de la prótesis de mano robótica y que además reproducirá unas trayectorias articulares dadas con el fin de desarrollar tareas de agarre de un objeto presente en el entorno tridimensional, permitiendo así analizar, identificar y visualizar las habilidades de la mano 3D. De esta forma se logrará el desarrollo de un entorno virtual libre, intuitivo y fácil de emplear, con el fin de que el paciente pueda desde cualquier computador y solo con la instalación de un hardware de acondicionamiento de señales de bajo costo, entrenarse y mantener sus conexiones nerviosas en buen estado, preparándolas para la implantación de la prótesis real, lográndose así una aplicación cuyo propósito es ayudar en la rehabilitación de pacientes amputados de mano.

Cabe anotar que este proyecto se delimita exclusivamente a la creación del entorno virtual y a la verificación de su funcionamiento a través de distintos agarres de la mano 3D virtual mediante simulación, la adecuación de las señales electromiográficas al igual que su acople con el entorno virtual corresponden a un trabajo futuro de este proyecto.

A continuación se describen en forma detallada la arquitectura del software, el concepto de tiempo real y todo el proceso de diseño e implementación llevado a cabo y con el cual se logró el desarrollo de la aplicación software.

4.1 ARQUITECTURA DEL SOFTWARE.

Una Arquitectura de Software consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para un sistema de información. La arquitectura software define, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación ente ellos; Se selecciona y diseña con base en objetivos y restricciones.

Existen al menos tres vistas absolutamente fundamentales en cualquier arquitectura:
La visión **estática**: describe qué componentes tiene la arquitectura.
La visión **funcional**: describe qué hace cada componente.
La visión **dinámica**: describe cómo se comportan los componentes y cómo interactúan entre sí.

Las vistas o modelos de una arquitectura pueden expresarse mediante uno o varios lenguajes. El más obvio es el lenguaje natural, pero existen otros lenguajes tales como UML, diagramas de estado, los diagramas de flujo de datos, etc (39) (40).

Con base al contenido teórico anteriormente citado, se procede a detallar cada una de las tres vistas fundamentales que conforman la arquitectura del entorno virtual desarrollado, el cual corresponde a la arquitectura que se emplea para la implementación de aplicaciones 3D desarrolladas en lenguaje Ada.

4.1.1 Visión Estática de la Implementación de Aplicaciones 3D en Ada.

La visión estática se encarga de describir qué componentes tiene la arquitectura (39) (40), por lo tanto de acuerdo al estudio llevado a cabo se indica que para el desarrollo de la aplicación software 3D en lenguaje Ada se hace uso de tres componentes principales que corresponden al IDE o entorno de desarrollo llamado GNAT Programming Studio o GPS; una herramienta de software libre con código abierto llamada Globe_3D escrita en lenguaje de programación Ada, basada en OpenGL y cuyo funcionamiento es en tiempo real; y por ultimo unas herramientas de software libre denominadas GtkAda y Glade las cuales funcionan en conjunto, brindando al usuario el uso de las características de

programación orientada a objetos del lenguaje Ada. En la Figura 4.1 se puede observar la forma en que se integran los elementos anteriormente nombrados y en los anexos 1 y 2 se describen detalladamente los procedimientos necesarios para lograr la integración de aquellas herramientas software (GPS, Globe_3D y GtkAda).

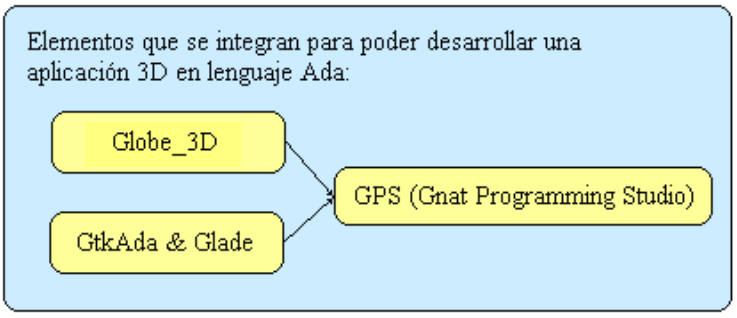


Figura 4.1 Vista Estática.
Fuente: Elaboración Propia.

4.1.2 Visión Funcional de la Implementación de Aplicaciones 3D en Ada.

Puesto que la visión funcional es aquella que describe qué hace cada componente de la arquitectura (39) (40), se procede entonces a indicar la funcionalidad de aquellas herramientas citadas en la visión estática descrita anteriormente.

GNAT Programming Studio: Corresponde al IDE o Entorno de Desarrollo Integrado Libre llamado también GPS el cual es el componente principal para el desarrollo de la aplicación, puesto que en él se lleva a cabo la edición de todo el código fuente en lenguaje Ada de la aplicación a desarrollar, se integran también en él las herramientas Globe_3D y GtkAda, igualmente en GPS se escriben todas las sentencias necesarias para lograr la interacción entre los diversos elementos que conforman la aplicación 3D final. En la siguiente Figura 4.2 se muestra la interfaz de Gnat Programming Studio (38).

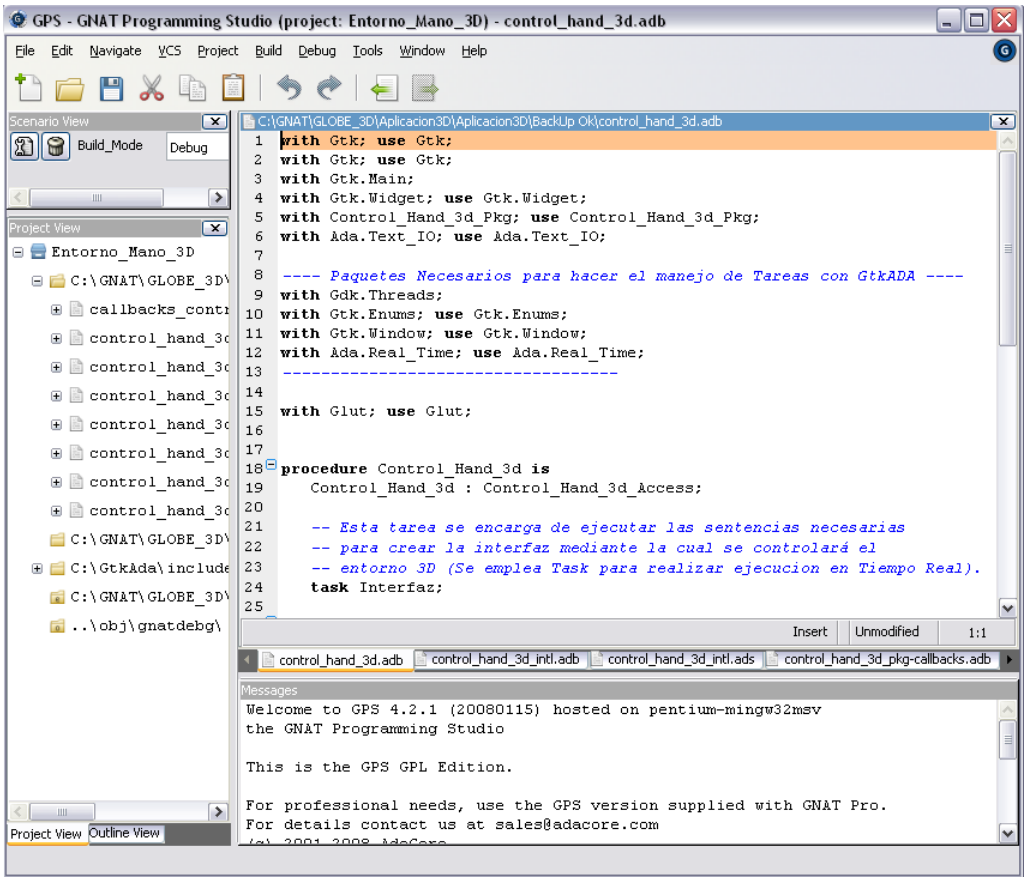


Figura 4.2 Interfaz del IDE GNAT Programming Studio.

Globe_3D: Motor de renderizado para la visualización de escenas tridimensionales en tiempo real, permite la construcción de gráficas 3D para diversas plataformas (Windows, Linux y MacOS), su código es abierto y escrito en lenguaje de programación Ada, su estructura está basada en *OpenGL* (*Open Graphics Library*) por lo cual permite realizar modelado y simulación de formas u objetos en tres dimensiones. Para poder hacer uso de Globe_3D éste se debe incluir en un proyecto de GPS para luego desarrollar todo el código de sentencias OpenGL de aquello que se desea modelar y simular en 3D (16). En la siguiente Figura 4.3 se puede apreciar la representación en 3D del modelo matemático de la prótesis de mano robótica de 9GDL.

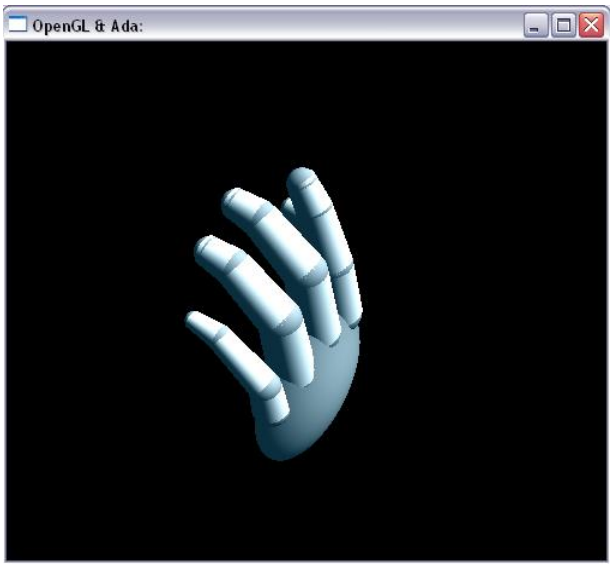


Figura 4.3 Entorno 3D de la Aplicación creada usando Globe_3D.
Fuente: Elaboración Propia.

GtkAda y Glade: GtkAda es una librería para la creación de interfaces gráficas de usuario (GUI) que provee un amplio conjunto de Widgets (elementos de la interfaz, por ejemplo botones, cuadros de texto, barras de menú, barras de progreso, ventanas, cuadros de diálogo, etc.) los cuales usan las características de la programación orientada a objetos que brinda el lenguaje Ada a los desarrolladores. La Figura 4.4 muestra la interfaz por medio de la cual se realiza el control de las distintas articulaciones que conforman el modelo de la mano 3D.

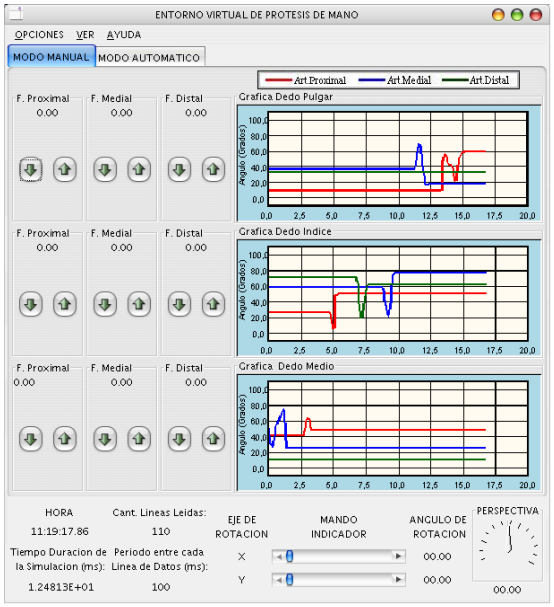


Figura 4.4 Interfaz de la Aplicación desarrollada usando GtkAda y Glade.
Fuente: Elaboración Propia.

Para facilitar el desarrollo de las interfaces gráficas de usuario mediante la librería GtkAda se hace uso de GtkGlade el cual es un software que permite al usuario diseñar de manera visual el acabado final de la GUI de la aplicación a ser desarrollada (41), el código del proyecto creado en Glade es en XML que debe ser posteriormente transformado a lenguaje Ada mediante un archivo herramienta de transformación llamado Gate o mediante el botón Build de la ventana principal de Glade (lo cual generará varios archivos .adb y .ads cuyos nombres dependen del nombre del programa creado en Glade y del nombre que se le haya dado a la ventana principal de la GUI desarrollada) (42).

4.1.3 Arquitectura de las Herramientas Software que se Integran en GPS.

Para poder entender la arquitectura que permite la implementación del entorno virtual 3D desarrollado en lenguaje Ada se considera necesario describir la arquitectura de las dos herramientas software (GtkAda y Globe_3D) que se integran en el proyecto que se lleva a cabo en el entorno de desarrollo Gnat Programming Studio.
La Figura 4.5 muestra cómo se interrelacionan los diversos elementos de los cuales se componen y hacen uso GtkAda y Globe_3D.

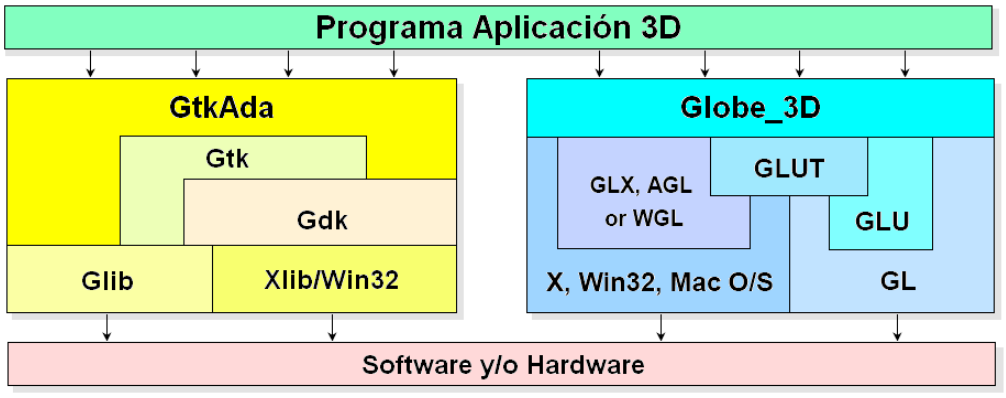


Figura 4.5 Arquitectura General de la Implementación de Aplicaciones 3D en Ada mediante GtkAda y Globe_3D.
Fuente: (43), (44).

4.1.3.1 Arquitectura de GtkAda.

GtkAda es la herramienta software empleada para la creación de la GUI, dicha herramienta se compone de tres librerías gtk, gdk y glib. (43), (45).

Glib no es una librería para gráficos, ésta brinda soporte para listas, h-tablas, manejo de hilos, etc. Es altamente optimizada, independiente de la plataforma y la mayoría de su contenido está en lenguaje Ada (46).

Gdk corresponde a la parte de GtkAda que es dependiente del sistema. Su implementación es completamente diferente en los sistemas Win32 y sistemas X11, aunque las interfaces son de hecho iguales. Provee un conjunto de funciones para el dibujado de líneas, rectángulos, mapas de pixeles sobre la pantalla, manipulación de colores, etc (46).

Gtk es una librería de alto nivel. Es independiente de la plataforma, y lleva a cabo todo el proceso de dibujado a través de llamadas a la librería Gdk. En ella se definen los widgets u objetos de alto nivel los cuales están completamente creados con una jerarquía de programación orientada a objetos (46).

4.1.3.2 Arquitectura de Globe_3D.

Globe_3D es la herramienta empleada para el desarrollo de gráficos tridimensionales el cual está basado en OpenGL (16), debido a ello su arquitectura se compone de aquellos elementos de dicha API(Interfaz de Programación de Aplicaciones) que corresponde a una librería de funciones que permite visualizar gráficos 3D en las aplicaciones, la gran ventaja que brinda esta librería es que la aplicación desarrollada es independiente del sistema operativo y hardware disponible; por lo tanto, esta librería se ejecuta a la par con el programa independientemente de la capacidad gráfica de la máquina que se use. Esto significa que la ejecución se dará por software a no ser que se cuente con hardware gráfico específico en la máquina, sin embargo si se cuenta con una tarjeta aceleradora de gráficos por supuesto se gozará de una ejecución muchísimo más rápida en tiempo real (47). Con base a lo anterior se puede aclarar entonces que la arquitectura de Globe_3D en cuanto al soporte de gráficos se divide en cuatro partes funcionales:

GL, librería que proporciona todo lo necesario para acceder a las funciones de dibujado de OpenGL.

GLU (OpenGL Utility Library), corresponde a una librería de utilidades que proporciona acceso rápido a algunas de las funciones más comunes de OpenGL, a través de la ejecución de comandos de más bajo nivel, perteneciente a la librería OpenGL

propiamente dicha. Es decir simplifica tareas tales como el graficado de superficies (por ejemplo: esferas, conos, cilindros, etc.).

GLUT (OpenGL Utility Toolkit) proporciona una interfaz independiente de la plataforma para crear aplicaciones de ventanas totalmente portables, al igual que también permite facilitar el manejo de animaciones y eventos.

GLX (para Sistemas X Windows común en plataformas Unix), WGL (para sistemas Microsoft Windows), AGL (para sistemas Apple Macintosh) son librerías que proporcionan acceso a OpenGL para poder interactuar con el sistema de ventanas del sistema operativo empleado (48), (44).

4.1.4 Visión Dinámica de la Implementación de Aplicaciones 3D en Ada.

Puesto que la visión dinámica describe cómo se comportan los componentes y cómo interactúan entre sí (39) (40)**,** se considera necesario entonces citar los elementos que conforman la aplicación creada a partir del proyecto en GPS de una forma modular. Como se ha descrito anteriormente, la librería GtkAda brinda los elementos (widgets) adecuados para poder interactuar con el entorno 3D desarrollado; para hacer uso de dichos elementos se emplean los archivos de la GUI desarrollada en Glade los cuales tienen la siguiente estructura:

- Archivo Principal: Tiene el mismo nombre del archivo de proyecto creado en Glade <nombre del programa>.adb, se ha definido así por conveniencia para que el usuario pueda visualizar su GUI; dicho archivo contiene el código de inicialización y creación del objeto de alto nivel (objeto principal) contenido en el archivo XML (46). En el Archivo Principal se deben incluir las sentencias necesarias para permitir la ejecución de tareas con GtkAda y además hacer el llamado al lazo o bucle principal de OpenGL, con el fin de permitir que tanto el Entorno 3D como la GUI se puedan ejecutar en paralelo.
- Archivo manejador principal: Este archivo llamado callbacks_<nombre del programa>.ads contiene todo el paquete manejador de instanciaciones de objetos gtkada (46) necesitados por la aplicación para el manejo de dichos objetos.
- Archivos de objetos de alto nivel: Para cada objeto de alto nivel, Gate genera un paquete <widget>_Pkg dentro de los archivos <widget>_pkg.ads y .adb (46); dichos archivos contienen todas las llamadas necesarias para crear los objetos que fueron ubicados dentro del objeto de alto nivel diseñado en Glade.
- Archivo de señales de objetos de alto nivel: Estos son los archivos más importantes creados por Gate. Cada uno es llamado <widget>_pkg-callbacks.adb y .ads (46); en

dichos archivos se incluye el código de las acciones que se deben llevar a cabo cada vez que se generen eventos o callbacks en los diversos elementos de la aplicación. En estos archivos se deben incluir las sentencias de código que se encargan de modificar las variables que controlan las acciones o características de los objetos del entorno 3D, es así como se establece la comunicación entre la GUI y el Entorno 3D.

En cuanto a la parte gráfica tridimensional, anteriormente se describió que Globe_3D permite hacer uso de las capacidades de la librería OpenGL para el desarrollo del entorno 3D, para ello se procede a crear dos archivos los cuales contendrán las sentencias para el modelado y simulación 3D mediante el llamado a funciones o procedimientos OpenGL del motor Globe_3D. En el trabajo realizado, se procedió a llamar a dichos archivos Entorno3D_procs.adb y .ads. En el archivo .adb se encuentra todo el código necesario para la creación del modelo y objetos 3D del entorno y en el archivo .ads se incluyen las declaraciones e inicializaciones de las variables, funciones y procedimientos que se emplean para el desarrollo del modelo y objetos tridimensionales.

En la Figura 4.6 se puede apreciar la interacción entre las librerías GtkAda con los archivos de la GUI desarrollada en Glade, e igualmente la interacción entre la librería Globe_3D y los archivos de código OpenGL, que al incluirlos todos al proyecto en Gnat Programming Studio permiten el desarrollo del entorno virtual 3D.

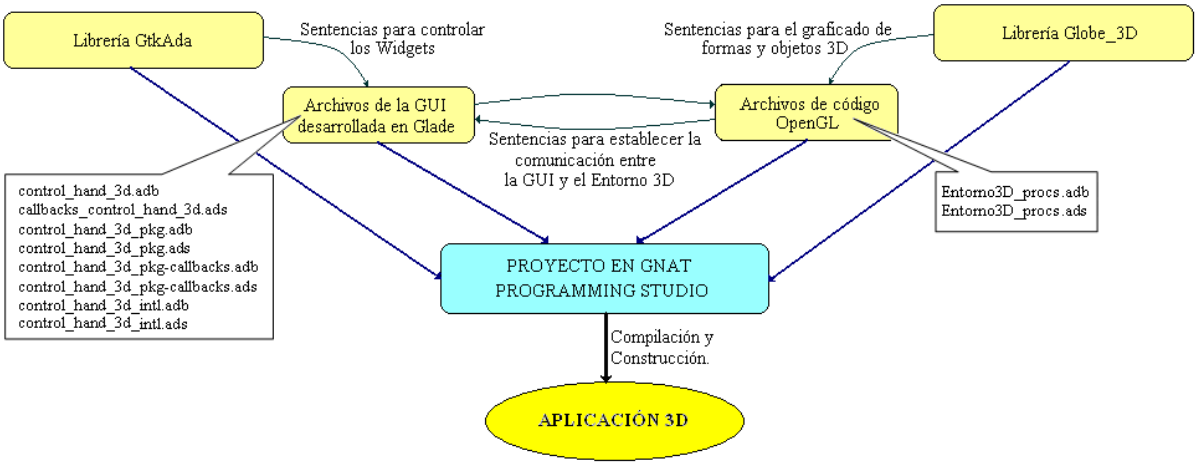


Figura 4.6 Interacción entre los elementos que permiten la Implementación de Aplicaciones 3D en Ada.
Fuente: Elaboración Propia.

4.2 CONCEPTO DE TIEMPO REAL Y SU USO EN LA APLICACIÓN EN LENGUAJE ADA.

Ada es un lenguaje de programación estándar ISO[1], es orientado a objetos, concurrente y de tiempo real. Debido a las grandes prestaciones en eficiencia y seguridad que brinda ha sido empleado especialmente en el desarrollo de aplicaciones grandes y de gran tiempo de vida en las cuales la fiabilidad y la eficiencia resultan ser primordiales, particularmente en sistemas de tiempo real. La esencia básica del lenguaje Ada es que añade la flexibilidad de la programación orientada a objetos con la seguridad de una buena comprobación de tipos de datos (característica de un lenguaje de programación que controla que no sean violados los tipos de datos en un código de programación) que lo enmarca, al igual que el soporte que proporciona para la programación de sistemas concurrentes y de tiempo real (22), (49). Es claro que al emplear un potente lenguaje como lo es Ada gracias a la flexibilidad que brinda para el desarrollo de sistemas de tiempo real en conjunto con una de las librerías que durante años se ha consolidado como la librería por excelencia para desarrollar aplicaciones 2D y 3D con independencia de la plataforma o el hardware gráfico tal como lo es OpenGL (47), se puede lograr una aplicación software fiable, eficiente, de gran calidad y portable capaz de poderse ejecutar en una amplia variedad de arquitecturas y de soportes gráficos, sin que el resultado se vuelva inconsistente. Para lograr soporte de actividades concurrentes y de tiempo real en las aplicaciones Ada, el lenguaje tiene definido el modelo de tareas (*task*) y paquetes especiales (Ada.Real_Time) para tales fines (50). A continuación se define el concepto de tiempo real y se describe la forma como fue empleado dicho concepto en el desarrollo del entorno virtual. Existen muchas interpretaciones sobre la naturaleza de un sistema de tiempo real; sin embargo, todas tienen en común la noción de tiempo de respuesta: El tiempo que precisa el sistema para generar la salida a partir de alguna entrada asociada.

