UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

SIMULADOR EN 3D DE ROBOTS MÓVILES

T E S I S
QUE PARA OBTENER EL GRADO DE:
INGENIERO EN COMPUTACIÓN

P R E S E N T A

MOISES ARIAS MORALES

DIRECTOR: DR. JESÚS SAVAGE CARMONA

CIUDAD UNIVERSITARIA, MÉXICO D.F., 2009

Agradecimientos

A Dios, por prestarme la vida necesaria para llegar hasta aquí.

A mis padres, por proporcionarme los medios necesarios para permitirme terminar una carrera.

A nuestra máxima casa de estudios la Universidad Nacional Autónoma de México, por el privilegio de permitirme ser parte de ella.

A Emmanuel Hernández Hernández, y Dr. Jesús Savage Carmona, por la confianza depositada en mi persona durante la elaboración de este trabajo.

A Laura Mata Montiel y toda la DGSCA, por haberme acogido durante todo este tiempo.

A mis amigos Roberto, Mauricio Héctor, Iván, Arturo, Lisette, Nidya, Jorge, José, Josafat, Sandra, Laura, Xavier y en especial a María Eugenia por toda su ayuda.

Índice general

Resumen ……..IX

Capítulo I. Antecedentes

1.1 Graficación por computadora ……1

1.2 Simuladores ……...2

1.3 Robots móviles ……..3

1.3.1 Definición ……3

1.3.2 Clasificación de robots ……..……..4

1.3.3 Como rontrolar el movimiento de un robot ……...……..5

1.4 Técnicas de control de render ……..….5

1.4.1 Modelos y transformaciones de vista ……..6

1.4.2 Iluminación ……6

1.4.3 Proyecciones ……....….8

1.4.4 Recorte ……..……..8

1.4.5 Imagen ……...……..9

1.5 Colisiones ……..…..11

1.6 Desarrollo multimedia ……...……..15

1.7 Librerías gráficas ……...……..16

1.7.1 OpenGL ……...……..17

1.7.2 Direct3D ……..……..18

1.8 Motores de física ……...……..18

1.9 Scripting ……..20

Capítulo II. Elección de las librerías de desarrollo

2.1 Introducción ……..……..23

2.2 Elección de la librería para la interfaz gráfica de usuario ……..24

2.3 Elección de la librería gráfica ……27

2.4 Elección de un motor de física ……....31

Capítulo III. Simulador

3.1. Introducción ……..……..33

3.2. Estructura interna del simulador ……..……..34

3.3. Ejecución de tareas del sistema ……..36

3.4. Ventanas y puntos de vista ……...…..40

3.5. Cámara libre ……..……..40

3.6. Cámara del robot ……..……..42

3.7. Cámara superior ……...……..43

3.8. Cámara lateral ……...…..43

3.9. Selección de objetos ……..……..44

Capitulo IV. Robots

4.1. Creación de robots ……...……..47

4.2. Estructura geométrica ……..……..48

4.2.1. Estructura lógica ……..….. 49

4.2.2. Movimiento del robot ……..49

4.3. Tipos de sensores ……..……..51

4.3.1. Sensores de contacto ……..……..51

4.3.2. Sensores reflectivos ……...……..56

4.3.3. Sensores infrarrojos ……...……..58

4.3.3.1. Emisor de luz infrarroja ……...……..59

4.3.3.2. Receptor de luz infrarroja ……..60

4.3.4. Sensores de ultrasonido ……..……..62

Capitulo V. Obstáculos

5.1. Introducción ……..……..65

5.2. Obstáculos tipo geometrías básicas ……..……..66

5.3. Obstáculos tipo modelos 3DS ……..……..66

5.4. Obstáculos tipo mapa ……...……..68

5.4.1. Formato de los mapas ……..68

5.4.2. Creación de elementos geométricos ……..…..69

5.5. Pistas ……..71

5.5.1. Creación de una pista ……..72

Capitulo VI. Pruebas al sistema ……..75

Capitulo VII. Conclusiones ……...……..81

Apéndice A Bibliotecas de enlace dinámico

A.1. Crear una librería dinámica para Windows ……..…..86

A.1.1 Crear un nuevo proyecto tipo librería DLL ……..……..86

A.1.2 Agregar código fuente ……..88

A.1.3 Crear el archivo de funciones de exportación ……..89

A.1.4 Crear la librería ……...……..89

A.2. Crear una librería dinámica para GNU/Linux ……..89

A.3 Mini manual de Makefile ……..…..92

A.4 Uso de variablesv …….……..93

Apéndice B. Manual del Usuario

B.1 Requerimientos mínimos ……..……..95

B.2 Instalación …….……..96

B.3 Entorno de sistema ……...……..97