4.2.1 Tipos de sistemas de tiempo real.

En el campo del diseño de sistemas de tiempo real se distinguen frecuentemente entre Sistemas de Tiempo Real Estrictos (*Hard Real Time Systems*) y Sistemas de Tiempo Real No Estrictos (*Soft Real Time Systems*); los sistemas de tiempo real estrictos son aquellos en los que es absolutamente imperativo que las respuestas se produzcan dentro del tiempo límite especificado. Los sistemas de tiempo real no estrictos son aquellos en los que los tiempos de respuesta son importantes pero el sistema seguirá funcionando correctamente aunque los tiempos límite no se cumplan ocasionalmente (22), (50).
Características básicas de un sistema de tiempo real:

- Grandeza y complejidad.

[1] *International Standards Organization* (Organización Internacional de Estandares).

- Manipulación de números reales.
- Fiabilidad y seguridad.
- Control concurrente de componentes separados del sistema.
- Control en tiempo real.
- Interacción con interfaces hardware.
- Implementación eficiente.

4.2.2 El modelo de tareas del lenguaje Ada.

Con el fin de implementar el modelo computacional del sistema, resulta necesario utilizar un lenguaje de programación que soporte las características para tal fin; de acuerdo a la investigación realizada se determinó que el lenguaje Ada resultaba ser el apropiado, puesto que permite la programación de sistemas de tiempo real, para lo cual en particular brinda las siguientes características:

- Permite implementar fácilmente tareas (*Task*) y objetos compartidos protegidos (PSOs – *Protected Shared Objects*).
- Soporta esquemas basados en prioridad (admite la asignación de prioridades),
- Permite la distribución de tareas y PSOs sobre un sistema (51).

Uno de los mayores problemas asociados con la programación concurrente surge de la interacción entre diversos procesos, puesto que rara vez existen procesos que se ejecutan independientemente, es decir, comúnmente algún(os) proceso(s) requiere(n) establecer comunicación con otro u otros procesos, y generalmente ciertos procesos se deben llevar a cabo en simultáneo con otros (22); por ello en el lenguaje Ada se ha definido el *Modelo de Tareas*, en el cual las tareas corresponden a una serie de actividades que se ejecutan en paralelo. Al estar estas facilidades definidas dentro del lenguaje y no a través de llamadas a un sistema operativo se logran grandes ventajas tales como una mayor portabilidad y mantenibilidad del software (52). Además de ello, si se requiere llevar a cabo comunicación sincrónica entre las tareas, entonces en la especificación de la tarea deben declararse accesos (*entries*) que pueden ser llamados desde otras tareas, uno de los métodos más empleados para llevar a cabo comunicación sincrónica es el paso de mensajes o datos entre tareas el cual es denominado en Ada como *Rendezvous* (22). Con la comunicación asincrónica no se requieren *entries*, en lugar de ello suelen utilizarse objetos compartidos protegidos, un objeto compartido define datos que pueden ser accedidos de forma mutuamente exclusiva por las tareas. En la Figura 4.7 se define un procedimiento “Ejemplo” en el cual se crean dos tareas “A” y “B”, que corresponden a dos procesos que se ejecutarán en forma concurrente una vez se lleve a cabo la ejecución del procedimiento en el cual se encuentran definidas.

```
 1
 2 procedure Ejemplo is
 3
 4 Especificación o declaración de las tareas A y B
 5    task A; -- especificación, sin interfaz con otras tareas
 6
 7    task B is
 8     declaraciones de la interfaz con otras tareas
 9    end B;
10
11 Cuerpo de las tareas A y B
12    task body A is -- cuerpo
13      declaraciones locales
14    begin
15      secuencia de instrucciones;
16    end A;
17
18    task body B is -- cuerpo
19      declaraciones locales
20    begin
21      secuencia de instrucciones;
22    end B;
23
24 begin -- A y B empiezan a ejecutarse aquí
25 Secuencia de instrucciones en paralelo con A y B
26 end Ejemplo; -- No termina hasta que finalizan A y B
```

Figura 4.7 Modelo básico de tareas en Ada
Fuente: (22).

4.2.3 Comunicación sincrónica basada en el paso de mensajes "*Rendezvous*".

El empleo de tareas en el desarrollo de un sistema invariablemente conlleva a que algunas tareas intercambien datos, es decir se comuniquen con el fin de que el sistema pueda funcionar correctamente. La comunicación de datos en Ada como se expresó anteriormente, es en general de dos tipos, basada en variables compartidas o basada en el mecanismo de paso de mensajes (*Rendezvous*). En el caso particular del paso de mensajes involucra el intercambio explicito de datos entre dos procesos por medio de un mensaje que pasa desde un proceso a otro. En el caso de las variables compartidas, aunque resultan ser un medio sencillo de paso de información entre procesos, su empleo no restringido conlleva a que el sistema desarrollado no sea del todo seguro (22).

Puesto que en un sistema existen procesos que se ejecutan independientemente, también existen situaciones en las cuales algunos procesos dependen o requieren comunicarse con otros, en aquellas situaciones es necesario que dos o más procesos lleven a cabo su ejecución en forma coordinada, se dice entonces que se requiere la sincronización entre procesos; que por lo general en la mayoría de los casos, la comunicación de datos necesitará sincronización, pues bien, el lenguaje Ada provee al programador del mecanismo denominado Rendezvous, en el cual la comunicación y la sincronización se encuentran estrechamente relacionadas e implementadas (22).

Las motivaciones para llevar a cabo la comunicación interprocesos mediante Rendezvous son cuatro:

- La comunicación de datos y el proceso de sincronización se consideran como actividades inseparables.
- El mecanismo de comunicación resulta ser simple y ameno para su análisis.
- Comunicación directa, en lugar de terceras partes que a su vez deben estar protegidas.
- Procesos más seguros, ya que se prescinde de variables compartidas pasivas, las cuales hacen que el sistema no sea del todo fiable.

En la Figura 4.8, se ilustra el mecanismo Rendezvous, en dicho caso se asume que unos datos deben ser pasados entre el proceso A “Tarea A” y el proceso B “Tarea B”. La transmisión de dichos datos se lleva a cabo cuando A “T” y B “Otra” están listos para comunicarse, se dice que un Rendezvous (paso de mensajes) toma lugar cuando los datos son pasados desde B hacia A o viceversa. El proceso que ejecuta sus sentencias primero, es decir aquel que queda listo primero para llevar a cabo la comunicación, se queda en estado de espera hasta que el otro proceso se encuentre listo para llevar a cabo el intercambio de datos. Una vez la comunicación se ha completado, el Rendezvous se interrumpe y ambos procesos continúan su ejecución independiente y concurrentemente, hasta que se requiera llevar a cabo el mecanismo de Rendezvous nuevamente (22).

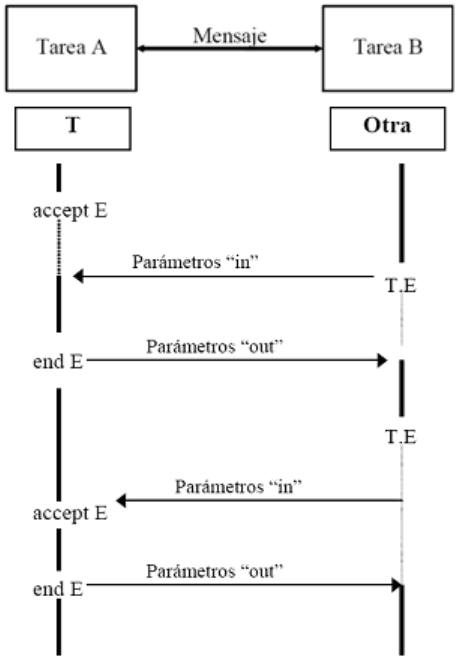


Figura 4.8 Esquema de comunicación empleando Rendezvous.
Fuente: (22).

En la Figura 4.9, se muestran dos modelos que corresponden al código básico para emplear el mecanismo de rendezvous en dos procesos, a saber “T” y “Otra_Tarea”. En este caso “Otra_Tarea” solicita o envía ciertos datos a “T” mediante el acceso *entry “*E” al proceso “T”, el cual procesa a través de la aceptación *accept* las instrucciones para la comunicación requerida con el proceso “Otra_Tarea”.

```
task T is
  entry E(parámetros);
end ;
```

```
task Otra_Tarea;
```

```
task body T is
   ...
begin
  ...
  accept E(...) do
   -- instrucciones
   -- para comunicación
  end E ;
  ...
end T ;
```

```
task body Otra_Tarea is
  ...
begin
  ...
  T.E (...) ;
  ...
end Otra_Tarea ;
```

Figura 4.9 Código base para la implementación del método Rendezvous en Ada. Fuente: (22).

4.2.4 Empleo de tareas en Ada para la ejecución en tiempo real de la Aplicación desarrollada.

En el software desarrollado, las tareas de mayor prioridad corresponden a las del graficado 3D, especialmente cuando se lleva a cabo la simulación en modo automático, modo en el cual se ejecutan tareas periódicas y su periodicidad está definida por el intervalo de simulación indicado por el usuario para todo el conjunto de variables articulares, el cual es definido antes de iniciar la simulación y que por defecto es de 100 milisegundos entre cada conjunto de datos de variables articulares; cada conjunto de datos está compuesto por nueve datos, uno para cada falange de los tres dedos modelados (pulgar, índice y corazón), los otros dos dedos (anular y meñique) siguen el movimiento desarrollado por el dedo corazón.

En la Figura 4.10, se muestra un esquema del código que lleva a cabo la labor de la simulación en forma automática del conjunto de datos de las variables articulares, las cuales son dadas a la aplicación en un archivo de texto plano. Como tal, se puede observar que se emplearon tareas con el fin de que dichas sentencias se ejecutaran en paralelo con otros componentes del programa, y también se le asignó una alta prioridad a dichas tareas, para así poder observar la simulación del modelo 3D en tiempo real. No fue necesario utilizar objetos compartidos protegidos porque no se requería realizar la protección de variables ni objetos, pero con el fin de realizar la comunicación de paso de

datos de valores articulares entre las tareas de lectura de datos “Tarea” y actualización del entorno 3D “Tarea_3D” se empleó el mecanismo conocido en Ada como *Rendezvous* (22), por lo cual si fue necesario emplear *entries*, en el esquema de código que se muestra a continuación se puede observar que el acceso (*entry*) que se empleó fue llamado “Actualizar_3D” y el cual es responsable del paso de datos entre las tareas con el fin de que el modelo 3D reciba el valor correspondiente para cada variable articular de los dedos de la mano 3D, y posteriormente mediante el uso de la función OpenGL denominada TimerFunc llevar a cabo la actualización gráfica del modelo tridimensional.

```
1  with Ada.Real_Time; use Ada.Real_Time;
2
3  -- Procedimiento en el cual están definidas las tareas que llevan a cabo
4  -- el proceso de simulación de forma periódica
5  procedure TimerFuncCb ( InterV : in Integer) is
6
7
8     -- INICIA ESPECIFICACION DE TAREA Y TAREA3D
9        -- Tarea que lleva a cabo la lectura del dato a simular
10       -- el cual se encuentra presente en el archivo que contiene el conjunto de
11       -- valores de las variables articulares.
12       task Tarea is
13          -- Prioridad que se le otorga a la tarea
14          pragma Priority (2);
15       end Tarea;
16
17
18       -- Tarea que lleva a cabo el paso de valores al modelo 3D y su respectiva
19       -- actualización grafica.
20       task Tarea3D is
21          -- Prioridad que se le otorga a la tarea
22          pragma Priority (1);
23
24          -- Acceso que se le brinda a Tarea3D para que pueda sincronizarse
25          -- con Tarea y así poder realizar bien el intercambio de datos.
26          entry Actualizar_3D;
27       end Tarea3D;
28       -- FIN DE ESPECIFICACION DE TAREA Y TAREA3D
29
30
31       -- INICIA CUERPO DE TAREA Y TAREA3D
32       task body Tarea is
33           T_Periodo : Time_Span := Milliseconds(20);
34           T_Inicio : Time;
35           -- Otras declaraciones.
36       begin
37
38          T_Inicio := Clock;
39
40          Ctrl_Sim :
41          loop
42             if "No se ha finalizado de leer los datos" then
43             -- Se hace el llamado a la Tarea3D, específicamente a
44             -- Actualizar 3D.
45              Tarea3D.Actualizar_3D;
46
47              -- Se incrementa el valor del conjunto de datos del archivo
48              -- (para leer los siguientes datos).
49
```

```
50              else
51               -- Se finaliza la ejecución de Tarea.
52              end if;
53
54
55              if "Botón Iniciar Simulación esta activo" then
56               -- Se hace el llamado a la función de OpenGL TimerFunc,
57               -- para actualizar el grafico 3D;
58              else
59               -- Terminar la ejecución de las tareas: Tarea y Tarea3D
60              end if;
61
62
63              T_Inicio := T_Inicio + T_Periodo;
64              delay until T_Inicio;
65              -- Se publican algunos datos de interés.
66
67           end loop Ctrl_Sim;
68
69        end Tarea;
70
71
72
73        task body Tarea3D is
74           T3D_Inicio    : Ada.Real_Time.Time;
75           Periodo3D  : constant Time_Span := Milliseconds (PeriodoIndicado);
76
77        begin
78           T3D_Inicio := Clock;
79
80           if "Fila corresponde a la siguiente fila de datos" then
81              accept Actualizar_3D do
82
83              -- Se pasan los datos de la línea seleccionada en el archivo a
84              -- simular, a las respectivas articulaciones de los
85              -- dedos del Modelo 3D.
86
87              end Actualizar_3D;
88
89           else
90              -- Se termina la ejecucion de Tarea3D
91           end if;
92
93              T3D_Inicio := T3D_Inicio + Periodo3D;
94              delay until T_Inicio;
95              -- Se publican algunos datos de interés.
96
97        end Tarea3D;
98
99  begin
100       null;
101 end TimerFuncCb;
```

Figura 4.10 Esquema de código en Ada para la simulación automática de datos de variables articulares.
Fuente: Elaboración Propia.

4.2.5 Adecuando GtkAda para el manejo de tareas en Ada.

Además de emplear el modelo de tareas de Ada para la ejecución en tiempo real de la aplicación, también resultó necesario lograr aplicar la ejecución de tareas en GtkAda, para lo cual se tuvo que incluir algunas sentencias (46), con el fin de lograr la ejecución en paralelo con otros elementos de la aplicación, como por ejemplo las sentencias de código de funciones OpenGL encargadas del modelado del entorno 3D.

Las sentencias anteriormente nombradas fueron incluidas en el archivo principal del proyecto creado en Glade <nombre del programa>.adb (en este caso Control_Hand_3D.adb), además de dichas sentencias se debe también incluir la sentencia de llamado al lazo o bucle principal de OpenGL, con el fin de permitir que tanto el Entorno 3D como la GUI se pudiesen ejecutar en paralelo. A continuación se muestra la Figura 4.11 que incluye el código de dicho archivo, para el caso particular de la aplicación desarrollada.

```
1  with Gtk; use Gtk;
2  with Gtk.Main;
3  with Gtk.Widget; use Gtk.Widget;
4  with Control_Hand_3d_Pkg; use Control_Hand_3d_Pkg;
5  with Ada.Text_IO; use Ada.Text_IO;
6
7  ---- Paquetes Necesarios para hacer el manejo de Tareas con GtkADA ----
8  with Gdk.Threads;
9  with Gtk.Enums; use Gtk.Enums;
10 with Gtk.Window; use Gtk.Window;
11 with Ada.Real_Time; use Ada.Real_Time;
12 -------------------------------------
13
14 with Glut; use Glut;
15
16
17 procedure Control_Hand_3d is
18    Control_Hand_3d : Control_Hand_3d_Access;
19
20    -- Esta tarea se encarga de ejecutar las sentencias necesarias
21    -- para crear la interfaz mediante la cual se controlará el
22    -- entorno 3D (Se emplea Task para realizar ejecucion en Tiempo Real).
23    task Interfaz;
24
25    task body Interfaz is
26    begin
27       Put_Line("Start Interfaz");
28
```

```
30          -- Gdk.Threads.* Permiten ejecutar Tareas en conjunto con
31          -- la aplicacion GtkADA (Con ello se permite aplicar Task
32          -- para llevar a cabo ejecucion paralela de codigo,
33          -- pudiendo así garantizarse Tiempo Real).
34          Gdk.Threads.G_Init;
35          Gdk.Threads.Init;
36          Gtk.Main.Set_Locale;
37          Gtk.Main.Init;
38          Gtk_New (Control_Hand_3d);
39          Show_All (Control_Hand_3d);
40          Gdk.Threads.Enter;
41          Gtk.Main.Main;
42          Gdk.Threads.Leave;
43          Put_Line("Exit of Interfaz");
44       end Interfaz;
45
46    begin
47
48       -- Esta función realiza el control principal del flujo del programa
49       -- a travez de GLUT, Que mediante "eventos" predefinidos irá
50       -- llamando a las funciones que han sido pasadas como Callbacks.
51       MainLoop;
52
53    end Control_Hand_3d;
```

Figura 4.11 Código del archivo Control_Hand_3D.adb
Fuente: Elaboración Propia.

4.3 DISEÑO DE SISTEMAS EN TIEMPO REAL EMPLEANDO EL MÉTODO HRT-HOOD.

Una etapa importante en el desarrollo de sistemas de tiempo real es la generación de un diseño consistente que satisfaga la especificación acreditada de los requisitos. En esto, los sistemas de tiempo real no difieren de las demás aplicaciones, aunque su escala en conjunto conlleva a problemas de diseño fundamentales. En esencia, todos los métodos de diseño incluyen una secuencia de transformaciones desde el estado de los requisitos iníciales hasta el código ejecutable. En cuanto a las fases habituales por la que se transcurre al respecto son (53) (50):

- Especificación de requisitos.
- Diseño arquitectónico.
- Diseño detallado.
- Implementación.
- Prueba.

En el presente trabajo se opto por utilizar la metodología HRT-HOOD (*Hard Real Time Hierarchical Object Oriented Design* o Diseño Jerárquico Orientado a Objetos de Sistemas Críticos de Tiempo Real) para construir el sistema en tiempo real que se encarga del correcto funcionamiento del entorno virtual de prótesis de mano 3D.

Este diseño se obtiene de desarrollar iterativa y concurrentemente tanto la arquitectura lógica como la arquitectura física. El diseño de la arquitectura lógica está destinado fundamentalmente a la satisfacción de los requisitos funcionales. En el diseño de la arquitectura física se relaciona la arquitectura lógica con los recursos de ejecución, se asignan atributos temporales a los objetos y se asegura el cumplimiento de los requisitos no funcionales en el sistema (requisitos de tiempos de respuesta, seguridad y fiabilidad de todo el sistema en general) (50).

En este proyecto se logró desarrollar adecuadamente la arquitectura lógica del sistema, pero en cuanto a la arquitectura física, puesto que para su desarrollo se requiere de la parte hardware que realice el enlace con el entorno virtual, la cual no es objeto de estudio para el presente proyecto, se aclara entonces que por dicha razón no se ha enfatizado mucho en el diseño de la arquitectura física.

4.3.1 Elementos de HRT-HOOD.

El sistema se diseña como una jerarquía de **objetos abstractos**

- un objeto se caracteriza por sus operaciones y su comportamiento (abstracción y ocultamiento de información)
- cada objeto se puede descomponer en otros de más bajo nivel
- modelo de objetos simple, sin herencia

Se puede **analizar** el **comportamiento temporal** si el entorno de ejecución es conocido y predecible

- los objetos tienen atributos temporales
- las relaciones entre objetos están restringidas para asegurar que el diseño se puede analizar

4.3.2 Objetos y Relaciones en HRT-HOOD.

Una aplicación de tiempo real diseñada con la metodología HRT-HOOD contendrá en su nivel terminal de descomposición objetos cíclicos, esporádicos, protegidos y pasivos. Los objetos activos no se permiten, ya que no son analizables, y con ello la aplicación sería de tiempo real no estricto. Pueden utilizarse en la descomposición del sistema principal, pero debe transformarse en uno de los anteriores antes de alcanzar el nivel terminal (53).

La Figura 4.12 muestra los elementos de un diseño HRT-HOOD. El objeto “Padre o Parent” ha sido descompuesto jerárquicamente en dos objetos: “Hijo1 o Child1” e “Hijo2 o Child2”. El objeto “Parent” es un objeto activo. Así lo indica la letra A en la esquina superior izquierda. Tiene dos operaciones: “Op1” y “Op2”. Estas operaciones son

implementadas mediante los objetos hijos. El hijo “Child1” invoca operaciones de un objeto Tío “Uncle” y el hijo “Child2”, este último un objeto pasivo. Un objeto tío es un objeto definido en un nivel superior de la descomposición. El diagrama muestra también los flujos de datos (53). Más adelante se detallarán en el diseño del sistema los respectivos diagramas de objetos.

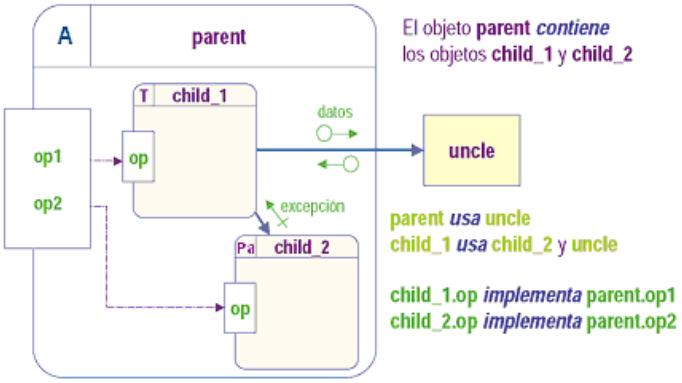


Figura 4.12 Elementos de un diseño HRT-HOOD.
Fuente: (53).

4.3.3 Tipos de objetos y sus características en HRT-HOOD.

La Figura 4.13 ilustra al lado izquierdo la representación gráfica de un objeto HRT-HOOD, en dicho diagrama se pueden observar cada uno de los elementos que lo constituyen; en el lado derecho se puede observar la representación en código Ada del objeto.

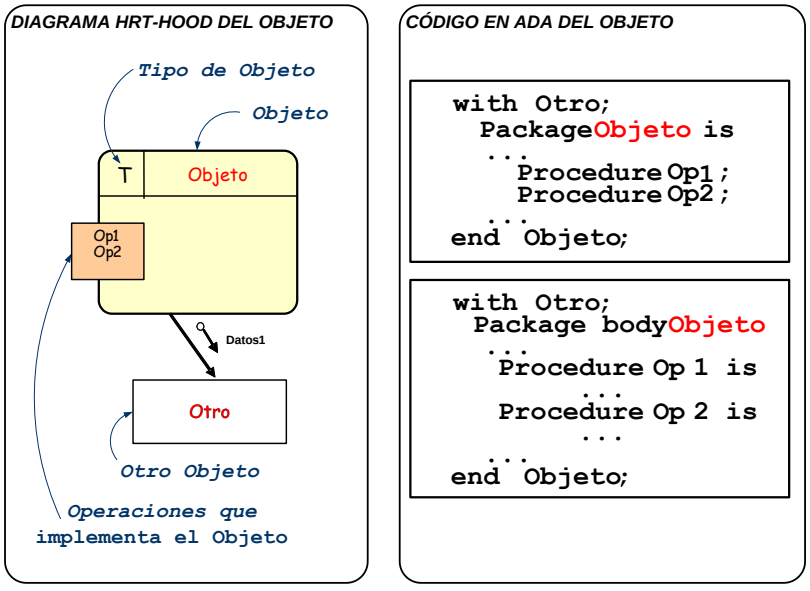


Figura 4.13 Representación gráfica de un objeto HRT-HOOD y su código en Ada.
Fuente: Elaboración Propia.

Pasivos

- No controla cuándo se ejecutan sus operaciones al ser éstas invocadas por otros objetos (es decir, no tiene restricciones de sincronización).
- No invoca operaciones en otros objetos de forma espontánea (es decir, no tiene tareas).

Este tipo de objeto se ilustra en la Figura 4.14.

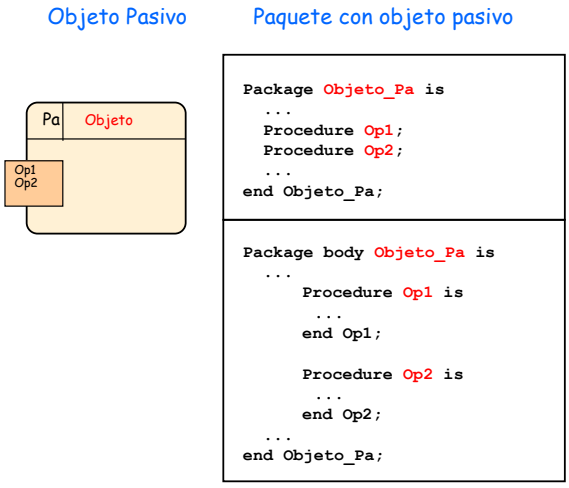


Figura 4.14 Objeto pasivo.
Fuente: (54).

Activos

- Pueden controlar cuándo se ejecutan sus operaciones.
- Pueden invocar operaciones de otros objetos espontáneamente.

En la Figura 4.15 y Figura 4.16 se muestra este tipo de objeto.

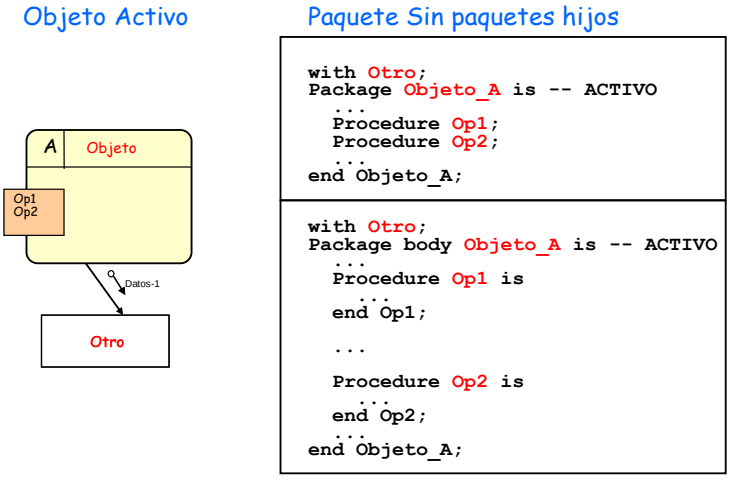


Figura 4.15 Objeto activo sin paquetes hijos.
Fuente: (54).

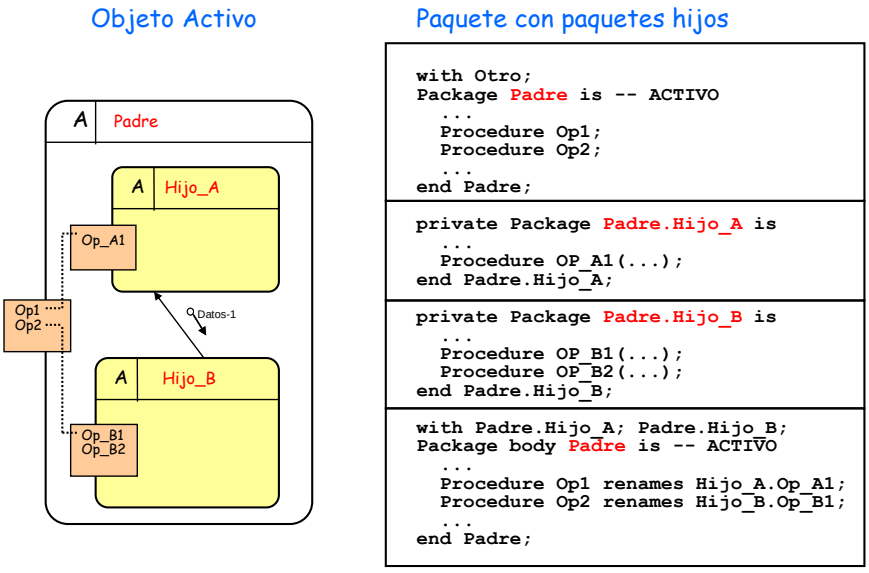


Figura 4.16 Objeto activo con paquetes hijos.
Fuente: (54).

Protegidos

- Pueden controlar cuándo se ejecutan sus operaciones.
- No invocan operaciones de otros objetos espontáneamente.

La Figura 4.17 permite observar este tipo de objeto.

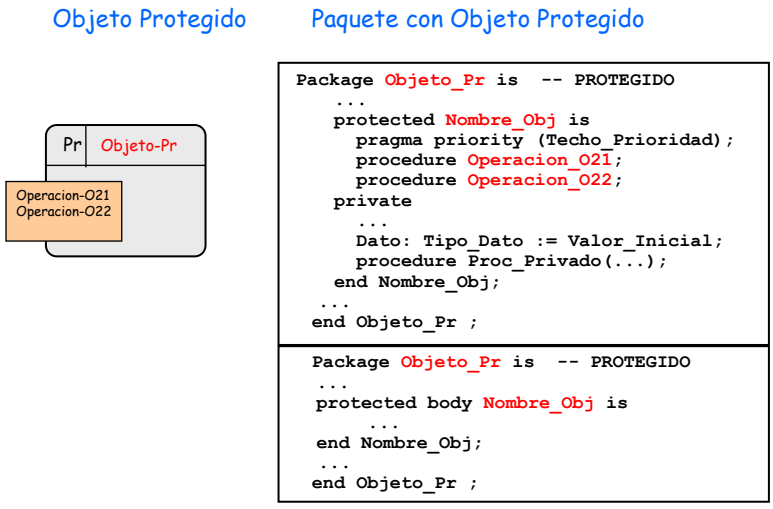


Figura 4.17 Objeto protegido.
Fuente: (54).

Cíclicos

- Sus operaciones se ejecutan a intervalos regulares, se representación se puede observar en la Figura 4.18.

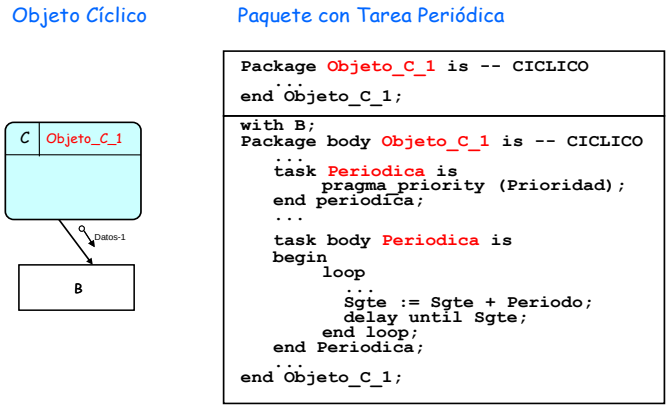


Figura 4.18 Objeto cíclico.
Fuente: (54).

Esporádicos

- Sus operaciones se ejecutan cuando ocurre un suceso externo o interno; se implementa así como lo muestra la Figura 4.19.

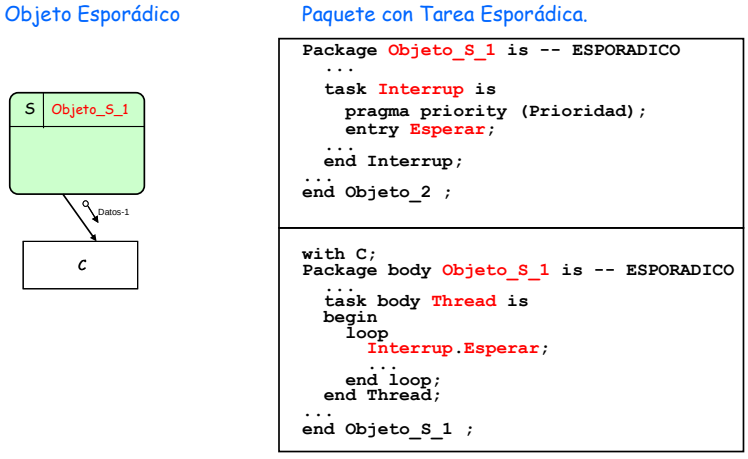


Figura 4.19 Objeto esporádico.
Fuente: (54).

4.4 APLICANDO HRT-HOOD EN EL DISEÑO DEL ENTORNO VIRTUAL DE PRÓTESIS DE MANO.

La función del sistema es simular algunos movimientos de la mano humana por ejemplo agarre tipo cilíndrico, gancho, punta, entre otros; mediante un entorno tridimensional de una mano virtual desarrollada con herramientas de software libre, tomando como elementos de entrada las trayectorias articulares que simulen las intenciones de movimiento del paciente. El principal requisito del sistema es simular en tiempo real no estricto las intensiones de movimiento mediante los cambios de las variables articulares, los cuales se deben ver reflejados en forma gráfica tanto en el modelo de mano 3D como en las respectivas gráficas de Valor_Angular vs. Tiempo para las diversas falanges de los dedos. En la Figura 4.20 se puede observar un diagrama esquemático del sistema.

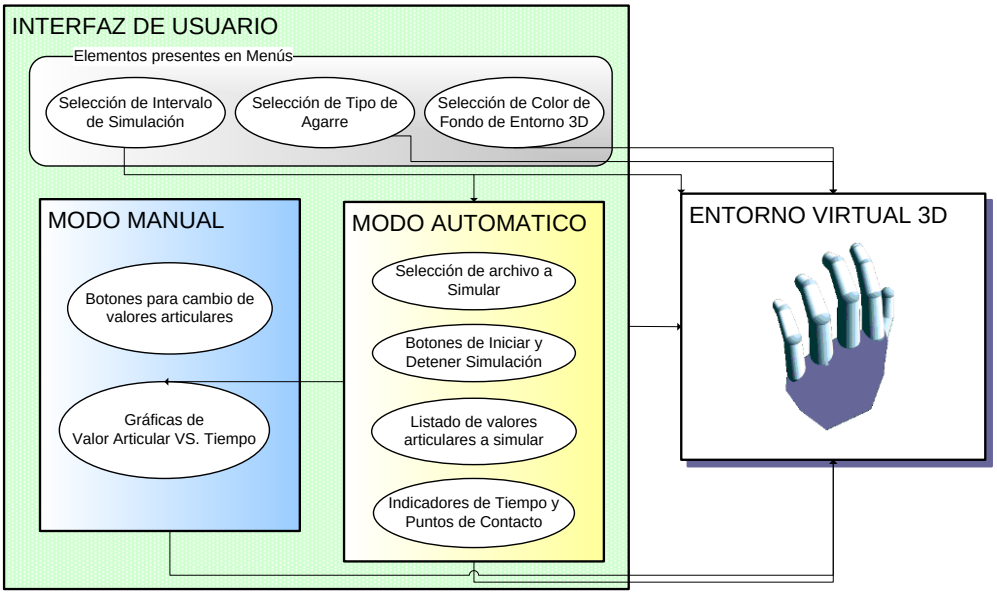


Figura 4.20 Diagrama Esquemático del Entorno Virtual de Prótesis de Mano.
Fuente: Elaboración Propia.

4.4.1 Especificación de requisitos.

Básicamente a nivel lógico el sistema se subdivide en dos partes importantes, el modo de ejecución manual y el modo de ejecución automático los cuales a su vez están relacionados en forma directa con la interfaz de usuario y en forma indirecta con el entorno 3D, dichas partes permitirán realizar la especificación de los requisitos funcionales y los requisitos no funcionales de la aplicación. A nivel funcional el sistema permitirá realizar las siguientes operaciones:

4.4.1.1 Requisitos funcionales.

La especificación funcional del sistema se puede dividir en dos componentes, a saber, Interfaz de usuario y Entorno virtual 3D.

INTERFAZ DE USUARIO:

- Atender eventos en elementos presentes en la barra de menús.
- Cambio de perspectiva en el entorno 3D.
- Selección del modo de simulación a través de las pestañas (Modo Manual y Modo Automático).
- Mostrar indicadores de tiempo.

- *Modo Manual*:
  - Aumentar o disminuir el valor de la variable articular de cualquiera de las articulaciones de los dedos.
  - Graficar en 2D los cambios angulares de las articulaciones de los dedos.
- *Modo Automático*:
  - Seleccionar el archivo de datos a simular.
  - Atender eventos de botones de iniciar y detener simulación
  - Listar y desplegar los datos de valores articulares a simular.
  - Indicar puntos de contacto de los dedos con el objeto.

ENTORNO VIRTUAL 3D:

- Desplegar en 3D el modelo de prótesis de mano.
- Desplegar en el entorno el objeto seleccionado en la interfaz de usuario.
- Atender eventos del teclado y del Mouse.

4.4.1.2 Requisitos no funcionales.

La temporización y la seguridad son dos requisitos no funcionales en el diseño de un sistema de tiempo real. Se tratará el primero, el tema de la seguridad no será descrita debido a que un mal funcionamiento del sistema (por ejemplo que se presenten errores debido a inconsistencia de datos a simular, bloqueos del sistema o se aborte la ejecución de forma inesperada) no conlleva a un riesgo potencialmente peligroso para el usuario, sin embargo se aclara que se eliminaron los posibles problemas de seguridad a que hubiere lugar, mediante la aplicación de excepciones la cual es una de las facilidades que le brinda el lenguaje Ada al programador. En cuanto a los requisitos temporales de se considerarán:

- Periodos de monitorización.
- Plazo de parada.

4.4.1.2.1 Periodos de monitorización.

En la metodología HRT-HOOD, la operación sobre los dispositivos es realizada por manejadores implementados como tareas, las cuales son implementadas en el interior de los respectivos procedimientos.

En cuanto a periodos de tiempo de monitorización se consideran relevantes a nivel general los correspondientes a los procesos de modelado y graficado 3D, y con mayor prioridad en el caso del modo de simulación automático. Estos periodos de tiempo fueron establecidos para que se pudiesen cumplir en un computador con las siguientes características:

Procesador: AMD Athlon 3500+
Memoria RAM: 1.0 GB
Tarjeta Aceleradora de Gráficos: ATI de 256MB.
Sistema Operativo empleado: Microsoft Windows XP – Service Pack 2.

Para el sistema desarrollado se contemplaron cuatro acciones importantes respecto a periodos de monitorización las cuales son: Graficado 3D, Paso de datos de valores articulares, Graficado 2D y Actualización del entorno 3D.

En la Tabla 4.1 se listan los tiempos de las acciones que se consideraron relevantes en cuanto a tiempos de monitorización. En dicha tabla se especifica el tipo de acción o tarea, el valor mínimo de periodo que puede cumplir dicha tarea y el plazo mínimo que tiene disponible para llevar a cabo la ejecución de las sentencias requeridas.

Acción o Tarea	***Tipo (Periódico o Esporádico)***	***Periodo (msg)***	***Plazo (msg)***
Graficado 3D	Periódico	100	85
Paso de datos de valores articulares	Periódico	100	85
Graficado 2D	Periódico	90	90
Actualización de Entorno 3D	Esporádico	1000	100

Tabla 4.1 Lista de tiempos de las tareas relevantes en el sistema.

Fuente: Elaboración Propia.

El Graficado 3D y el Paso de datos de valores articulares son tareas periódicas de alta prioridad, estableciéndose la mayor prioridad para el Graficado 3D, dichas tareas se deben llevar a cabo en forma sincronizada, su sincronismo se logra mediante el empleo del método de paso de mensajes conocido en Ada como Rendezvous. Ambas poseen un periodo *P* de 100ms por defecto en modo de ejecución automático y debido a la latencia *L* de 15msg que existe en la ejecución de las tareas, el plazo *D* de las tareas que soportan

estas sentencias ha de cumplir que $D \leq T - L$, por tanto se determina que el plazo para su ejecución es de 85msg. En modo de ejecución manual el Graficado 3D está determinado por la Actualización del Entorno 3D, y en cuanto al Paso de datos de valores articulares no opera en este modo, ya que los valores articulares dependen de las acciones que el usuario lleve a cabo para el movimiento de las falanges por medio del teclado o bien por medio de los botones que controlan el movimiento de las articulaciones.

El Graficado 2D es una tarea de menor prioridad que las dos tareas anteriores, y se ejecuta siempre ya sea en modo automático o manual, debido a que su ejecución no consume muchos recursos del sistema, se considera adecuado un periodo de 90ms y de igual manera para el plazo, puesto que no se encontró una latencia considerable para dicha tarea.

La Actualización del Entorno 3D es una de las tareas más importantes en el sistema desarrollado, y opera siempre en ambos modos (manual o automático), se encarga básicamente de dar a conocer al usuario en forma gráfica el estado actual en el cual se encuentra el modelo de la mano, es decir revelar en forma gráfica la ubicación de las falanges de los dedos y la posición de la mano en el entorno 3D; corresponde a una tarea que demanda buena cantidad de recursos del sistema y por lo tanto se le ha establecido un periodo máximo de ejecución de 1000msg y generalmente en ejecución se lleva a cabo en aproximadamente 100msg. Cabe aclarar que los periodos y plazos indicados en la tabla anterior se cumplen siempre y cuando el PC en el cual se ejecute el software sea de características similares o superiores a las anteriormente indicadas y que además en el PC no se encuentre ejecutándose ninguna aplicación que pueda afectar el rendimiento del sistema, ello debido a que el sistema operativo empleado (Windows XP) no corresponde a un sistema operativo de tiempo real.

4.4.1.2.2 Plazo de parada.

El plazo de parada en el sistema viene determinado de dos formas, ambas se presentan solo en modo de ejecución automático y corresponden a los siguientes: parada debido a evento del usuario y parada debido a finalización de la simulación.

La parada debido a evento del usuario se lleva a cabo en un plazo de aproximadamente menos de 100msg y se presenta cuando una vez iniciada la simulación el usuario oprime el botón Detener, momento en el cual el sistema deja de realizar las tarea de Paso de valores articulares y Graficado 3D, las tareas de Graficado 2D y Actualización gráfica 3D se siguen ejecutando normalmente.

La parada debido a finalización de la simulación posee un plazo de tiempo que depende del número de datos a simular y del periodo de simulación indicado por el usuario; se lleva

a cabo en el momento en el cual el sistema realiza el graficado del último dato a ser simulado.

4.4.2 El diseño de la arquitectura lógica.

La arquitectura lógica trata los requisitos que son independientes de las restricciones físicas impuestas por el entorno de ejecución, como por ejemplo la velocidad del procesador. Pertenecen a esta categoría los requisitos funcionales identificados anteriormente.

4.4.2.1 Descomposición de primer nivel.

El primer paso en la construcción de la arquitectura lógica es identificar las clases de objetos adecuados a partir de los cuales se va a construir el sistema. Los requisitos funcionales del sistema sugieren dos subsistemas, según muestra la
Figura 4.21.

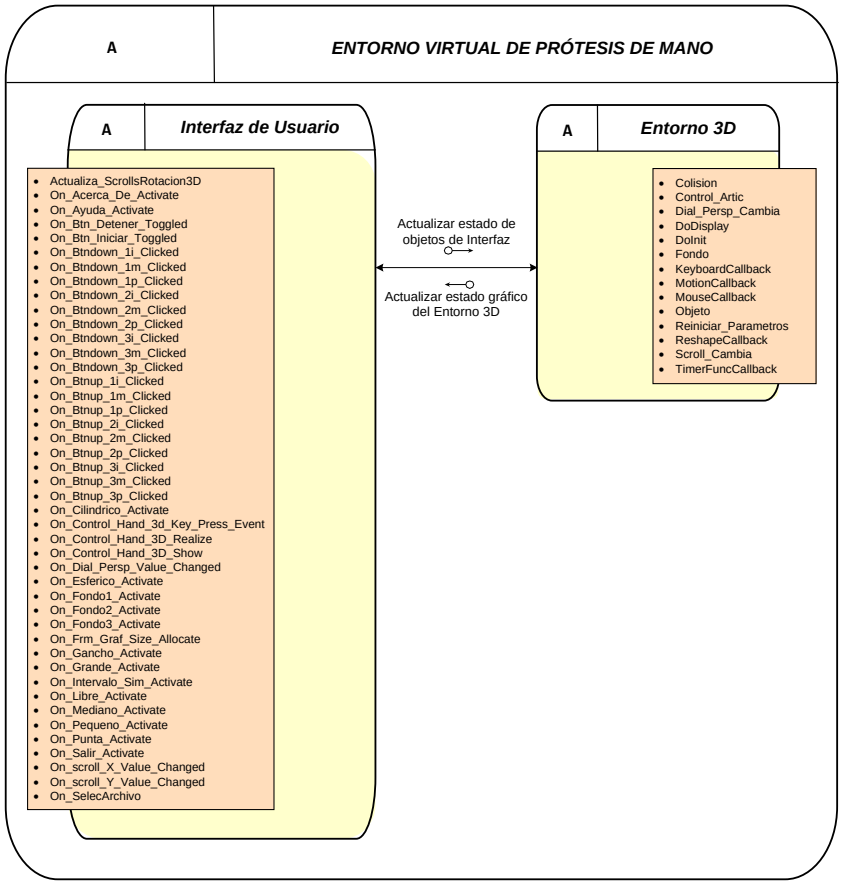


Figura 4.21 Descomposición jerárquica de primer nivel “Entorno Virtual de Prótesis de Mano”

Fuente: Elaboración Propia.

En el anterior diagrama se muestra el **Subsistema Interfaz de Usuario**, el cual es el responsable de la creación, despliegue y funcionamiento de la interfaz gráfica de usuario que corresponde a la parte de la aplicación que contiene los diversos elementos que permiten llevar a cabo acciones sobre el Subsistema Entorno 3D. El subsistema Interfaz de usuario actualiza constantemente el estado de sus diversos elementos con base al estado de las variables del Entorno 3D según los eventos del usuario. El **Subsistema Entorno 3D** se encarga de la creación y despliegue gráfico 3D del entorno virtual, cuya función es mostrar en forma gráfica tridimensional el modelo de prótesis de mano, su estado gráfico es controlado por los eventos indicados por el usuario en el Subsistema Interfaz de Usuario.