B.4 Interacción con el sistema ……....…..99

B.5 Interacción con la ventana de inserción de objetos ……..108

B.6 Interacción con el sistema a través de las librerías de usuario ……..……..113

B.7 Robel ……...……..114

Referencias ……..……..117

Bibliografía ……...……..……..121

Resumen

En la actualidad existe una gran variedad de robots cuyos usos son muy variados, ya que pueden realizar tareas peligrosas, sucias, repetitivas o tareas en las que se requieren realizar operaciones de gran precisión.

Los robots se pueden clasificar en fijos y móviles, en donde los robots fijos como su nombre lo indica son todos aquellos que se encuentran fijos en una base y que pueden tener extremidades que les permiten realizar diversas funciones, estos se utilizan desde hace tiempo en las líneas de producción así como en la industria automotriz. Los robots móviles son robots que disponen de dispositivos electromecánicos que les permiten desplazarse por si mismos, con lo que pueden tener una mayor interacción con su entorno y que actualmente se han comenzado a utilizar en el hogar como robots de limpieza.

Gracias a los avances de la ciencia y tecnología, se han logrado grandes disminuciones en los tiempos de diseño y fabricación de robots, con lo que se ha visto una importante disminución en los costos de producción de estos. Por lo que, el uso de los robots ha crecido en distintos sectores como es el caso de la industria automotriz, pero a pesar de los

avances que se han hecho, estos no han sido suficientes para disminuir los altos costos de producción.

En el mercado existe una gran infinidad de simuladores de robots, que son utilizados tanto para el diseño y producción de robots como para realizar pruebas y simulaciones, con la finalidad de verificar el correcto funcionamiento de estos, probar nuevas configuraciones del robot o verificar el funcionamiento de los nuevos algoritmos de control.

Tomando como objetivo facilitar el desarrollo de pruebas de algoritmos de robots móviles, es que los exalumnos de la Facultad de Ingeniería de la UNAM, Emmanuel Hernández Hernández y Gabriel Vázquez Rodríguez desarrollaron el simulador Robot Command Center (ROC2), que permite simular desde uno hasta cinco robots móviles dentro de un espacio tridimensional, y cuyos robots cuentan con distintos tipos de sensores que les permiten obtener información del mundo virtual en el que se encuentran, también es posible controlar los robots mediante scripts del lenguaje Robel, o librerías externas programadas por el usuario, en las que se encuentran funciones que permiten controlar al robot que se este simulando.

Para desarrollar la interfaz grafica fue utilizada la API World Tool Kit (WTK), diseñada para el desarrollo de aplicaciones con gráficos 3D en tiempo real, donde requiere que los objetos tengan asociados comportamientos y propiedades reales. Para realizar la visualización de las imágenes fue utilizada la librería gráfica DirectX 8, todo esto bajo el lenguaje C.

Debido al lanzamiento de Windows Vista el cual sólo proporciona soporte para Directx 10, y al próximo lanzamiento de Windows 7, se vio la necesidad de desarrollar una nueva versión del simulador ROC2.

Por lo que el objetivo de esta tesis es desarrollar una nueva versión del simulador de robots móviles ROC2, que conserve todas las funcionalidades de la versión anterior pero que sea compatible con distintos sistemas operativos, como Windows Vista y las versiones anteriores de Windows e incluso futuras versiones de windows, así como sistemas operativos tipo Unix, como GNU/Linux y Mac OS.

Organización de la tesis.

La tesis se encuentra dividida en siete capítulos.