4.4.2.2 Descomposición jerárquica de los sub-módulos de la arquitectura lógica del sistema.

En su descomposición jerárquica el objeto Interfaz de Usuario contiene un total de 46 operaciones como lo muestra la Figura 4.22, las cuales son llevadas a cabo por dos módulos principales a saber Diseño_Interfaz (implementado en los archivos de código fuente Control_Hand_3D_Pkg.ads y .adb) y Callbacks_Interfaz (el cual es implementado en Control_Hand_3D_Pkg-Callbacks.ads y .adb), el primero se encarga de definir y crear la interfaz gráfica de usuario de la aplicación, el segundo se encarga de implementar los procedimientos y funciones que atienden los eventos que se llevan a cabo sobre los elementos presentes en la interfaz de usuario y sobre el Entorno_3D, éste objeto requiere de otros sub-objetos para poder desarrollar diversas acciones que juegan un rol importante en la aplicación desarrollada, en el diagrama se pueden observar los cuatro sub-objetos que se consideran relevantes (Create_List, Create_Plot_RealTime, Create_Time_Chooser y Create_File_Chooser) y que serán descritos más adelante en la sección 4.4.2 del presente documento. Cabe aclarar que el subsistema Interfaz de usuario emplea además de los objetos listados en el diagrama otros objetos los cuales se encuentran presentes en la librería GtkAda que debido a su gran numero y al estar bien documentados en la ayuda de la librería no fueron listados en el diagrama.

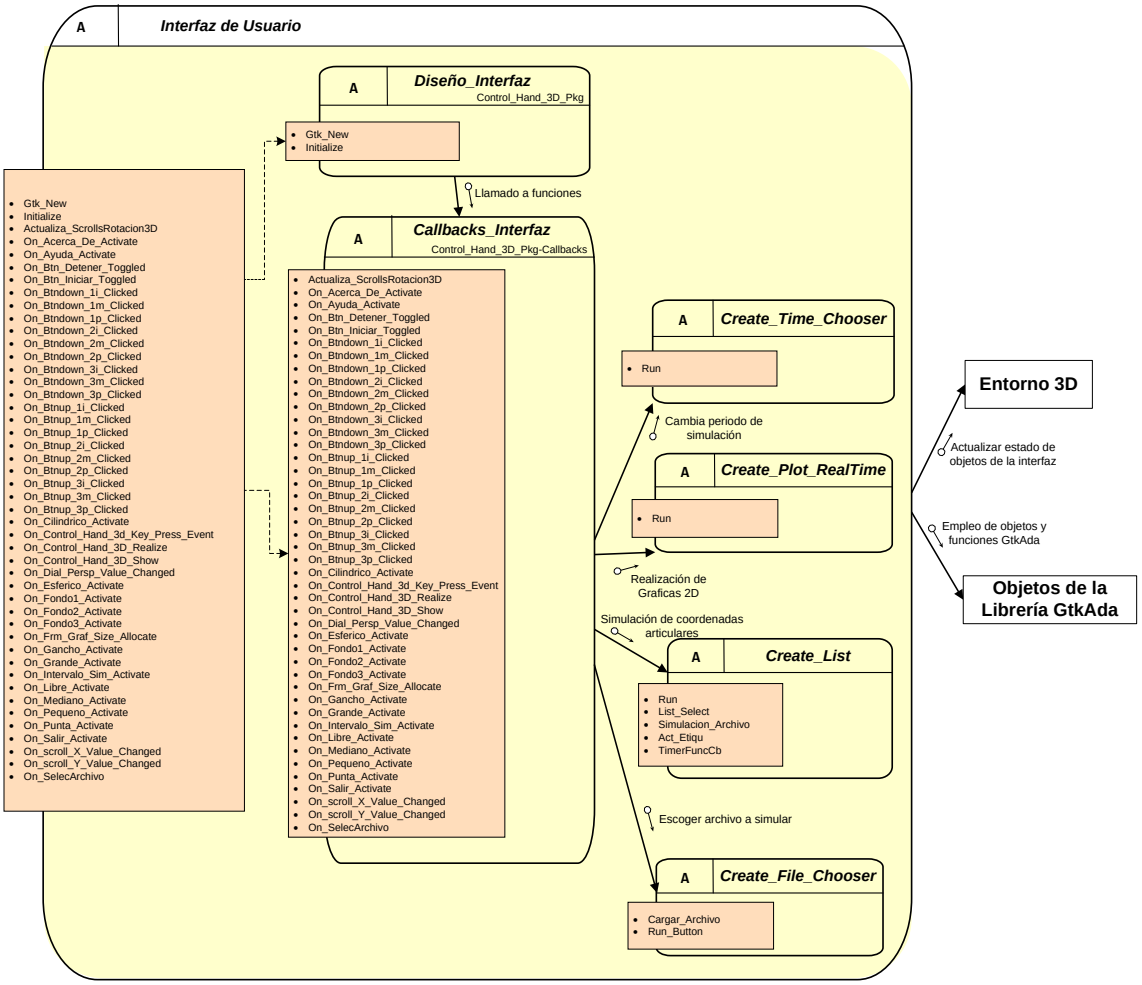


Figura 4.22 Descomposición jerárquica del objeto “Interfaz de Usuario”
Fuente: Elaboración Propia.

A continuación se describen brevemente las operaciones que implementa el subsistema Interfaz de Usuario las cuales son desarrolladas por los objetos Diseño Interfaz y Callbacks_Interfaz.

Gtk_New: Procedimiento que permite crear el objeto de tipo ventana llamado Control_Hand_3D en el cual se ubican los elementos que conforman la aplicación (Ejemplo: menús, pestañas, botones, gráficas, etc.).

Initialize: Procedimiento empleado para inicializar la Aplicación en GtkAda; es decir, para generar y colocar los elementos internos de la interfaz gráfica de usuario (GUI).

Actualiza_ScrollsRotacion3D: Procedimiento que permite pasar los ángulos de rotación en X, Y del entorno 3D desde OpenGL a la aplicación GtkAda.

On_Acerca_De_Activate: Muestra el cuadro de dialogo Acerca de.

On_Ayuda_Activate: Muestra la Ayuda de la Aplicación.

On_Btn_Detener_Toggled: Procedimiento que se encarga de Detener la simulación.

On_Btn_Iniciar_Toggled: Procedimiento que se encarga de activar el botón que Inicia la simulación, la cual se ejecuta inmediatamente únicamente si existen datos a ser simulados.

On_Btndown_1i_Clicked, On_Btndown_1m_Clicked, On_Btndown_1p_Clicked: Procedimientos que controla el botón para disminución del valor de la articulación asociada a la falange Proximal del dedo Índice, Medio y Pulgar respectivamente.

On_Btndown_2i_Clicked, On_Btndown_2m_Clicked, On_Btndown_2p_Clicked: Procedimientos que controla el botón para disminución del valor de la articulación asociada a la falange Medial del dedo Índice, Medio y Pulgar respectivamente.

On_Btndown_3i_Clicked, On_Btndown_3m_Clicked y On_Btndown_3p_Clicked: Procedimientos que controla el botón para disminución del valor de la articulación asociada a la falange Distal del dedo Índice, Medio y Pulgar respectivamente.

On_Btnup_1i_Clicked, On_Btnup_1m_Clicked, On_Btnup_1p_Clicked: Procedimientos que controla el botón para aumento del valor de la articulación asociada a la falange Proximal del dedo Índice, Medio y Pulgar respectivamente.

On_Btnup_2i_Clicked, On_Btnup_2m_Clicked, On_Btnup_2p_Clicked: Procedimiento que controla el botón para aumento del valor de la articulación asociada a la falange Medial del dedo Índice, Medio y Pulgar respectivamente.

On_Btnup_3i_Clicked, On_Btnup_3m_Clicked, On_Btnup_3p_Clicked: Procedimientos que controla el botón para aumento del valor de la articulación asociada a la falange Distal del dedo Índice, Medio y Pulgar respectivamente.

On_Cilindrico_Activate: Ordena al entorno virtual que muestre un objeto 3D con forma cilíndrica para que así se pueda simular con la mano un agarre tipo cilíndrico.

On_Control_Hand_3D_Key_Press_Event: Función que permite enviar al Entorno 3D el valor de la tecla oprimida desde la Aplicación. (Para controlar los movimientos de las falanges).

On_Control_Hand_3D_Realize: Mediante este procedimiento se hace la inicialización del Entorno 3D mediante directivas OpenGL.

On_Control_Hand_3D_Show: Procedimiento que se encarga de llamar a la función que realiza las Gráficas 2D para cada dedo (cada gráfica muestra tres trazados correspondientes a las variaciones angulares de las respectivas articulaciones).

On_Dial_Persp_Value_Changed: Permite controlar el Dial asociado a la Perspectiva del Entorno_3D (es decir, permite acercar o alejar la mano y los objetos 3D).

On_Esferico_Activate: Ordena al entorno virtual que muestre un objeto 3D con forma esférica para que así se pueda simular con la mano un agarre tipo esférico.

On_Fondo1_Activate, On_Fondo2_Activate, On_Fondo3_Activate: Cambia el color de fondo del entorno virtual al color indicado por la variable Fondo_Selec.

On_Frm_Graf_Size_Allocate: Permite Indicar que el tamaño de los frames que contienen las gráficas 2D de variaciones articulares ha cambiado.
On_Gancho_Activate: Ordena al entorno virtual que muestre un objeto 3D con forma de gancho para que así se pueda simular con la mano un agarre tipo gancho.

On_Intervalo_Sim_Activate: Procedimiento que se encarga de llamar al cuadro de dialogo en el cual se indica el intervalo de tiempo máx. entre cada línea de datos a simular.

On_Libre_Activate: Ordena al entorno virtual que no se muestre ningún objeto 3D cerca a la mano.

On_Grande_Activate, On_Mediano_Activate, On_Pequeno_Activate: Indica al entorno virtual a través del Menú Ver -> Dimensión del Objeto que muestre un objeto 3D de tamaño grande, mediano o pequeño respectivamente.

On_Punta_Activate: Ordena al entorno virtual que muestre un objeto 3D con forma de una pequeña esfera para que así se pueda simular con la mano un agarre tipo punta.

On_Salir_Activate: Procedimiento empleado para Cerrar o Salir de la aplicación.

On_scroll_X_Value_Changed, On_scroll_Y_Value_Changed: Permite controlar la barra de desplazamiento asociado a la rotación en el eje X y eje Y respectivamente para todo el conjunto de Objetos 3D del Entorno.

On_SelecArchivo: Permite obtener la ruta y realizar la lectura del Archivo que contiene los datos a ser simulados.

Ahora se procede a describir los objetos y las operaciones implementadas en los objetos que operan en conjunto con Callbacks_Interfaz los cuales son: Create_List, Create_Plot_RealTime, Create_Time_Chooser y Create_File_Chooser; cada objeto se descompuso jerárquicamente con el fin de poder ser descrito en forma más detallada, a excepción del objeto Create_File_Chooser el cual no posee sub objetos que deban ser descritos.

El objeto Create_List implementa cinco operaciones (Run, List_Select, Simulacion_Archivo, Act_Etiqu y TimerFuncCb) y a nivel interno posee dos tareas (Tarea y Tarea3D) cuyas operaciones son Ctrl_Simulación y Actualizar_Datos_3D respectivamente. La Figura 4.23 muestra el diagrama de descomposición jerárquica del objeto Create_List.

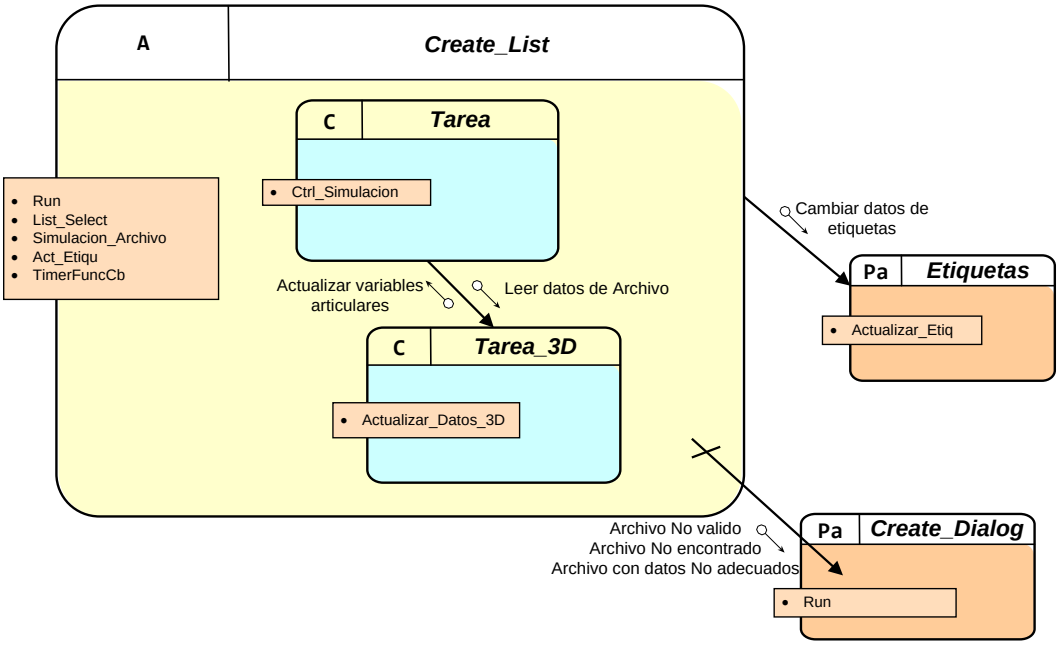


Figura 4.23 Descomposición jerárquica del objeto “Create_List”

Fuente: Elaboración Propia.

La función del objeto Create_List en la aplicación es crear un objeto tipo lista para que puedan ser desplegados los datos de los valores articulares del archivo que se desea simular. La Tabla 4.2 muestra que los datos son ubicados en filas, cada una debe contener nueve valores articulares expresados en grados uno para cada articulación cuyo orden de los datos en cada fila debe ser el siguiente:

Dato 1	Dato 2	Dato 3	Dato 4	Dato 5	Dato 6	Dato 7	Dato 8	Dato 9
Artic. Proximal dedo Medio	Artic. Medial dedo Medio	Artic. Distal dedo Medio	Artic. Proximal dedo Índice	Artic. Medial dedo Índice	Artic. Distal dedo Índice	Artic. Proximal dedo Pulgar	Artic. Medial dedo Pulgar	Artic. Distal dedo Medio

Tabla 4.2 Forma de ordenar los datos en el archivo a simular.
Fuente: Elaboración Propia.

Cada columna hace referencia a cada articulación de los dedos modelados y cada fila de datos corresponde a una intensión de movimiento cuya velocidad de movimiento de cada articulación en el modelo 3D está determinado por el cambio de los valores articulares entre cada línea de datos, el tiempo de simulación de una a otra línea de datos es por defecto de 100 milisegundos, sin embargo el usuario puede modificarlo en el menú de opciones de la aplicación. En la Figura 4.24 se puede apreciar la forma en la cual son desplegados los datos en la aplicación.

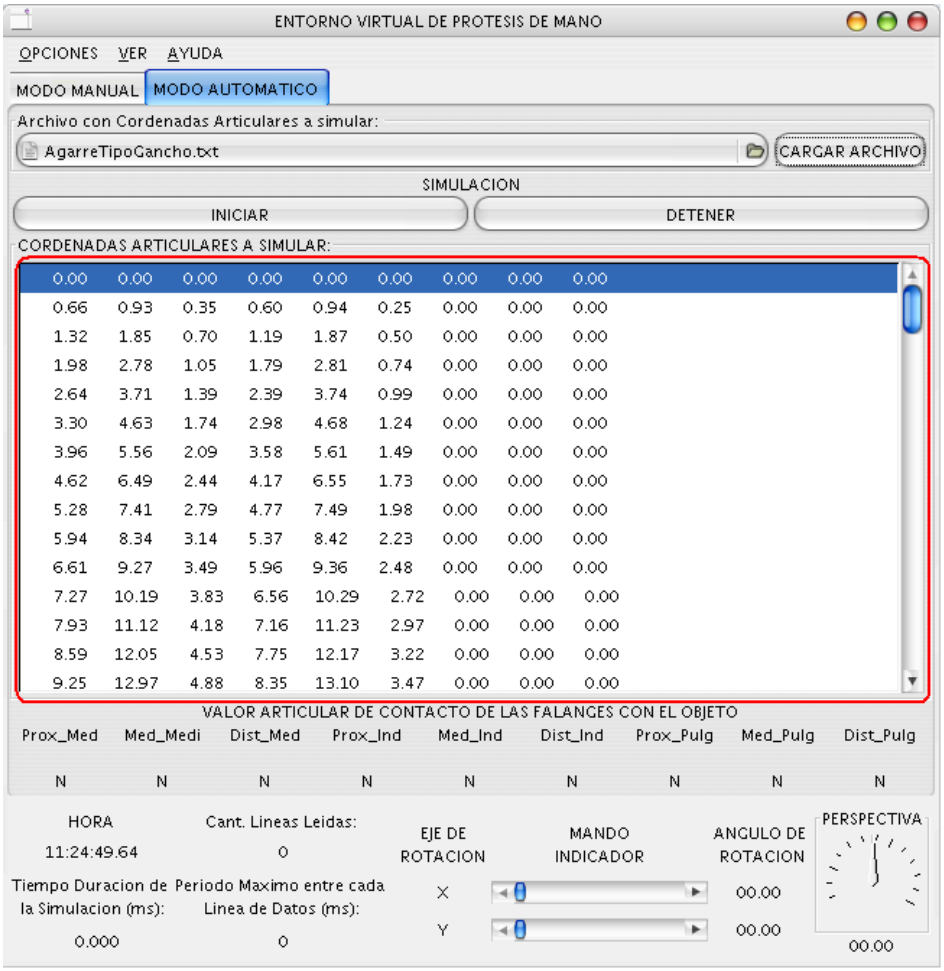


Figura 4.24 Forma en la cual se despliegan los datos en la aplicación
Fuente: Elaboración Propia.

A continuación se procede a describir cada una de las operaciones que implementa el objeto Create_List:

Run: Procedimiento que se encarga de crear y configurar un objeto de tipo Lista, en el cual son desplegados los datos del archivo a ser simulado. Cuando el usuario intenta cargar un archivo que: no existe, no es válido o no contiene datos adecuados para que sean simulados, se genera una excepción mediante la cual se llama al objeto pasivo Create_Dialog que se encarga a través de la operación Run de indicar al usuario en una pequeña ventana el mensaje respectivo al suceso ocurrido.

List_Select: Procedimiento que permite simular la posición de las falanges de los dedos de acuerdo con los valores articulares de los datos presentes en la línea seleccionada en la lista.

Simulacion_Archivo: Procedimiento encargado de llamar desde GtkAda al procedimiento TimerFunc de OpenGL con el fin de llevar a cabo la ejecución de la simulación. Dicho procedimiento es llamado únicamente si en el objeto Lista existe un archivo con datos validos a simular.

Act_Etiqu: Procedimiento que actualiza el valor de las etiquetas que indican: Número de líneas simuladas, Periodo máximo de simulación entre cada línea de datos y el Tiempo total que dura la simulación; la actualización se lleva a cabo gracias a la operación Actualizar_Etiq del objeto pasivo Etiquetas. Aplica solo si se está desarrollando alguna simulación, de lo contrario publica 0 en las etiquetas.

TimerFuncCb: Procedimiento que se encarga de llevar a cabo la simulación de los datos presentes en la Lista, este procedimiento establece como máximo periodo de tiempo de simulación entre cada línea de datos el valor indicado por el usuario (por defecto 100ms). Implementa en su interior dos tareas Tarea y Tarea3D las cuales trabajan en forma cíclica y sincronizada para llevar a cabo el paso de datos entre la Interfaz gráfica de usuario (desarrollada con GtkAda) y el Entorno 3D (desarrollado con funciones OpenGL) respectivamente. La
Figura 4.25 muestra el diagrama jerárquico del objeto Create_Plot_RealTime el cual implementa una operación (Run) y a nivel interno posee tres elementos dos cíclicos (Update y Free) y uno esporádico (Destroy); este objeto se vale de los objetos pasivos Reloj y Etiquetas para realizar operaciones de actualización de datos en la interfaz de usuario.

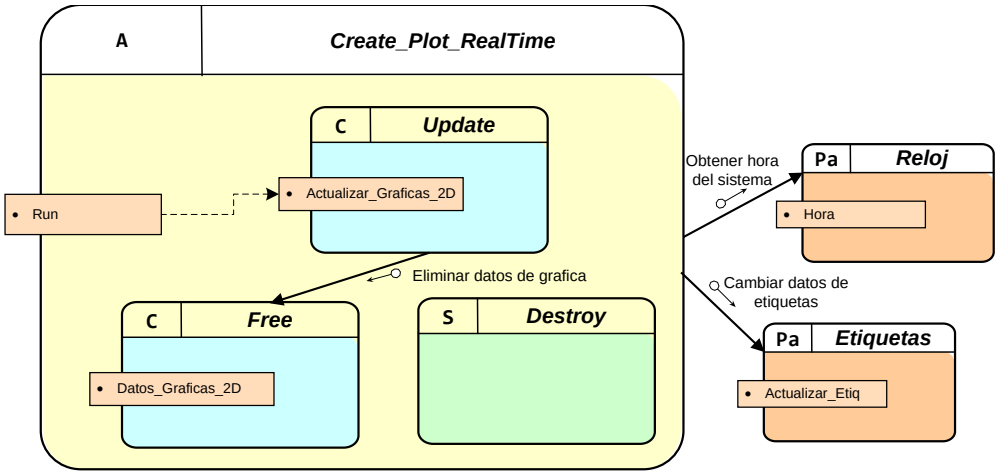


Figura 4.25 Descomposición jerárquica del objeto “Create_Plot_RealTime”
Fuente: Elaboración Propia.

El objeto Create_Plot_RealTime se encarga de crear las gráficas 2D en las cuales se trazan los valores angulares de las articulaciones de los dedos (grados) vs. tiempo (msg), en total se despliegan tres cuadros de gráficas en la aplicación cada uno para cada dedo modelado (pulgar, índice y corazón), cada gráfica contiene tres trazos correspondientes a las tres articulaciones de las falanges de los dedos, el color del trazo para la falange proximal (rojo), falange medial (azul) y falange distal (verde).

Mediante este paquete también se lleva a cabo la actualización de las etiquetas que despliegan la hora, el tiempo de duración de la simulación, la cantidad de líneas simuladas y el periodo de tiempo máximo entre cada línea de datos.

A continuación se procede a describir la operación que implementa el objeto Create_Plot_RealTime:

Run: Procedimiento empleado para crear y configurar las gráficas 2D en las cuales son desplegados los cambios angulares de las articulaciones de los dedos modelados, este procedimiento emplea los procedimientos internos **Updater** y **Free**, el primero se encarga de actualizar periódicamente el contenido del objeto que contiene la gráfica y el segundo permite eliminar los datos pasados o residuales que existan en los Arrays o Vectores que almacenan los valores de los trazos de las gráficas.

La Figura 4.26 muestra el diagrama jerárquico del objeto Create_Time_Chooser el cual implementa una operación (Run). El objeto Create_Time_Chooser corresponde a un paquete que se encarga de mostrar una ventana Dialogo en la cual se le indica al usuario que debe establecer el Intervalo de Tiempo máximo de simulación entre cada línea de datos; por defecto dicho tiempo es de 100ms.

Run: es una operación que crea el cuadro de dialogo en la cual va el cuadro de texto y el mensaje que se desea publicar al usuario y los botones para que el usuario realice la escogencia del periodo de tiempo máximo entre cada línea de datos, una vez el usuario realice la selección de dicho tiempo entonces llama al procedimiento **Destruir_Dialogo** que se encarga de aplicar la operación Eliminar_Objeto_Dialogo, para así cerrar el cuadro de dialogo y permitir al usuario seguir trabajando en la aplicación y poder ejecutar la simulación con el periodo de tiempo seleccionado.

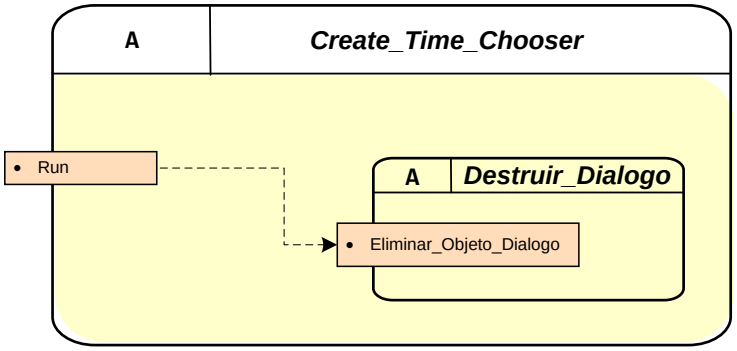


Figura 4.26 Descomposición jerárquica del objeto “Create_Time_Chooser”
Fuente: Elaboración Propia.

El objeto **Entorno_3D** es importante en la aplicación puesto que corresponde al paquete que mediante el empleo de funciones o procedimientos OpenGL del motor Globe_3D logra que en la aplicación se pueda representar en forma gráfica tridimensional el modelo de la prótesis de mano robótica de 9GDL y unos objetos, de tal forma que la mano 3D pueda desarrollar tareas de agarre sobre alguno de los objetos que se encuentre presente en el entorno tridimensional. En la Figura 4.27 se puede observar la descomposición jerárquica de dicho objeto, el cual realiza la actualización del estado gráfico del entorno 3D según las ordenes indicadas por el usuario en el objeto **Interfaz de Usuario.** En cuanto al despliegue grafico, éste se lleva a cabo gracias al llamado de funciones OpenGL que incluye el motor **Globe_3D**.

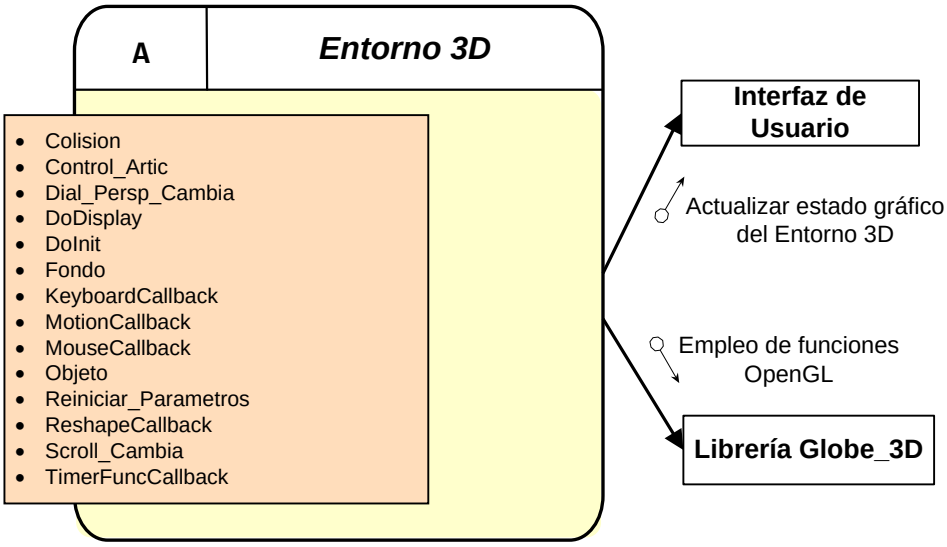


Figura 4.27 Descomposición jerárquica del objeto "Entorno 3D"
Fuente: Elaboración Propia.

Las operaciones que implementa el objeto Entorno_3D se describen a continuación:

Colision: Procedimiento que realiza los cálculos para la colisión de las falanges de los dedos con el objeto que se encuentre presente en el entorno. Es decir, en este procedimiento se encuentra definido el algoritmo para la detección de colisiones.

Control_Artic: Procedimiento empleado para enviar el valor de los Diales que controlan las articulaciones al Entorno 3D.

Dial_Persp_Cambia: Procedimiento empleado para enviar el valor del dial que controla la perspectiva del modelo virtual al Entorno 3D.

DoDisplay: Procedimiento en el cual se llevan a cabo todas las operaciones de graficado 3D, por ejemplo se crea la palma y los dedos que componen la mano, se crean los objetos y demás elementos mediante los cuales se hace la detección de colisiones.

DoInit: Procedimiento empleado para la inicialización del ambiente y definición de las coordenadas del sistema (funciones que se ejecutan una sola vez).

Fondo: Procedimiento que controla el Color de Fondo del Entorno_3D.

KeyboardCallback: Procedimiento empleado para capturar y manejar eventos del teclado. En él se definen las acciones que se deben llevar a cabo en la aplicación de acuerdo la tecla que se haya oprimido.

MotionCallback: Procedimiento que se ejecuta en conjunto con Mousecallback y se emplea para que los ángulos de rotación del Entorno 3D queden Entre 0º Y 360º. Además hace el llamado a las funciones que pasan los ángulos de rotación del Entorno 3D a la interfaz gráfica de usuario GtkAda.

MouseCallback: Procedimiento empleado para capturar y manejar eventos del mouse o ratón. En este procedimiento se definen la acción de rotación de todo el modelo 3D cuando el botón izquierdo del mouse se mantiene oprimido.

Objeto: Procedimiento que controla el objeto que aparece en el Entorno_3D.

Reiniciar_Parametros: Procedimiento que restablece valores de algunos parámetros tanto para la Interfaz de usuario como para el Entorno_3D.

ReshapeCallback: Procedimiento llamado cada vez que se re-escale la ventana. Permite hacer la actualización del tamaño de los elementos 3D presentes en el entorno según el tamaño actual de la ventana (el primer parámetro será el ancho y el segundo el alto, después del re-escalado).

Scroll_Cambia: Procedimiento empleado para pasar los valores de los ángulos de rotación desde la Aplicación GtkAda al Entorno 3D.

TimerFuncCallback: Procedimiento muy importante, puesto que se encarga de actualizar periódicamente el estado grafico del Entorno_3D y los objetos ante la recepción de cualquier evento.

4.5 DESARROLLO DEL ENTORNO GRÁFICO 3D.

OpenGL se define estrechamente como "una interfaz software para gráficos por hardware". En esencia, es una librería de gráficos 3D y modelado portátil muy rápido. Usando OpenGL se pueden crear gráficos 3D elegantes y bellos con una calidad visual cercana a la de un trazador por rayos, con la gran ventaja de ser más rápida que un trazador. Emplea algoritmos desarrollados y optimizados por Silicon Graphics, Inc. (SGI). OpenGL se propone para el empleo de hardware informático diseñado y optimizado para la visualización y manipulación de gráficos 3D. Las implementaciones genéricas de OpenGL, que sólo constan de código, también son posibles, y en esta categoría entran las implementaciones de Windows por renderización software (55).

Las empresas de hardware gráfico para PCs están añadiendo aceleración 3D a sus productos, influenciados principalmente por las exigencias del mercado de juegos 3D, hecho este que tiene un paralelismo cercano a la evolución de las aceleradoras gráficas 2D, basadas en Windows, que optimizan operaciones como el trazado de líneas y el relleno y manipulación de mapas de bits.

OpenGL es un lenguaje procedimental más que un lenguaje descriptivo. En lugar de diseñar la escena y cómo debe aparecer, el programador describe los pasos necesarios para conseguir una cierta apariencia o efecto. Estos "pasos" implican llamadas a una API altamente portátil que incluye aproximadamente 120 comandos y funciones. Estos se emplean para dibujar primitivas gráficas como puntos, líneas y polígonos en tres dimensiones. Además, OpenGL soporta iluminación y sombreado, mapeado con texturas, animación y otros efectos especiales (56).

Para entender mejor cómo funciona o cómo trabaja OpenGL, resulta necesario observar cómo se estructura su funcionamiento. OpenGL es una librería gráfica escrita originalmente en lenguaje de programación C que permite la manipulación de gráficos 3D a todos los niveles. Esta librería se ejecuta a la par con la aplicación independientemente de la capacidad gráfica de la máquina que se use. Esto significa que la ejecución se dará por software a no ser que se cuente con hardware gráfico específico en la máquina, sin embargo si se cuenta con una tarjeta aceleradora de gráficos por supuesto se gozará de una ejecución muchísimo más rápida en tiempo real. Así esta librería puede usarse bajo todo tipo de sistemas operativos e incluso usando una gran variedad de lenguajes de programación (57).

Los pasos a seguir en el renderizado de una escena a pantalla es la siguiente.

1. Construir primitivas geométricas (dada como descripción matemática: puntos, líneas, polígonos, imágenes y mapas de bits).
2. Ubicar los objetos en el espacio 3D y definir el punto de vista.
3. Calcular los colores de todos los objetos.
4. Convertir la información matemática y de color de objetos a píxeles en la pantalla (Rasterización) (58).

El funcionamiento de OpenGL está definido por el pipeline de procesamiento geométrico, esto es que la geometría que se desea dibujar será la entrada del pipeline gráfico (arquitectura gráfica) y como salida se obtiene una imagen en pantalla. El proceso que ocurre con todos los puntos, vectores y polígonos desde que entran hasta que salen se presenta en la Figura 4.28.

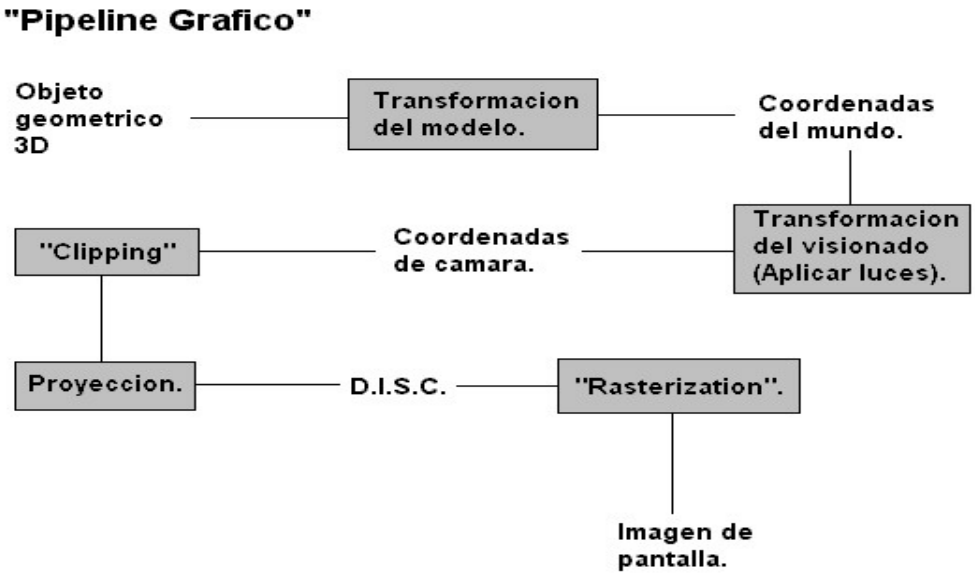


Figura 4.28. Pipeline de OpenGL.
Fuente: (57).

4.5.1 Análisis del los módulos del pipeline de OpenGL.

Objeto geométrico: El mundo se compone de puntos, líneas, polígonos, etcétera; en OpenGL consideradas primitivas. Inicialmente estos objetos tienen unos atributos que se definen pero pueden ser movibles o fijos, deformables o rígidos, y, por tanto, se debe ser capaz de trasladarlos, rotarlos y escalarlos antes de dibujarlos en pantalla.

Transformación del modelo: Módulo encargado de trasladar, rotar, escalar cualquier objeto para que sea dibujado en la pantalla. OpenGL realiza dichas funciones multiplicando a la geometría (vértices) por varias matrices, cada una de las cuales implementa un proceso (rotar, trasladar...) Las funciones de OpenGL se usan para "crear" estas matrices y emplearlas con la geometría.

Coordenadas del mundo: Posiciones de todos los objetos en el mundo virtual. Son posiciones referidas a un sistema de coordenadas que se define única y exclusivamente para el mundo que se está creando.

Transformación del visionado: Indica cómo se ven esos objetos, ya posicionados correctamente en el mundo, desde la cámara virtual. Por lo tanto, los objetos se iluminan para que sean visibles y se obtienen sus posiciones tal y como se ven desde la cámara.

Coordenadas de cámara: tras aplicar luces ya se conoce cuáles son las coordenadas de todos los objetos respecto a la cámara, es decir, como el usuario observa en forma real dichos objetos.

Clipping: Consiste en recortar (ocultar) todo aquello que "está" pero "no se ve" desde la cámara. Es decir, todo aquello que la cámara no puede mostrar porque no entra en su ángulo de visión se elimina, a dicho proceso se denomina Clipping.

Proyección: Conversión de coordenadas 3D del mundo a coordenadas 2D del plano de proyección para el usuario.

D.I.S.C. (*Device Independent Screen Coordinates*): tras proyectar se tiene lo que se llaman "coordenadas de pantalla independientes del dispositivo". Las coordenadas que se tienen calculadas en ese momento todavía no se han asociado a ningún tipo de pantalla o monitor. En este punto del pipeline OpenGL no sabe si el monitor es de 15 pulgadas y de resolución 800 x 600 o si es de 19 pulgadas y de resolución 1024 x 1480. De hecho esto no lo controla el usuario. Se trata de asociar la imagen recortada 2D que se encuentra en el frame buffer con los píxeles de la pantalla. Según la resolución de pantalla sea mayor o menor, un punto del mundo para el usuario ocupara más o menos unos cuantos píxeles.

Rasterización: Proceso que se conoce también como "*scan conversion*". Finalmente se asocian todos los puntos a píxeles en pantalla.

Imagen de pantalla: Aquí termina el proceso. Ya se tiene la imagen de lo que la cámara muestra en el monitor (57).

4.5.2 Tratamiento matemático llevado a cabo por OpenGL.

Uno de los aspectos más importantes en la elección de OpenGL para crear el modelo 3D de la mano es su congruencia matemática asociada a la robótica dado que esta enormemente ligada al algebra matricial (56). Así Debido al carácter espacial de los sistemas robóticos, es importante realizar una agrupación de todos los parámetros existentes: localización y orientación como en la siguiente matriz (más adelante se explicaran este tipo de matrices).

$$\begin{bmatrix} Cos(\alpha) & -Sen(\alpha) & 0 & L \cdot Cos(\alpha) \\ Sen(\alpha) & Cos(\alpha) & 0 & L \cdot Sen(\alpha) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \{1\}$$

La matriz {1} podría corresponder con la de transformación individual de una sola articulación en la cual la rotación se realiza sobre un único eje cartesiano según el método de Denavit.-Hartenberg (59), (60), (61). Si el sistema tuviese tres articulaciones más, la posición y orientación del manipulador vendría dada por la expresión {2}, donde el superíndice indica a qué falange va referido y el subíndice sobre qué falange se realiza la operación, ${}^{0}T_{3}$ sería la trasformación de la falange 3 (56).

$$ {}^{0}T_{3}={}^{0}T_{1}*{}^{1}T_{2}*{}^{2}T_{3} \qquad \{2\} $$

Como ya se puede intuir, la mayor ventaja de la estructura matricial es la sencillez y claridad de las expresiones resultantes. Sin esta agrupación, en el caso anterior, se tendría que recurrir a 12 fórmulas (hay que tener en cuenta que la última columna es un añadido que no aporta información, por lo que no es necesario calcularlo), siendo éstas, en muchos casos, de una longitud considerable (56). Es así que OpenGL provee al usuario de funciones cuyo funcionamiento a nivel matemático es muy congruente con el de la robótica. OpenGL al igual que en robótica trabaja con coordenadas y matrices homogéneas.

La representación mediante coordenadas homogéneas de la localización de sólidos en un espacio n-dimensional se realiza a través de un espacio (n+1) dimensional. Es decir, un espacio n-dimensional se encuentra expresado en coordenadas homogéneas por (n+1) dimensiones, de tal forma que un vector P(x,y,z) es representado por P(wx,wy,wz,w), donde w tiene un valor arbitrario y representa un factor de escala. De forma general, un vector P= ai + bj + ck, donde i,j,k son los vectores unitarios de los ejes OX, OY, OZ del sistema de referencia OXYZ, se representa en coordenadas homogéneas mediante el vector columna (62):

$$ P=\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}=\begin{bmatrix} aw \\ bw \\ cw \\ w \end{bmatrix}=\begin{bmatrix} a \\ b \\ c \\ 1 \end{bmatrix} $$

Por ejemplo, el vector 2i + 3j + 4k se puede representar en coordenadas homogéneas como $[2,3,4,1]^{T}$, o como $[4,6,8,2]^{T}$, etc. Los vectores nulo se representan como $[0,0,0,n]^{T}$, donde n es no-nulo (62).

A partir de la definición de coordenadas homogéneas surge inmediatamente el concepto de matriz de transformación homogénea. Se define como **matriz de transformación homogénea T** a una matriz de dimensión 4x4 que representa la transformación de un vector de coordenadas homogéneas de un sistema de coordenadas a otro (62).

$$T=\begin{bmatrix} R_{3x3} & P_{3x1} \\ f_{1x3} & w_{1x1} \end{bmatrix}=\begin{bmatrix} Rotacion & Traslación \\ Perspectiva & Escalado \end{bmatrix}$$

Se puede considerar que una matriz homogénea se haya compuesto por 4 sub-matrices de distinto tamaño: una sub-matriz R_{3x3} que corresponde a una matriz de rotación; una sub-matriz P_{3x1} que corresponde al vector de translación; una sub-matriz F_{1x3} que representa una transformación de perspectiva, y una sub-matriz W_{1x1} que representa un escalado global (62). En robótica generalmente solo se interesará conocer el valor de las matrices de rotación y traslación, considerándose las componentes de la matriz de perspectiva nulas y la de escala como la unidad. Sin embargo OpenGL si utiliza las matrices de perspectiva y escalado además de las de traslación y rotación para las diferentes figuras 3D. A continuación se muestran algunas de las funciones más importantes para las transformaciones geométricas.

Traslación: La llamada a **Gl.translate(x,y,z)** genera T, donde:

$$\mathbf{T=}\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Los parámetros x,y,z son las nuevas coordenadas donde se trasladará la primitiva (63).

Escalado: La llamada a **Gl.scale(x,y,z)** genera S, donde:

$$\mathbf{S=}\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Los parámetros x,y,z son los factores que afectarán a cada componente en x,y,z respectivamente (63).

Rotación: La llamada a **Gl.rotate(a,x,y,z)** genera una matriz que rotará un ángulo "(α)" sobre cualquiera de los ejes x,y,z. Si x,y,z son iguales a cero la matriz resultante será la identidad. A menudo se rotará solo en uno de los ejes coordenados, las matrices correspondientes para la llamada en cada eje x,y,z respectivamente son como sigue (63):

Gl.rotate (α, 1.0,0.0,0.0) :
$$\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -sen(\alpha) & 0 \\ 0 & sen(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Gl.rotate (α, 0.0,1.0,0.0) :
$$\begin{bmatrix} \cos(\alpha) & 0 & sen(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -sen(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Gl.rotate (α, 0.0,0.0,1.0) :
$$\begin{bmatrix} \cos(\alpha) & sen(\alpha) & 0 & 0 \\ -sen(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4.5.3 Usando sentencias OpenGL de Globe_3D en el código del proyecto de GPS.

El código de las funciones del motor Globe_3D utiliza los mismos parámetros de cada función OpenGL, aunque algunas funciones siguen la convención de tipo punto comúnmente empleada en Ada para indicar que una función, procedimiento u objeto hace parte de un paquete o librería, cabe aclarar que los nombres son muy similares a las funciones originales OpenGL pero hay ciertas diferencias que hay que tener en cuenta. En la Figura 4.29 se muestran algunos segmentos de código de los dos archivos (archivo de especificaciones Entorno3D_Procs.ads y cuerpo de programa Entorno3D_Procs.adb) en lenguaje Ada del código de las funciones OpenGL empleadas.

```
--- Archivo de especificaciones Entorno3D_Procs.ads ---

--  Las siguientes líneas de código son las instrucciones en
--  Ada para importar las librerías o paquetes necesarios
--  para la aplicación.
with Gl; use Gl;
with Glut; use Glut;
with Glut.devices; use Glut.devices;
with Glib; use Glib;
with Ada; with Ada.Text_IO;

-- Se inicia el paquete Entorno3D_Procs
package Entorno3D_Procs is

  --  Declaración e inicialización de Variables y Tipos de
  --  variables para el Entorno_3D.
  --  Por ejemplo:
Pi: constant gl.double := 3.14159265;

-- Articulacion de los dedos de la mano
Articulac_1I, Articulac_1M  :=0.0;
Articulac_1P : gl.Double :=0.0;
-- 1P..3P articulaciones del dedo pulgar

-- Id_Dedo: Permite identificar a cada dedo.
   -- DedoMedio=>D_Medio, DedoIndice=>D_Indice,
   -- DedoPulgar=>D_Pulgar
   type Id_Dedo is (D_Medio, D_Indice, D_Pulgar);

   -- Callbacks para las Funciones OpenGL
   procedure DoInit;
   procedure DoDisplay;
   procedure KeyboardCallback (key : Key_type; x : Integer; y : Integer);
   procedure ReshapeCallback (w : Integer; h : Integer);
   procedure MouseCallback( button, state, x, y: Integer );
   procedure MotionCallback( Xx, Yy: Integer );
   procedure TimerFuncCallback( Intervalo : in Integer);

end Entorno3D_Procs;

---  Archivo de cuerpo de programa Entorno3D_Procs.adb  ---

--  Se incluyen las librerías o paquetes necesarios para la aplicación.
-- with …; use …;

-- Inicia el cuerpo del paquete Entorno3D_Procs
package body  Entorno3D_Procs is

-- Se declaran e inicializan variables locales
-- Se crean los procedimientos y funciones
  procedure Objeto (IdObj: in IdObjeto; Dimension : in
  gl.double:= 0.0 )is
   begin
     -- Sentencias a ejecutar del procedimiento Objeto.
   end Objeto;
```

```
-- Ejemplo del código completo de DoInit
   procedure DoInit is
   begin
     gl.Light (GL.LIGHT0, GL.AMBIENT, (0.0, 0.7, 1.0, 1.0));
     gl.Light (GL.LIGHT0, GL.DIFFUSE, (1.0, 1.0, 1.0, 1.0));
     gl.Light (GL.LIGHT0, GL.SPECULAR,(1.0,1.0,1.0, 1.0));
     gl.Light (GL.LIGHT0, GL.POSITION, (1.0, 1.0, 1.0, 0.0));
          Enable (GL.LIGHTING);
          Enable (GL.LIGHT0);
          Enable (GL.DEPTH_TEST);
          DepthFunc (GL.LESS);
   end DoInit;

begin
--  Inicia la funcion Init de Glut y negocia una sesión con el sistema de ventanas, la
--  función en C es glutInit (&argc, argv); nótese que en Globe_3D hay un punto de por
--  medio Glut.init;
Glut.Init;

--  Inicializa una ventana con dos parámetros enteros, el ancho y el alto,
--  la función original en el manual de referencia es:
--  glutInitWindowSize (ANCHO, ALTO);
Glut.InitWindowSize (500, 500);

--  fija el modo de inicio de la pantalla, para este caso la instrucción original en C es:
--glutInitDisplayMode(GLUT_RGB||GLUT_DEPTH ||GLUT_SINGLE);
 Glut.InitDisplayMode(GLUT.RGB or GLUT.DEPTH or GLUT.SINGLE);
```

Figura 4.29 Segmentos de código en lenguaje Ada de las funciones OpenGL empleadas.
Fuente: Elaboración Propia.

4.5.4 Técnicas básicas usadas en Detección de Colisión

En aplicaciones interactivas en las que es necesario conocer la colisión entre objetos se necesita reducir el número de tests de colisión entre pares de objetos. Para ello son necesarias estructuras de datos especiales y heurísticas que reduzcan estos pares de tests en el peor de los casos. Además de lo anterior también se trata de reducir el número de pares de características a considerar en el test de colisión detallado entre un par de objetos.

4.5.4.1 Volúmenes Envolventes

Comprobar la colisión entre dos objetos es muy costoso a nivel computacional, sobre todo cuando están formados por miles de polígonos. Para minimizar este coste, suele comprobarse previamente la colisión entre sus volúmenes envolventes, de manera que sólo si colisionan se realiza un test de colisión detallado entre objetos.
Un volumen envolvente está formado por un volumen simple que contiene uno o más objetos de naturaleza compleja. La idea consiste en utilizar un test de colisión entre volúmenes envolventes, menos costoso que el propio test de colisión entre objetos. Esta

técnica establece que el número de intersecciones entre volúmenes envolventes (con una relación de aspecto y factor de escala constantes) es casi el mismo que el número de intersecciones entre objetos.

Cuando los objetos se encuentran en colisión, es cierto que se añade un pequeño coste a la detección de colisión, pues previamente se ha realizado un test de colisión entre volúmenes envolventes. En realidad se produce una ganancia, pues la mayoría de las veces no se produce colisión entre los volúmenes envolventes, con el consiguiente ahorro de tests detallados entre objetos.

Suele llamarse también test de colisión estático o test de la segunda fase de la detección de colisión, es decir, fase ancha (narrow phase). Para que sea útil, un volumen envolvente debe cumplir las siguientes propiedades:

- Debe ajustarse al objeto tanto como sea posible.
- El cálculo de colisión entre volúmenes envolventes debe ser poco costoso en comparación con el test de colisión entre objetos.
- El cálculo del volumen envolvente debe ser poco costoso, sobre todo si éste debe recalcularse con cierta regularidad.
- El volumen envolvente debe poder rotarse y transformarse fácilmente.
- Debe representarse utilizando poca cantidad de memoria. Los volúmenes envolventes más representativos son las *esferas envolventes*, las *cajas envolventes alineadas con los ejes,* las *cajas envolventes orientadas* y los *k-DOPs*, utilizándose normalmente las siglas en inglés de estos términos. En la Figura 4.30 se pueden ver estos volúmenes envolventes y cómo se ajustan unos mejor que otros a un objeto determinado.

A continuación se describe cada tipo de volumen envolvente.

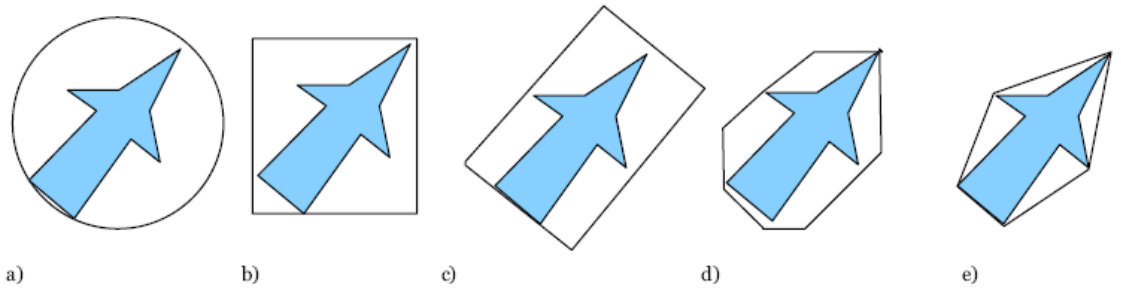


Figura 4.30 Tipos de volúmenes envolventes a) Esfera. b) Caja envolvente alineada con los ejes. c) Caja envolvente orientada. d) *k*-dop. e) Envolvente convexa.

Esfera envolvente: Se le suele llamar BS (Bounding sphere). Una esfera envolvente se beneficia de un test de colisión poco costoso, además tiene la ventaja de ser un volumen invariante ante rotaciones y de necesitar poco espacio de almacenamiento. El

inconveniente de este tipo de volumen envolvente consiste en que no se ajusta a determinados objetos, siendo difícil obtener la esfera que mejor ajuste a un objeto dado.

Caja envolvente alineada con los ejes: Normalmente conocida como AABB (Axis aligned bounding box), es una caja rectangular cuyas caras están orientadas de manera que sus normales son paralelas a los ejes coordenados. La ventaja de este tipo de volumen es que el test de colisión es muy rápido, consistiendo en la comparación directa de los valores de coordenadas. Sin embargo, no se produce un ajuste apropiado para muchos objetos. Además, actualizar este volumen envolvente suele ser más costoso que otros cuando los objetos rotan.

Caja envolvente orientada: También conocida como OBB (Oriented Bounding Box), este tipo de volumen envolvente es representado por una caja rectangular con una orientación arbitraria, aunque suelen utilizarse determinadas direcciones prefijadas. Al contrario que las AABB, este tipo de volumen envolvente tiene un test de colisión bastante costoso, basado en el Teorema del eje separador cuya utilización para detección de colisiones fue sugerida por Larcombe. Sin embargo, su uso viene justificado por su mejor ajuste a los objetos y su rápida actualización en las rotaciones. Además es necesario guardar mucha más información para representar este volumen, normalmente bajo la forma del centro de la caja, tres longitudes y una matriz de rotación.
Politopo de orientación discreta: Es un volumen envolvente basado en la intersección de *"slabs"*. Un slab es la región infinita del espacio delimitada por dos planos paralelos. Este volumen conocido como *k*-DOP (Discrete Orientation Polytope) es un politopo convexo definido por slabs cuyas normales pertenecen a un conjunto de direcciones predefinidas y comunes para todos los *k*-DOPs. Las componentes de los vectores normales están restringidas al conjunto de valores {-1,0,1}, y éstos no están normalizados (Figura 4.31).

Un AABB es un 6-DOP, pues está formado por 3 slabs. Evidentemente, mientras mayor sea el número de slabs de un *k*-DOP mejor se ajusta éste a los objetos, siendo necesario comprobar *k*/2 intervalos para determinar la colisión de volúmenes envolventes. La actualización de un *k*-DOP es algo más costosa que un AABB debido a su mayor número de slabs, pero proporcional a *k*. El almacenamiento de un *k*-DOP depende también del número de slabs pero es poco costoso debido sobre todo a que estos slabs están restringidos a ciertas configuraciones. Este volumen tiene la ventaja adicional de que siempre se puede modificar el tipo de ajuste a un objeto cambiando el número de slabs a utilizar. Existen muchos otros tipos de volúmenes envolventes que los aquí expuestos, como conos, cilindros, elipsoides etc.

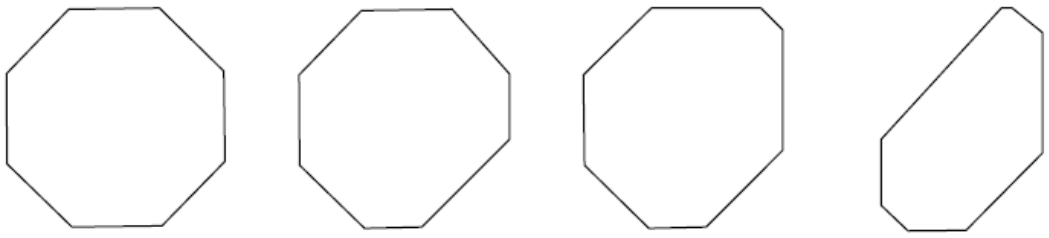


Figura 4.31 Ejemplos de k-dops.

Los distintos slabs son ajustados para adaptarse a un objeto, con lo que cambia la forma del volumen envolvente. Los planos que forman un slab siempre deben tener el mismo ángulo con la vertical que la configuración original.

Jerarquías de Volúmenes Envolventes: Utilizar Volúmenes Envolventes, puede disminuir el tiempo de cálculo de colisión entre objetos. Sin embargo el número de parejas de objetos sobre el que realizar este test no varía, por lo que, el tiempo de ejecución es el mismo aunque reducido por una constante. Uniendo los volúmenes envolventes a una jerarquía se puede reducir esta complejidad a logarítmica, construyendo para ello una jerarquía de volúmenes envolventes.

Para *n* objetos se puede construir una jerarquía de volúmenes envolventes en forma de árbol (Figura 4.32) de la siguiente manera: En primer lugar se obtienen los volúmenes envolventes de los objetos individuales y los colocamos como nodos hoja del árbol que representa dicha jerarquía. Estos nodos son agrupados utilizando un nuevo volumen envolvente de manera recursiva hasta obtener un volumen envolvente que agrupe todos los objetos (representado por la raíz del árbol). Con esta jerarquía comprobaríamos la colisión entre hijos de un nodo sólo si se produce colisión entre los nodos padre.
En una jerarquía de volúmenes envolventes no es necesario que un volumen padre englobe los volúmenes hijos, sino que simplemente debe englobar los dos objetos que representan los volúmenes envolventes. Además dos volúmenes envolventes del mismo nivel pueden tener partes comunes (intersecciones).

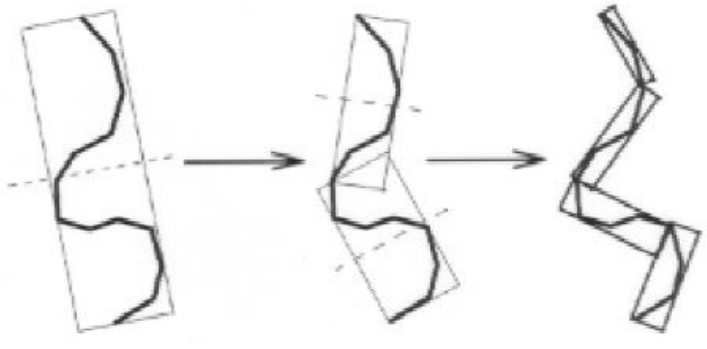


Figura 4.32 Jerarquía de Volúmenes Envolventes.

Una jerarquía de volúmenes envolventes puede utilizarse para organizar los objetos de una escena o bien pueden utilizarse jerarquías de volúmenes envolventes individuales para cada uno de los objetos. Suelen usarse para objetos poliédricos no convexos, de manera que se dividen en componentes convexas. Se debe tener en cuenta que esta partición en componentes convexas no suele ser trivial. Sería deseable que una jerarquía de volúmenes envolventes tuviera una serie de propiedades, por ejemplo:

- La geometría contenida en los nodos de cualquier sub-árbol debería estar próxima entre sí.
- Cada nodo de la jerarquía debería tener el mínimo volumen posible.
- Debemos considerar en primer lugar los nodos cercanos a la raíz de la jerarquía, pues eliminar uno de estos nodos es siempre mejor que hacerlo a mayor profundidad del árbol.
- El solapamiento entre nodos del mismo nivel debería ser mínimo.
- La jerarquía debería ser equilibrada en cuanto a su estructura y su contenido.
- En aplicaciones de tiempo real, la determinación de colisión utilizando la jerarquía no debe ser mucho peor que el caso medio. Además la jerarquía debe generarse de forma automática, sin la intervención de un usuario.
- Algunas de estas descomposiciones mediante jerarquías de volúmenes envolventes incluyen entre otros: *AABB-Trees*, *OBB-Trees* , *Box-Trees Sphere-Trees* , *k-DOP trees* , *Shell-Trees* , *R-Trees* y *QuOSPO-Trees* .

4.5.4.2 Volúmenes Envolventes a distinto nivel de detalle.

Podemos utilizar distintos volúmenes envolventes, cada uno con mayor nivel de ajuste o de detalle que el anterior y formar una sucesión de volúmenes envolventes (Figura 4.33). La proximidad entre objetos se obtiene de la misma forma para todos los niveles de este conjunto. Cuando no se puede determinar de forma exacta si dos objetos colisionan, se pasa al siguiente nivel de detalle.

Esta técnica implica un mayor coste de almacenamiento debido a la multiplicidad de volúmenes envolventes y un mayor número de tests cuando se produce colisión entre objetos. Además la actualización de un mayor número de volúmenes es mucho más costosa. Sin embargo, esta variabilidad en el ajuste de los volúmenes envolventes permite realizar tests de colisión entre volúmenes envolventes en menor número cuanto más ajustado al objeto sea el volumen (y más costoso), y mayor número de tests entre volúmenes envolventes con menor ajuste (y menor coste).

Figura 4.33 Volúmenes envolventes a distinto nivel de detalle.

4.5.4.3 Librerías específicas para la Detección de Colisiones.

En esta sección se describen algunas librerías que implementan métodos de detección de colisión.

I-COLLIDE: Es una librería de detección de colisión para grandes entornos. Utiliza un método de dos fases con cajas envolventes y determina la colisión exacta entre pares de objetos poliédricos convexos. Utiliza la coherencia temporal para acelerar sus cálculos, siendo bastante rápido cuando el movimiento entre frames es pequeño.

RAPID: Es una librería para la detección de colisiones entre polígonos no estructurados. No realiza la fase de poda, de la que se encarga la librería VCOLLIDE que se verá a continuación. Es especialmente apropiada por su sencillez de uso y por su rapidez cuando hay pocos objetos en el entorno.

PQP: Es una librería similar a RAPID en cuanto a su interfaz. Es capaz de responder a diversas cuestiones sobre colisión con una cierta tolerancia. Puede usarse con polígonos sin necesidad de información topológica. Utiliza volúmenes basados en esferas de barrido para la determinación de la distancia entre objetos, así como diversos tipos de volúmenes envolventes que pueden ser seleccionados por el usuario.

V-COLLIDE: Es una librería construida para la poda previa en el sistema RAPID. Determina los pares de objetos que se encuentran potencialmente en contacto. Estos

objetos son utilizados por la librería RAPID para la determinación de colisión. Permite entornos con un gran número de objetos, estáticos y dinámicos, así como la inserción o borrado interactivo de objetos.

SWIFT: Es también una librería para la detección de colisiones que hace uso de la coherencia espacial y temporal mediante el uso de regiones de Voronoi y jerarquías de niveles de detalle, utilizando el algoritmo de Lin- Canny de las características más cercanas el cual ha sido mejorado. Se utiliza para poliedros convexos con movimiento rígido. Permite resolver cuestiones como la detección de intersección, la distancia exacta o aproximada entre objetos, y la determinación del contacto entre objetos.

SWIFT++: Es una librería basada en SWIFT. Se le ha incorporado la posibilidad de descomposición jerárquica de los objetos. Genera pares de jerarquías de volúmenes envolventes sobre las que comprobar la colisión con el algoritmo de la librería SWIFT.

H-COLLIDE: Es un entorno de trabajo para la detección de colisiones para aplicaciones *"Haptics"*. Está especializada en determinar los contactos entre un dispositivo y objetos en un entorno virtual. Utiliza una adaptación específica de algoritmos clásicos de detección de colisión para este tipo de aplicación. Además usa una descomposición espacial en rejillas uniformes así como una jerarquía de volúmenes envolventes basada en OBBTrees. También hace uso de la coherencia entre frames para realizar cálculos incrementales.

IMMPACT: Es una librería adecuada para modelos masivos compuestos de millones de primitivas geométricas. Utiliza optimizaciones para el almacenamiento y recuperación del modelo o partes del mismo desde memoria secundaria. Implementa grafos de solapamiento para obtener las zonas del modelo a recuperar de memoria secundaria. Utiliza también jerarquías de volúmenes envolventes y la coherencia espacial y temporal para mejorar su eficiencia.

DEEP: Es un algoritmo incremental para estimar el grado de penetración entre politopos convexos, así como la dirección de penetración entre objetos. Hace uso de la coherencia entre frames.

PIVOT: Es una librería para polígonos no convexos y cerrados en 2D. Utiliza el hardware gráfico y técnicas basadas en regiones de Voronoi para determinar la colisión, la intersección, la distancia de separación y los puntos de contacto. Hace uso de una técnica híbrida en el espacio de los objetos y de la imagen que permite ponderar el uso de la CPU y GPU según las necesidades del usuario.

V-CLIP: Está basado en el algoritmo de Lin-Canny [LC91] de características más cercanas y trata casos especiales que no maneja el algoritmo de Lin-Canny, además de

aportar una solución más simple y robusta. Es un algoritmo de bajo nivel que trata objetos que pueden ser poliedros no convexos, para esto los descompone utilizando una jerarquía de piezas convexas.

SOLID: Es también una librería para la detección de colisión entre objetos poliédricos convexos que pueden deformarse. Realiza una implementación eficiente del algoritmo GJK y cajas envolventes orientadas con los ejes como volumen envolvente.

CULLIDE y Q-CULLIDE: Son sistemas de poda para determinar la intersección y la auto-intersección entre objetos complejos y deformables mediante el uso del Hardware Gráfico. Realiza cálculos de visibilidad en la GPU para eliminar conjuntos de primitivas que no están próximas entre si.

QuickCD: Es una librería para la detección de colisiones de propósito general, capaz de tratar objetos extremadamente complejos que pueden estar formados por sopas de polígonos. Utiliza una jerarquía de volúmenes envolventes basados en *k*-dops.

4.5.5 Detección de Colisiones en la aplicación desarrollada.

Aunque todas estas librerías son en su mayoría muy eficientes y robustas además de estar en código abierto y ser grandes desarrollos por grupos dedicados exclusivamente a esta tarea resulta sumamente complicado su aplicación o enlace al entorno tridimensional que se está desarrollando, pues está escrito 100% en lenguaje ADA y ninguna de las librerías mencionadas anteriormente se encuentra escrita en este lenguaje, es así que la traducción de un lenguaje como C++ a ADA resulta sumamente complicado. De esta manera se optó por desarrollar el algoritmo de detección de colisiones completamente en ADA y si ningún tipo de librería como ayuda, simplemente utilizando ideas de una de las técnicas menos complejas para la detección de colisiones.

El algoritmo usado para la mano virtual emplea el principio de esferas envolventes pero de una forma distinta, no se utilizaran algoritmos para la detección con más detalle, solamente la detección de la colisión entre esferas. La elección de este método es debido a que la aplicación 3D tiene fuertes especificaciones en cuanto a tiempo real se refiere y básicamente este algoritmo es muy rápido debido a que solamente se limita a calcular distancias. En este caso no se elije una esfera envolvente para cada objeto, si no que se sitúan tantas esferas como sean necesarias en cada objeto de acuerdo al radio de los cilindros o diferentes espacios en la mano tal como se puede apreciar en la Figura 4.34. De esta manera no se alterara su forma cuando la mano adquiere su forma sólida (Figura 4.35).

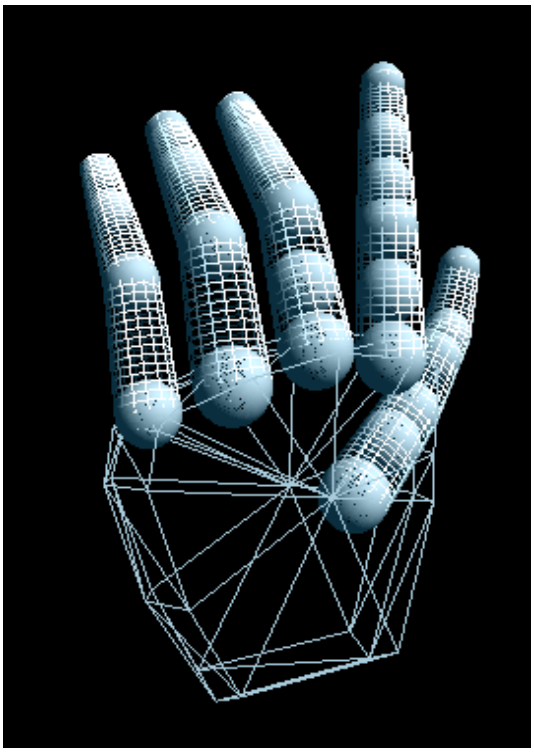


Figura 4.34 Mano con las esferas

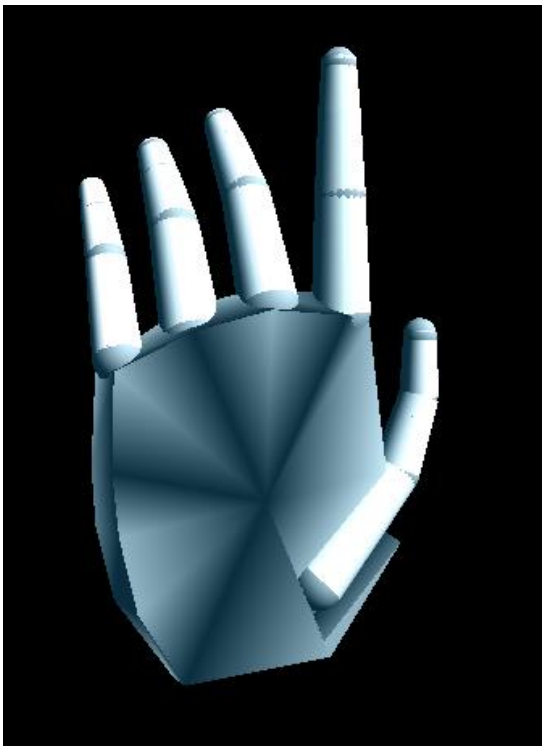


Figura 4.35 Mano completa

Fuente: Elaboración Propia.

Con la técnica de esferas envolventes tradicional lo que se busca es recubrir los objetos deseados con una esfera, así el primer paso es que el algoritmo detecte si hay colisión o no entre dichas esferas, si ocurre una colisión el algoritmo debe ahora hacer una serie de cálculos más precisos para determinar si realmente hay colisión entre los objetos ya que con objetos complejos generalmente abra mucho volumen de la esfera que no está ocupado por el objeto, los cálculos que se pueden realizar en este punto pueden variar; pueden ser algoritmos recursivos de jerarquía de esferas envolventes o en el peor de los casos comprobación píxel a píxel (esta última opción computacionalmente es muy costosa). Con la técnica utilizada en este proyecto se evitan los cálculos más rigurosos (calcular la colisión píxel a píxel o la jerarquía de esferas envolventes) después de la primera detección, simplemente se limita a calcular distancias entre esferas, cabe anotar que esta técnica fue factible por la anatomía de la mano y el hecho de que pueda rellenarse con esferas teniendo una buena exactitud.
En este caso la técnica que se ha utilizado es el test múltiple de intersecciones y por tanto se reduce el problema de la detección de colisiones a la detección de intersecciones en instantes concretos del tiempo.

La característica fundamental de este algoritmo es el hecho de que no estudia directamente la colisión entre las diferentes primitivas que forman la mano y los objetos a

sujetar si no que lo hace entre esferas que están amoldadas a la geometría de las primitivas (tantas esferas como sean necesaria), como siempre entre mas esferas se tenga se tendrá más precisión pero en contraste mas costo computacional. El saber si dos esferas intersecan es muy sencillo puesto que consiste en ver si la distancia entre los centros es menor que la suma de los radios. Dadas dos esferas caracterizadas por sus centros (x1, y1, z1) y (x2, y2, z2) y sus radios (r1, r2) podemos asegurar que no existe intersección entre ellas si se cumple.

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \leq r_1 + r_2$$

De esta manera los únicos elementos que intervienen en la representación son esferas, de manera que el problema se reducirá siempre a manejar conjuntos de esferas, independientemente de cuál sea el objeto real que subyace de la representación de las esferas. Básicamente el algoritmo sería de la siguiente manera:
En la llamada a *display*:

- Dibujo de la mano con las respectivas esferas envolventes en posición
- guardar las posiciones de las esferas envolventes en una matriz
- Llamar al procedimiento “colisión” encargado de efectuar los cálculos de las colisiones

En la llamada a *colision:*

- Se extraen la coordenadas cartesianas de cada esfera y se efectúa el cálculo de distancia entre cada esfera.
- Se evalúa si la distancia calculada entre cada esfera es inferior a la suma de sus radios si no es así los dedos siguen moviéndose.
- Cuando la distancia es inferior o igual a la suma de sus radios se dice que hay colisión, en este momento se pone a verdadero un booleano como bandera para saber si hay o no colisión.
- Se determina la acción que tomara el cuerpo después de detectada la colisión, en este caso que los dedos dejen de moverse.

5 RESULTADOS OBTENIDOS.

En el presente capitulo se detalla la forma empleada para realizar la verificación del entorno computacional gráfico 3D mediante la comprobación de diferentes tipos de agarre (cilíndrico, de punta y de gancho). Los listados de valores articulares que se emplearon para llevar a cabo la simulación de los diferentes tipos de agarre corresponde a un archivo de texto plano.

Debido a que la respuesta en tiempo real del sistema depende en gran medida de las capacidades de la máquina, se ha desarrollado la prueba en tres computadores de diferentes características con el fin de analizar la respuesta del sistema obtenida en cada uno de ellos. Las características de los PC's empleados se listan a continuación:

PC 1: Computador de Escritorio

- Procesador: AMD Athlon 64bits 3500+ Velocidad: 2.21GHz
- Memoria RAM: 1.00 GB
- Tarjeta Aceleradora de Video: ATI X1300 de 256MB

PC 2: Computador Portátil TOSHIBA Satellite Pro

- Procesador: AMD Athlon X2 - Dual-Core Processor TK-55 Velocidad: 1.76GHz
- Memoria RAM: 896 MB
- Tarjeta Aceleradora de Video: ATI X1100 de 128MB (Memoria compartida).

PC 3: Computador de Escritorio

- Procesador: AMD Athlon 64bits 3000+ Velocidad: 2.01GHz
- Memoria RAM: 1.50 GB
- Tarjeta Aceleradora de Video: ATI All In Wonder 9600 de 128MB

De acuerdo a la información suministrada uno de los requerimientos es que el sistema sea capaz de responder como mínimo en 130ms ante un evento. Para este caso a nivel general cada archivo a simular contiene 110 líneas de datos, así la simulación completa debería tener una duración total de 14.300ms o durar un menor tiempo. Sin embargo el

sistema fue configurado para responder si es posible en 100ms ante un evento en cuanto a cambios en alguna variable articular.

5.1 RESULTADOS DE SIMULACIÓN PARA EL TIPO DE AGARRE CILÍNDRICO.

Para este tipo de agarre se hace la aclaración de que el dedo pulgar ya debe estar posicionado, puesto que el modelo matemático de la mano no permite el agarre de objetos cilíndricos largos en la posición inicial por defecto de la mano, debido a que el objeto haría contacto con dicho dedo evitando su trayectoria para su correcto posicionamiento y posterior agarre. Por lo cual la posición inicial de la mano para la simulación es como lo muestra la siguiente Figura 5.1; en la Figura 5.2 se puede observar que la mano realiza adecuadamente el agarre del objeto cilíndrico.

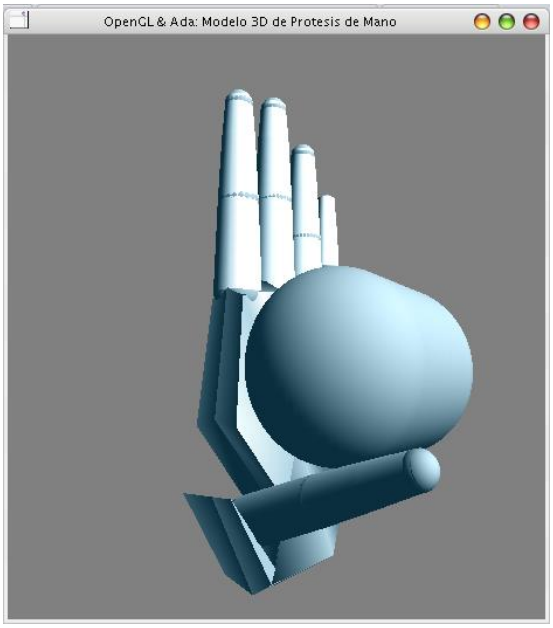


Figura 5.1 Posición inicial para agarre tipo cilíndrico.

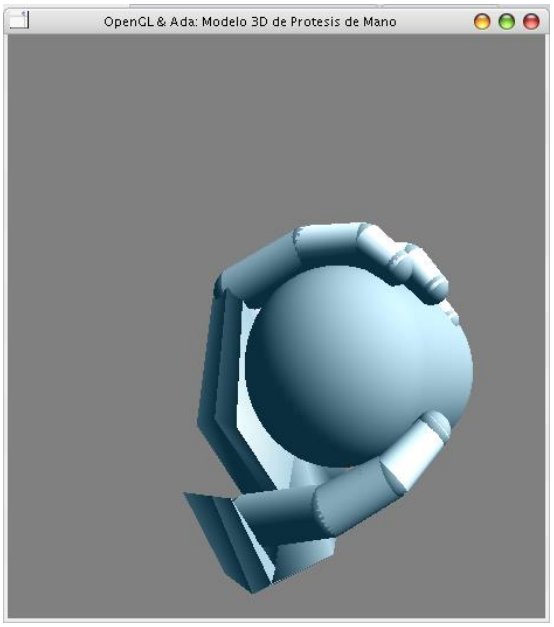


Figura 5.2 Posición final para agarre tipo cilíndrico.

Fuente: Elaboración Propia.

Simulación con PC 1	Simulación con PC 2	Simulación con PC 3
TIMER: 14:51:39.84 — Cant. Lineas Leidas: 110 — Tiempo Duracion de la Simulacion (ms): 10.862 — Periodo Maximo entre cada Linea de Datos (ms): 100	TIMER: 14:56:21.34 — Cant. Lineas Leidas: 110 — Tiempo Duracion de la Simulacion (ms): 15.377 — Periodo Maximo entre cada Linea de Datos (ms): 100	HORA: 15:57:51.24 — Cant. Lineas Leidas: 110 — Tiempo Duracion de la Simulacion (ms): 11.001 — Periodo Maximo entre cada Linea de Datos (ms): 100

Figura 5.3 Resultados de simulación para el tipo de agarre cilíndrico.

Fuente: Elaboración Propia.

Como se puede observar en la Figura 5.3 el tiempo de duración de la simulación para los PC's 1 y 3 es de 10.862ms y 11.001ms respectivamente; el tiempo máximo de simulación esperado es de 11.000ms, por lo cual se puede observar que el sistema responde con buena precisión dentro del rango de tiempo de simulación esperado y además logra cumplir rigurosamente el máximo tiempo de duración de la simulación permitido (14.300ms).

En cuanto al tiempo de duración de la simulación para el PC 2 es de 15.377ms, pero el tiempo de simulación esperado es de 11.000ms pues el periodo de tiempo de ejecución entre cada línea es ajustable con un valor mínimo de 100ms para un total de 110 líneas a simular, de esta manera las pruebas se realizan con el menor tiempo de respuesta posible (100ms). Así, se obtuvo un desfase en tiempo de respuesta de 4.377ms, lo que es un desfase considerable ya que sobrepasa el máximo tiempo de duración de la simulación permitido (14.300ms). Por lo tanto para este equipo el menor tiempo de respuesta posible fue 15.377ms, siendo muy difícil cumplir con los tiempos de ejecución requeridos.

5.2 RESULTADOS DE SIMULACIÓN PARA EL TIPO DE AGARRE GANCHO.

A continuación se muestran las figuras (Figura 5.4 y Figura 5.5) correspondientes a la posición inicial y final obtenidas en la simulación para este tipo de agarre.

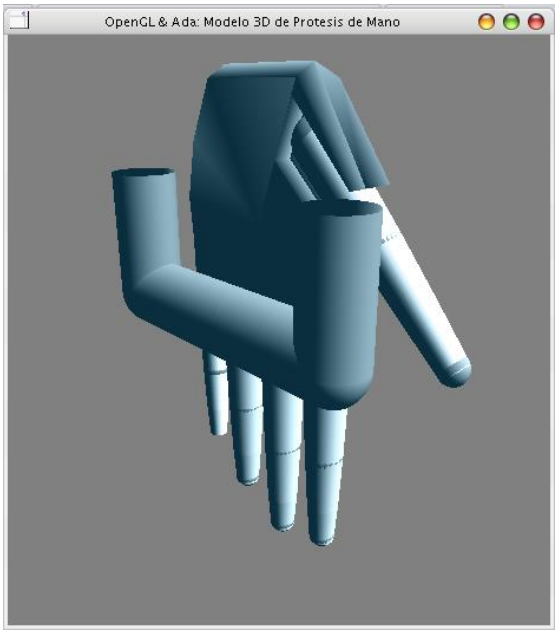


Figura 5.4 Posición inicial para agarre tipo gancho.

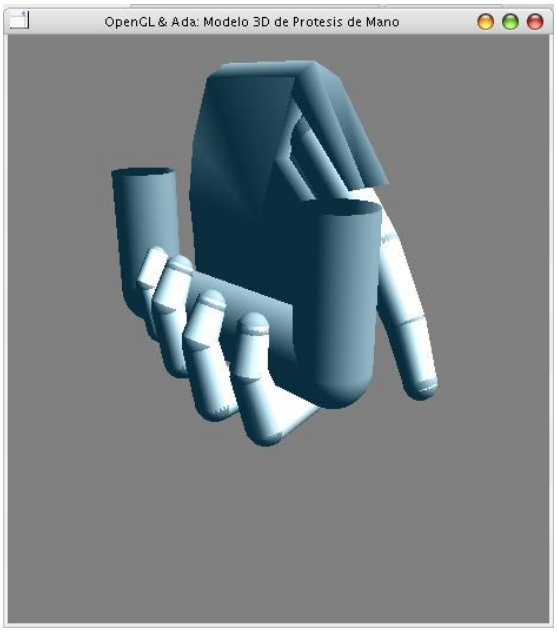


Figura 5.5 Posición final para agarre tipo gancho.

Simulación con PC 1	Simulación con PC 2	Simulación con PC 3
TIMER: 16:15: 3.15	TIMER: 16:31:58.64	HORA: 16: 8:28.88
Cant. Lineas Leidas: 110	Cant. Lineas Leidas: 110	Cant. Lineas Leidas: 110
Tiempo Duracion de la Simulacion (ms): 10.973	Tiempo Duracion de la Simulacion (ms): 15.713	Tiempo Duracion de la Simulacion (ms): 10.996
Periodo Maximo entre cada Linea de Datos (ms): 100	Periodo Maximo entre cada Linea de Datos (ms): 100	Periodo Maximo entre cada Linea de Datos (ms): 100

Figura 5.6 Resultados de simulación para el tipo de agarre gancho.
Fuente: Elaboración Propia.

Como se puede observar en la Figura 5.6 el tiempo de duración de la simulación es de 10.973ms y 10.996ms para los PC's 1 y 3 respectivamente; el tiempo máximo de simulación esperado es de 11.000, por lo cual se puede observar que el sistema responde dentro del rango de tiempo de simulación esperado y además logra cumplir holgadamente el máximo tiempo de duración de la simulación permitido (14.300ms).

Respecto al tiempo de duración de la simulación para el PC 2 este es de 15.713ms, pero el tiempo de simulación esperado es de 11.000ms en este caso pues el periodo de tiempo de ejecución entre cada línea es ajustable con un valor mínimo de 100ms. Así, se obtuvo un desfase en tiempo de respuesta de 4.713ms, lo que es un desfase considerable puesto que sobrepasa el máximo tiempo de duración de la simulación permitido (14.300ms). Por lo tanto para este equipo el menor tiempo de respuesta posible fue 15.713ms, no cumpliendo así con los tiempos de ejecución requeridos.

5.3 RESULTADOS DE SIMULACIÓN PARA EL TIPO DE AGARRE PUNTA.

En la Figura 5.7 y la Figura 5.8 se puede apreciar las posiciones inicial y final obtenidas en la simulación para este tipo de agarre. Igualmente de la Figura 5.9 se puede concluir que en los PC's 1 y 3 el sistema empleo un tiempo simulación de 10.931ms y 10.526ms respectivamente y el tiempo máximo de simulación esperado es de 11.000, por lo cual se puede decir que el sistema responde dentro del rango de tiempo de simulación esperado y además logra cumplir cómodamente el máximo tiempo de duración de la simulación permitido (14.300ms).

En cuanto al tiempo de duración de la simulación del sistema en el PC 2 se puede observar que es de 14.826ms, pero el tiempo de simulación esperado es de 11.000ms en este caso pues el periodo de tiempo de ejecución entre cada línea es ajustable con un valor mínimo de 100ms. Así, se obtuvo un desfase en tiempo de respuesta de 3.826ms, lo que es un desfase considerable ya que sobrepasa el máximo tiempo de duración de la

simulación permitido (14.300ms). Por lo tanto para este equipo el menor tiempo de respuesta posible fue 14.826ms, no cumpliendo así con los tiempos de ejecución requeridos.

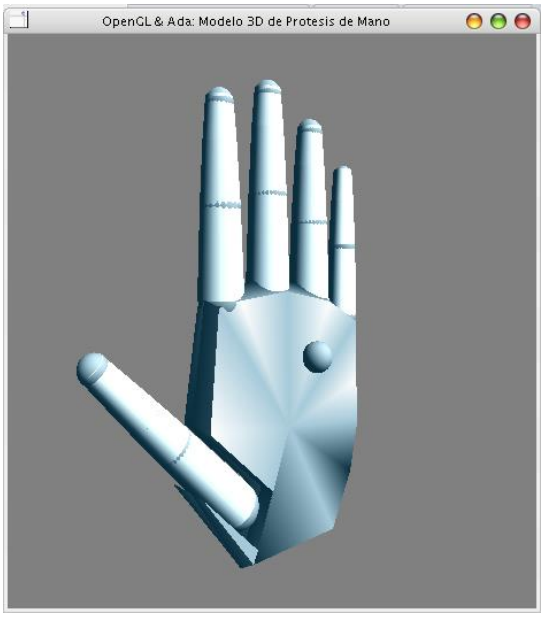


Figura 5.7 Posición inicial para agarre tipo punta.

Figura 5.8 Posición final para agarre tipo punta.

Fuente: Elaboración Propia.

Simulación con PC 1	Simulación con PC 2	Simulación con PC 3
TIMER: 16:56:28.70 — Cant. Lineas Leidas: 110 — Tiempo Duracion de la Simulacion (ms): 10.931 — Periodo Maximo entre cada Linea de Datos (ms): 100	TIMER: 16:54:28.16 — Cant. Lineas Leidas: 110 — Tiempo Duracion de la Simulacion (ms): 14.826 — Periodo Maximo entre cada Linea de Datos (ms): 100	HORA: 16:19:35.91 — Cant. Lineas Leidas: 110 — Tiempo Duracion de la Simulacion (ms): 10.526 — Periodo Maximo entre cada Linea de Datos (ms): 100

Figura 5.9 Resultados de simulación para el tipo de agarre punta.

Fuente: Elaboración Propia.

5.4 CONCLUSIÓN DE LOS RESULTADOS DE LA SIMULACIÓN.

Según los resultados anteriormente mostrados, se puede observar que a pesar de que el sistema operativo empleado no es un sistema operativo de tiempo real, la aplicación software desarrollada realiza correctamente la simulación de los diferentes tipos de movimiento de agarre y además responde adecuadamente a los requerimientos temporales, desarrollando apropiadamente el muestreo y simulación de los datos de

variables articulares que se le suministran como entrada, gran parte de eso se debe a que la aplicación fue desarrollada empleando un lenguaje de programación concurrente como lo es el lenguaje Ada en conjunto con la técnica de diseño de sistemas de tiempo real (HRT-HOOD) y el haber realizado la parte grafica tridimensional empleando una librería gráfica de tiempo real basada en OpenGL (Globe_3D) con un algoritmo de detección de colisiones sencillo y eficiente. Además, según los resultados obtenidos la aplicación debería ser empleada con un computador de características similares o superiores a las de los PC 1 y PC 3 cuyas características hardware corresponden a computadores que actualmente en el mercado se pueden conseguir a precios asequibles, estos requerimientos hardware se deben a que la parte gráfica tridimensional de la aplicación requiere de la realización de diversos cálculos cuyos resultados sean rápidos y precisos, lo cual no es factible en computadores con bajas características hardware. También es preciso indicar que no se desarrollo la aplicación empleando la tecnología GRID a pesar de ser una buena solución, básicamente debido a que una GRID utiliza de forma coordinada recursos que no están sujetos a un control centralizado, es decir, permite una nueva forma de computación distribuida, en la cual los recursos pueden ser heterogéneos (tener diferentes arquitecturas, computadores, clusters...) y se encuentran conectados mediante redes (64). Sin embargo, otro de los requerimientos es el de considerar que el usuario que va a emplear la aplicación requiere que ésta se ejecute en su computador, y además se asume que dicho computador no se encuentra conectado a ninguna red de computadores, por ello el empleo de una GRID no sería factible en este caso en particular.

Cabe aclarar que se realizaron diversas pruebas, con diferentes parámetros de simulación, tales como cambio del máximo periodo de simulación, diferente cantidad de líneas de datos en los archivos a simular, etcétera. Obteniéndose en promedio resultados que se corresponden con los anteriormente mostrados. Resulta igualmente importante resaltar que en la máquina la aplicación se debería de ejecutar sin ninguna otra aplicación ejecutándose en paralelo con ella, puesto que el rendimiento del software desarrollado se vería afectado si en la máquina se ejecuta otro tipo de aplicación que consuma buena cantidad de recursos del sistema, ello se debe a que el tipo de sistema operativo empleado no es un sistema operativo de tiempo real, lo cual conlleva a considerar que la aplicación desarrollada sea de tiempo real no estricto.

Otra forma con la cual se validó el modelo gráfico desarrollado fue realizando una comparación entre la distancia que hay de la punta de los dedos al plano que contiene la palma de la mano después de cierto movimiento al rotar con determinado ángulo las diferentes falanges de los dedos entre el modelo matemático desarrollado en (34) y el modelo tridimensional desarrollado en este proyecto. La idea es que un programa desarrollado en Matlab, el cual contiene las ecuaciones derivadas de la matriz de transformación homogénea resultante para los dedos medio, índice y pulgar arroja las

coordenadas de la punta de cada dedo después de ciertas rotaciones en sus falanges, de aquí se puede obtener fácilmente la distancia observando en que ejes se encuentra el plano que contiene la palma de la mano y por consiguiente el eje resultante siempre dará la distancia deseada. El plano que contiene la palma de la mano para el modelo desarrollado en (34) es el plano X-Z, por lo tanto el eje Y dará la distancia a ese plano. El plano que contiene la palma de la mano del modelo tridimensional desarrollado en este proyecto es el plano X-Y por lo tanto el eje Z dará la distancia deseada. Así, después de realizar las mismas rotaciones en cada modelo, las distancias deberían ser las mismas o muy aproximadas para que el modelo desarrollado en este proyecto sea válido. Cabe anotar que las coordenadas que arroja Matlab deben multiplicarse por el factor 0.026 (este factor resulta de una regla de tres simple) el cual se utilizó para adecuar las medidas de la prótesis real a las dimensiones que maneja OpenGL.

El programa utilizado en Matlab es el siguiente:

```
clear all
 cons1 =[ pi/2     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00;
       0.00     pi/2     0.00     0.00     0.00     0.00     0.00     0.00     0.00;
       0.00     0.00     pi/2     0.00     0.00     0.00     0.00     0.00     0.00;

       0.00     0.00     0.00     pi/2     0.00     0.00     0.00     0.00     0.00 ;
       0.00     0.00     0.00     0.00     pi/2     0.00     0.00     0.00     0.00 ;
       0.00     0.00     0.00     0.00     0.00     pi/2     0.00     0.00     0.00;

       0.00     0.00     0.00     0.00     0.00     0.00     pi/2     0.00     0.00 ;
       0.00     0.00     0.00     0.00     0.00     0.00     0.00     pi/2     0.00 ;
       0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     pi/2;
      % con diferentes ángulos al tiempo
      %              20
       pi/2     pi/4     pi/9     0.00     0.00     0.00     0.00     0.00     0.00 ;
       0.00     0.00     0.00     pi/2     pi/4     pi/9     0.00     0.00     0.00 ;
       0.00     0.00     0.00     0.00     0.00     0.00     pi/2     pi/4     pi/9 ;
      % 30       25       35
       pi/6     5*pi/36   7*pi/36   0.00     0.00     0.00     0.00     0.00     0.00 ;
       0.00     0.00     0.00     pi/6     5*pi/36   7*pi/36   0.00     0.00     0.00 ;
       0.00     0.00     0.00     0.00     0.00     0.0      pi/6     5*pi/36   7*pi/36;

       pi/4     pi/4     pi/4     0.00     0.00     0.00     0.00     0.00     0.00 ;
       0.00     0.00     0.00     pi/4     pi/4     pi/4     0.00     0.00     0.00 ;
       0.00     0.00     0.00     0.00     0.00     0.00     pi/4     pi/4     pi/4;
 ];
```

```
Ts    = 1e-3 ;
[f c] = size(cons1) ;
instant = Ts;
for i = 1:f,
  x(i,:) = mgd_dedos(cons1(i,:))
end
```

En este programa se entregan los diferentes ángulos para cada falange de cada uno de los dedos (medio, índice, pulgar). La función a la cual llama este programa es (mgd_dedos) cuyo código es el siguiente:

```
function x = mgd_dedos(q)

q1 = q(1); q2 = q(2); q3= q(3); q5 = q(4); q6 = q(5); q7 = q(6); q9 = q(7); q10 = q(8); q11= q(9);

% Modelo geométrico directo para los tres dedos de la protesis de mano

D2 = 0.057;     D3 = 0.039;     D4 = 0.027;
D6 = 0.052;     D7 = 0.036;     D8 = 0.025;
D10 = 0.032;    D11 = 0.039;    D12 = 0.044;

B5 = 0.021;     B9 = 0.0105;

A = (sqrt(2)/2);

xa1 = 0;
ya1 = -cos(q1+q2)*sin(q3)*D4-sin(q1+q2)*cos(q3)*D4-sin(q1+q2)*D3-sin(q1)*D2;
za1 = -sin(q1+q2)*sin(q3)*D4+cos(q1+q2)*cos(q3)*D4+cos(q1+q2)*D3+cos(q1)*D2;

xa2 = B5;
ya2 = -cos(q5+q6)*sin(q7)*D8-sin(q5+q6)*cos(q7)*D8-sin(q5+q6)*D7-sin(q5)*D6;
za2 = -sin(q5+q6)*sin(q7)*D8+cos(q5+q6)*cos(q7)*D8+cos(q5+q6)*D7+cos(q5)*D6;

xa3 = sin(q10+q11)*D12 + sin(q10)*D11 + B9;
ya3 =-A*(sin(q9)+cos(q9))*cos(q10+q11)*D12 - A*(sin(q9)+cos(q9))*(cos(q10)*D11 + D10);
za3 =-A*(sin(q9)-cos(q9))*cos(q10+q11)*D12 - (sin(q9)-cos(q9))*(cos(q10)*D11 + D10);
x = [xa1 ya1 za1 xa2 ya2 za2 xa3 ya3 za3];
```

El resultado se muestra en la Tabla 5.1 para las siguientes pruebas: Estas son las distancias encontradas para los diferentes ángulos. Los ángulos se dan en grados y a la distancia que se da en el eje z en globe_3D se le debe sumar 10 ya que el plano esta ubicado en -10 y posteriormente dividirla por 0.026 para dejarlo en las mismas unidades.

Distancias	Dedo medio	Distancia Matlab	Distancia Globe:3D	Conversion a mm.
F. Proximal	90º	123	-6.87	120.3
F. Medial	90º	66	-8.37	62.7
F. Distal	90º	27	-9.39	23.4

Distancias	Dedo Índice	Distancia Matlab	Distancia Globe:3D	Conversion a mm
F. Proximal	90º	113	-7.05	113.07
F. Medial	90º	61	-8.41	61.1
F. Distal	90º	25	-9.35	25

Tabla 5.1 Resultado comparativo con el modelo matemático de la mano de (34).

Como se puede apreciar para un ángulo de 90º en las diferentes falanges de los dedos medio e índice, las distancias encontradas son muy similares encontrándose una variación muy pequeña lo cual indica que el modelo tridimensional realizado es congruente con el modelo matemático de la mano pues con las mismas rotaciones obtenemos aproximadamente las mismas distancias. Los errores de aproximación pueden deberse a la forma en que se calculan las diferentes operaciones matemáticas en OpenGL y Matlab. Es importante anotar que la validación para el dedo pulgar no pudo realizarse exitosamente pues al realizar la simulación en el modelo matemático para la mano en reposo, es decir con todos los ángulos de las articulaciones en cero se presenta una inconsistencia pues el dedo pulgar aparece desplazado 8,13 cm del plano de referencia lo cual no debería suceder.

6 CONCLUSIONES, CONTRIBUCIONES Y TRABAJO FUTURO.

Este trabajo de grado ha presentado una herramienta que posteriormente permitirá el entrenamiento de pacientes con amputación de mano que serán candidatos a una prótesis robótica de mano controlada por señales electromiográficas. La herramienta desarrollada corresponde a un entorno virtual que permite la visualización de un modelo de la mano realizando diferentes tipos de agarre y mostrando que responde al modelo matemático de mano de 9GDL's preestablecido. En este capítulo se resumen las conclusiones del trabajo, sus contribuciones e igualmente se describe el trabajo futuro a realizar para éste.

6.1 CONCLUSIONES.

- El software libre está teniendo más impulso a nivel mundial, cada vez mas programadores y científicos adoptan esta tendencia, no solo porque es gratis si no porque goza de otras características favorables como la facilidad para que muchos programadores con experiencia tengan acceso al código fuente y puedan aportar masivamente al mejoramiento y desarrollo de una aplicación. Actualmente existen numerosas herramientas libres para el modelamiento 3D en tiempo real, la elección de una dependerá de las características de la aplicación. Hay numerosos programas y cada uno posee un lenguaje de programación especifico, si se quiere realizar una aplicación 3D en tiempo real la elección del lenguaje nativo del programa es importante pudiéndose elegir por ejemplo entre java en tiempo real, C++, y por supuesto Ada, otra de las consideraciones es la profundidad a la que la persona quiere llegar con respecto a la programación de la aplicación, pues hay programas que pueden ser manejados por personas con poca experiencia (teniendo poco o nada de control sobre la exactitud del modelo 3D, su movimiento e interfaz de usuario), programas para personas con mediana experiencia donde se le permite al programador tener acceso a muchas variables e incluso al código de ciertas partes en el proceso de modelado, o se pueden encontrar también simplemente los motores gráficos que implementan las diferentes funciones de una API determinada para ser tratadas por programadores con experiencia pues el desarrollo de la aplicación debe lograrse a través de un entorno de desarrollo estándar para un lenguaje de programación y prácticamente el 80% de la aplicación corre por cuenta de la persona,

el otro 20% la es facilidad que brinda el motor gráfico para implementar primitivas 3D, pero la manejabilidad (interfaz de usuario) y eficiencia del programa respecto a determinadas características dependen en gran medida de la habilidad del programador. Para este caso se escogió un motor gráfico escrito en lenguaje de programación Ada puesto que según la bibliografía es uno de los lenguajes más eficientes en cuanto a tiempo real se refiere, sin embargo uno de los mayores problemas es la poca o inexistente documentación sobre un motor gráfico escrito en Ada y basado en OpenGL, ya que el lenguaje nativo de OpenGL es C. Una vez terminada la aplicación se pudo constatar que gracias a la concurrencia del lenguaje Ada se pudieron manejar tiempos de respuesta en la aplicación que corresponden con las exigencias de tiempo real no estricto.

- El diseño y construcción de un entorno virtual es una tarea compleja, pues el desarrollo del modelo 3D de la mano implica el conocimiento de una API determinada, para este caso, OpenGL que si bien es muy potente resulta ser muy extenso, igualmente se sumaba el inconveniente de que el lenguaje escogido es Ada ya que OpenGL está escrito originariamente en C y casi toda la documentación al respecto incluidos los ejemplos están escritos en C, lo que hizo una tarea dispendiosa entender el funcionamiento de OpenGL y además emplear sus funciones mediante el lenguaje Ada dado que para este lenguaje con respecto de OpenGL la documentación es muy reducida obligándonos así a descubrir su funcionamiento prácticamente por nuestra cuenta. Cuando se terminó la aplicación se pudo constatar el gran beneficio de programar la aplicación en Ada pues el manejo a través de la interfaz de usuario del modelo 3D por medio de tareas fue bastante preciso, pues se cumplió con los requerimientos en tiempo de respuesta para lograr las exigencias de tiempo real no estricto, se puede observar que Ada tiene gran concurrencia o gran capacidad para manejar tiempo real pues la concurrencia se encuentra definida en el lenguaje mediante el empleo de tareas. A su vez OpenGL funciona perfectamente y demuestra ser una API muy potente y flexible para el tratamiento de objetos 3D, pues para este caso el manejo de luces, sombreado, perspectiva, construcción de las diferentes partes de la mano y por supuesto las diferentes transformaciones geométricas necesarias para la construcción y movimiento de la mano son de muy buena calidad y eficientes en cuanto a tiempo de respuesta para el renderizado del modelo, además de la facilidad para el soporte de manejo de eventos y llamadas por parte del teclado y el ratón gracias a su librería auxiliar Glut, asimismo de todo ello cabe anotar la sencillez de construcción que demostró OpenGL gracias a su funcionamiento como maquina de estados lo cual quedó en evidencia en el momento de crear el modelo, aplicarle color, sombra, luces etc. Otro de los aspectos a resaltar y quizá uno de los más importantes es su congruencia matemática asociada a la robótica dado que está fuertemente ligado al algebra matricial lo cual permitió crear un modelo 3D congruente con el modelo matemático establecido. En cuanto a la detección de colisiones

necesaria para la correcta visualización y agarre de los diferentes objetos en la aplicación, se puede destacar que si bien existen numerosos algoritmos (de hecho es un campo abierto a la investigación pues todavía no hay un método 100% eficiente) y algunos con tiempo de respuesta muy bueno, es complicado estudiarlos todos a fondo para elegir el mejor pues son muchos y además se basan en un fuerte tratamiento matemático a lo que se le suma que no hay ninguno depurado en Ada (por lo menos en este caso) lo que implica comprender el algoritmo y además traducirlo de un lenguaje determinado (generalmente en C o C++) a Ada, tarea nada sencilla debido a la gran diferencia en sus sentencias y de hecho a que cada lenguaje posee algunas cosas que el otro no. En este caso se opto por estudiar algunos algoritmos generales y escoger aquel que pudiera Adaptarse de tal forma que cumpla con los tiempos necesarios y buena precisión en la detección de colisiones, así, una Adaptación del método de volúmenes envolventes (para este caso fueron esferas) demostró gran sencillez en el manejo del código y gran eficiencia y precisión pues los agarres son buenos y el sistema completo responde bien ante los cálculos necesarios para la detección, además este método es flexible en cuanto a la precisión Vs tiempo de respuesta pues el programador es libre de elegir que tanta precisión quiere en la detección a costa de su tiempo de respuesta, algo que no es posible en muchos algoritmos.

- Aunque se consideró que la capacidad de aprehensión de la mano era posible por lo menos para 4 agarres distintos (cilíndrico, de gancho, de punta y lateral) en la aplicación se consideraron solo tres de nuevo debido al poco tiempo, pues la Adaptación de la mano para cada agarre requiere no solo de la creación de los nuevos objetos si no la intervención de diferentes variables a la hora de programar y además la creación de la lista de los parámetros articulares para su simulación, la cual es extensa y además debe ser exacta,. Así la aplicación demostró ser eficiente en cuanto al agarre de los diferentes objetos para los tres tipos de agarre seleccionados (cilíndrico, de punta y de gancho) ya que la precisión es bastante buena para tres tamaños diferentes de cada objeto, la mano logra Adaptarse a dicho tamaño y logra el agarre con escaso error, en algunos casos el error es mas considerable en cuanto a exactitud del agarre por parte de los dedos anular y meñique pero este error se debe a que el modelo de la mano solo contempla tres dedos, pulgar, índice y corazón, dando así solo 9 grados de libertad, por lo tanto el anular y el meñique no son independientes y se supone que siguen fielmente los ángulos de movimiento de cada falange del dedo corazón, sin embargo debido a las diferencias estructurales en cuanto a tamaño y ubicación en la palma, el hecho de que sigan fielmente al dedo corazón ocasiona desacoples al momento del agarre por lo que fue necesario realizar ciertos ajustes de ángulos en estos dos dedos para cada tamaño preconcebido del objeto. La razón de estos dos dedos extras que no son independientes y que no aportan grados de libertad al sistema es puramente estético, fuera de esto se puede concluir que los

agarres se realizan con éxito gracias al algoritmo de detección de colisiones implementado que permite la adaptación de la mano a diferentes tamaños del objeto y que cumple las respuestas necesarias para trabajar en tiempo real suave.

6.2 CONTRIBUCIONES.

Actualmente existen entornos virtuales con el mismo fin de este proyecto pero desarrollados con herramientas comerciales, lo cual implica al paciente un costo extra de adquisición. En ese sentido este trabajo pretende realizar dicha aplicación con herramientas software totalmente libre con características de tiempo real y que sea multiplataforma. Es así que para el desarrollo de la aplicación se escogió el lenguaje de programación Ada debido a su gran flexibilidad en cuanto a tiempo real se refiere ya que su entorno de desarrollo (GNAT) es uno de los más potentes y completos entre los IDE libres disponibles y gracias a que la concurrencia se encuentra establecida en el lenguaje. Otro aspecto relevante en el proyecto es que actualmente existen pocos proyectos relacionados con aplicaciones 3D escritos en lenguaje Ada, este trabajo pretende además crear un vinculo entre una de las interfaces 3D estándar más conocidas (*OpenGL*) y el lenguaje de programación Ada, dando así la posibilidad de tener un entorno virtual con características de tiempo real, multiplataforma y libre.

Otro de los aspectos importantes es acerca de cómo se trabaja con Ada para la creación de escenas 3D, pues actualmente hay variada información de cómo se estructura y funciona Ada, sin embargo es muy poca la relativa a su uso en entornos tridimensionales como no es el caso de C, C++, Java, Visual Basic entre otros. Así gracias al desarrollo de este proyecto se brinda la posibilidad de profundizar más en un lenguaje tan versátil y potente además de estudiar y analizar el funcionamiento del motor gráfico Globe_3D que está escrito totalmente en Ada y del cual no existe documentación alguna sobre su funcionamiento. Cabe anotar que para la carrera es muy importante conocer lenguajes que permitan la creación de sistemas en tiempo real y Ada es un lenguaje que no ha sido oficialmente estudiado en la carrera de Ingeniería en Automática Industrial.

6.3 TRABAJO FUTURO.

- Mejorar el aspecto físico de la mano, de tal forma que tenga una apariencia más humana para lo cual se necesita el mapeo de texturas y el uso de funciones más avanzadas en OpenGL para sombreado y tratamiento de curvas, sin embargo ello debe hacerse de una forma especial puesto que el uso de dichas técnicas conlleva a un detrimento en el rendimiento del sistema.

- Emplear o adicionar un algoritmo de detección de colisiones más eficiente, que por lo general resultan ser muy complejos y ya están implementados en diversas librerías pero en lenguajes C++, Java, entre otros; la dificultad radica en traducir dicho algoritmo tan complejo en lenguaje de programación Ada.

- Implementar en el sistema la creación de registros en video de las diferentes simulaciones.

- Desarrollar más objetos y más tipos de agarres, e implementar la facilidad de que la mano pueda mover el objeto a medida que va desarrollando el agarre con el fin de que se pueda lograr una mejor aprehensión.

7 BIBLIOGRAFÍA.

1. Cirugía y trauma de mano. [En línea] [Citado el: 21 de Noviembre de 2007.] http://www.plastica-medica.com.mx/ReconstructivaCirugiaTraumaMano.aspx.
2. **Mexico, Revista Digital Universitaria UNAM -.** La robótica aplicada al ser humano: biónica. [En línea] Dirección General de Servicios de Cómputo Académico-UNAM. [Citado el: 20 de Febrero de 2008.] http://www.revista.unam.mx/vol.6/num1/art01/art01-1.htm.
3. **Banguero, L. F., Herrera, A. y Jaramillo, R. M.** sEMG para el control de prótesis bioeléctricas. [En línea] 2006. http://bioinstrumentacion.eia.edu.co/docs/signals/2006/exposiciones/sEMG_documento.pdf.
4. **Millstein, S.G., Heger, H. y Hunter, G. A.** *Prosthetic use in adult upper limb amputees: a comparison of the body powered and electrically powered prostheses.* s.l. : Prosthet Orthot Int., 1986. págs. 27-34. Vol. 10.
5. **Crone, N.** *A comparison of myoelectric and standard prostheses - A case study of a pre-school aged congenital amputee.* s.l. : Canadian Journal of Occupational Therapy, 1986. págs. 217-222. Vol. 53.
6. **Roeschlein, R. A. y Domholdt, E.** *Factors related to successful upper extremity prosthetic use.* s.l. : Prosthet Orthot Int., 1989. págs. 14-18. Vol. 13.
7. **Herberts, P., y otros.** *Rehabilitation of unilateral below-elbow amputees with myoelectric prostheses.* s.l. : Scandinavian Journal of Rehabilitation Medicine, 1980. págs. 123-128. Vol. 12.
8. **Scoufield, J. y Schreiner, S.** *Design of a myoelectric prosthetic training and assessment system.* s.l. : Proceedings of the IEEE 30th Annual Northeast Bioengineering Conference, 2004. págs. 200-201. Vol. 30.
9. **Heckathorne, C.W.** *Prosthetic arm design and simulation system (PADSS) for assessing alternative fitting of upper-limb prostheses.* s.l. : Capabilities, 1995. págs. 1-2. Vol. 4.
10. **Martinez, M.** *Desarrollo de un sistema de edición de escenas basado en una librería gráfica de alto nivel: OpenSceneGraph.* Escuela Técnica Superior de Ingeniería. Valencia : Universidad de Valencia, 2006. págs. 11-42.
11. VisualNastran 4D. [En línea] http://www.aertia.com/productos.asp?pid=88.
12. Cosmos/motion. [En línea] http://www.solidworks.com.
13. 3D GameStudio. [En línea] http://www.conitec.net.

14. **Mendez, A.** *Motores de Juegos.* Informática Gráfica, Universitat Jaume. 2003. Proyecto de Investigación.
15. Torque Game Engine. [En línea] http://www.garagegames.com.
16. Globe_3D - stands for GL Object Based Engine for 3D. *SourceForge.* [En línea] http://globe3D.sourceforge.net; http://homepage.sunrise.ch/mysunrise/gdm/g3d.htm.
17. Blender. [En línea] http://www.blender.org.
18. Ogre. *SourceForge.* [En línea] http://ogre.sourceforge.net.
19. Crystal Space. *SourceForge.* [En línea] http://crystal.sourceforge.net.
20. Genesis3D. [En línea] http://www.genesis3D.com.
21. OpenGL. [En línea] www.opengl.org.
22. **Burns, A. y Wellings, A.** *Concurrent And Real-Time Programming In Ada.* s.l. : Cambridge University Press, 2007.
23. **Wright, R., Lipchak, B. y Haemel, N.** *OpenGL Superbible - comprehensive tutorial and reference.* 4. New York : Addison-Wesley, 2007. pág. 1262.
24. **Howard, T. y Murta, A.** *Graphics Programming with OpenGL.* Manchester : University of Manchester, 1999. págs. 7-14.
25. **Mackenzie, C. L. y Iberall, T.** *The grasping hand.* Amsterdam : North Holland, 1994. págs. 17-31.
26. La cirugía plástica – anatomía de la mano. [En línea] [Citado el: 24 de Enero de 2008.] http://www.healthsystem.virginia.edu/uvahealth/adult_plassurg_sp/anatomy.cfm.
27. **Tylor, C. L. y Chwartz, R. J.** *The anatomy and mechanics of the human hand.* s.l. : Artificial Limbs, 1955. págs. 22-35. Vol. 2.
28. **Miralles, R. C.** *Valoración del daño corporal en el aparato locomotor.* Barcelona : Elsevier-Masson, 2001. págs. 177-179.
29. **Carrozza, M. C., y otros.** *The development of a novel prosthetic hand – Ongoing research and preliminary results.* s.l. : IEEE/ASME Transactions on Mechatronics, 2002. págs. 108-114. Vol. 7.
30. The Cyberhand will be hard wired into the nervous system, allowing sensory feedback from hand to brain. [En línea] [Citado el: 29 de Noviembre de 2008.] http://www.medicalnewstoday.com.
31. **Martínez, A.** *Estudio y desarrollo de primitivas motoras para manipulación con manos robóticas antropomorfas.* [ed.] Doctor Fleming. S/N 30202.
32. **Romo, H. A., Realpe, J. y Jojoa, P. E.** *Análisis de Señales EMG Superficiales y su Aplicación en Control de Prótesis de Mano.* Electrónica Instrumentación y Control, Universidad del Cauca. Popayán : Universidad del Cauca, 2007. pág. 10, Proyecto de Investigación.

33. **Gaviria, C., Diaz, J. y Mosquera, V.** *Agarre estable de objetos con una prótesis de mano robótica.* Popayán : Universidad del Cauca, 2008. págs. 5-7.
34. **Vivas, A. y Aguilar, E.** *Modelado geométrico y dinámico de una prótesis de mano.* Cartagena, Colombia : III IEEE Colombian Workshop on Robotics and Automation, 2007.
35. **Khalil, W. y Dombre, E.** *Modeling, Identification and Control of Robots.* Londres : Hermes Penton Science, 2002.
36. **Muñoz A., M.** *Diseño de una prótesis de mano en un ambiente de diseño asistido por computador.* Popayán : Universidad del Cauca, 2007. págs. 25-30, Proyecto de Investigación.
37. **Van Den, G.** *Collision detection in interactive 3D environments.* San Francisco : Elsevier, 2004. págs. 67-72.
38. GNAT Programming Studio GPL Edition 2008. *AdaCore.* [En línea] http://libre.adacore.com.
39. **Fernández, A. y David, R.** *Arquitectura de Software.* s.l. : Universidad Tecmilenio, ITESM, 2000.
40. **Clements, P. y Kruchten, P.** *Documenting Software Architectures.* s.l. : Addison-Wesley, 2002.
41. GtkAda: a complete Ada graphical toolkit. *AdaCore.* [En línea] https://libre.adacore.com/GtkAda.
42. Glade - a User Interface Designer for GTK+ and GNOME. [En línea] http://glade.gnome.org.
43. **Briot, E., Brobecker, J. y Charlet, A.** GtkAda Reference Manual v.2.8. [En línea] 2006. https://libre.adacore.com/GtkAda/docs/2.8/gtkada_rm /gtkada_rm.html.
44. **Shreiner, D., Angel, E. y Shreiner, V.** *An Interactive Introduction to OpenGL Programming.* s.l. : SIGGRAPH, 2001.
45. **Charlet, A.** GtkAda Tutorial. [En línea] 2006. https://libre.adacore.com/GtkAda/GtkAda-tutorial/GtkAda-tutorial.htm.
46. **Briot, E., Brobecker, J. y Charlet, A.** GtkAda User Guide v.2.8. [En línea] 2006. https://libre.adacore.com/GtkAda/docs/2.8/gtkada_ug/gtkada_ug.html.
47. **Martz, P.** *OpenGL Distilled.* New York : Addison Wesley Professional, 2006.
48. **García, J.** Curso de introducción a OpenGL. [En línea] 2004. http://www.bardok.net.
49. **Ben-Ari, M.** *Ada for Software Engineers.* s.l. : Weizmann Institute of Science - John Wiley & Sons, 1998.
50. **Burns, A., Wellings, A.** *Sistemas de Tiempo Real y Lenguajes de Programación.* s.l. : Addison Wesley, 2003.
51. **Joseph, M.** *Real-time Systems Specification, Verification and Análisis.* s.l. : Prentice Hall International, 1996.
52. **Barnes, J.** *Programming in Ada 2005.* s.l. : Addison-Wesley, 2006.
53. **Diaz Martin, J. C.** La metodología HRT-HOOD. *Sistemas de Tiempo Real.* s.l. : Universidad de Extremadura - UEx, 1998.

54. **García Martín, J.** *Diseño de Sistemas de Tiempo Real HRT-HOOD.* s.l. : Universidad EUI, 2008.

55. **Wright, J. R.** *Programación en OpenGL.* s.l. : Anaya, 1997.

56. **Martinez, F. y Serrano, Fco.** *Creación de Entornos 3D con Open-GL y ECOSIMPRO: Aplicaciones a la Robótica.* s.l. : Universidad de Córdoba, 2001.

57. **Garcia, O. y Guevara, A.** Introducción a la Programación Gráfica con Opengl. [En línea] 2004. www.salle.url.edu/~oscarg/resources/openGLTutorialSpanish.pdf.

58. **Cesar, I.** Introducción a la computación gráfica. [En línea] 2008. http://educnet.decom-uv.cl/educnet/uploads/icg-ch-ogl-01.pdf?nombre=p370/icg-ch-ogl-01.pdf.

59. **Craig, J.J.** *Introduction to Robotics, Mechanics and Control.* s.l. : Addison-Wesley, 1986.

60. **Fu, González y Lee.** *Robótica: Control, detección, visión e inteligencia.* s.l. : McGraw_Hill, 1989.

61. **Shabana, A.** *Computational dynamics.* New York : John Wiley and Sons, 1995.

62. **Barrientos, A., y otros.** *Fundamentos de Robótica.* Madrid : McGraw-Hill, 1997.

63. **Segal, M. y Akeley, K.** *The OpenGl Graphic System: A Specification.* s.l. : Addison-Wesley, 2004.

64. **Prado D, Vidal.** *La estructura de la GRID Computing.* Monterrey, Mexico : Universidad Autónoma de Nuevo León, 2005.

65. **Delgado, J. J.** *Detección de Colisiones Mediante Recubrimientos Simpliciales.* Universidad de Granada. 2006. Tesis Doctoral. http://hera.ugr.es/tesisugr/16446665.pdf.