El capítulo 1 tiene como objetivo realizar una introducción a los conceptos manejados durante este trabajo.

El capítulo 2 muestra una pequeña descripción de las herramientas utilizadas para realizar este trabajo, en particular las librerías de desarrollo utilizadas.

El capítulo 3 explica el funcionamiento del simulador, con lo cual se pretende que se entienda el funcionamiento interno de este para realizar futuras actualizaciones.

El capítulo 4 muestra la lógica utilizada para realizar la definición de los robots que se utiliza para manipularlos.

En el capítulo 5 se describen los diferentes tipos de obstáculos utilizados y su lógica.

En el capítulo 6 se discuten las pruebas realizadas al sistema.

El capítulo 7 muestra las conclusiones obtenidas con base en las pruebas realizadas.

El simulador puede obtenerse de la sección de descargas de la página del laboratorio de Biorobotica de la Facultad de Ingeniería de la UNAM (http://biorobotics.fi-p.unam.mx/). En esta sección existen cuatro versiones disponibles del sistema, la versión anterior del sistema (Roc2) y tres distintas nuevas versiones del Roc2008 para sistemas GNU/Linux, Windows XP y Windows Vista respectivamente.

Capítulo I

Antecedentes

1.1 Graficación por computadora

La graficación por computadora es una rama de la computación que se encarga de la representación de cualquier tipo de información a través de un dispositivo electrónico en forma visual ayudado por una computadora, ya sea en dos dimensiones (2D) o en tres dimensiones (3D) [Fol96]. Las gráficas por computadoras han existido desde los inicios de la computación, ya que desde entonces se han realizado representaciones visuales de información (gráficos rudimentarios) en dispositivos de información como eran los teletipos [Ref01] e impresoras de línea [Wik01].

La graficación por computadoras es una de las ramas de la computación que se ha desarrollado con mayor rapidez desde los inicios de la computación, ya que desde que surgió la primera computadora se han necesitado dispositivos para visualizar la información

contenida en ellas, por lo que surgió la necesidad de construir dispositivos de salida como son teletipos e impresoras para visualizar la información.

Para poder realizar gráficos por computadora es necesario tener dos componentes que son el hardware [Ref02], medio encargado de procesar y mostrar la información (gráficas) y el software [Ref03] que es el encargado de manipular la información, mediante procesos que realiza el hardware para mostrar una representación visual que pueda entender el usuario final.

El hardware utilizado en la graficación por computadora no ha sufrido grandes cambios desde los primeros sistemas como el computador Whirlwind, desarrollado en el MIT en el año de 1950; éste tenía un tubo de rayos catódicos dirigido por computador que se empleaba para representar la información generada [Fol96].

Mientras que la evolución del hardware se realizaba paulatinamente, el software evolucionaba a pasos agigantados, forzando a los fabricantes de computadoras a desarrollar computadoras mas potentes para soportar la carga de trabajo producida por los primeros sistemas; como el sistema DAC de General Motors para el diseño de automóviles y el sistema Digitek de Itek para el diseño de lentes, éstos utilizan técnicas para automatizar la elaboración de bosquejos y otras actividades que requieren mucho trabajo de dibujo. Hasta llegar a los modernos sistemas de diseño asistido por computadora como AutoCAD, Solid Edge, SolidWorks [Ref04] [Hea95] [Fol96].

1.2 Simuladores

H. Maisel y G. Gnugnoli, definen la palabra simulación como:

“Simulación es una técnica numérica para realizar experimentos en una computadora digital. Estos experimentos involucran ciertos tipos de modelos matemáticos y lógicos que describen el comportamiento de sistemas de negocios económicos, sociales, biológicos, físicos o químicos a través de largos periodos de tiempo.” [Cos92]

Robert E. Shannon define simulación como:

“Simulación es el proceso de diseñar y desarrollar un modelo computarizado de un sistema o proceso y conducir experimentos con ese modelo con el propósito de entender el comportamiento del sistema o evaluar varias estrategias con las cuales se puede operar el sistema.” [Cos92].

Como se puede observar en las definiciones anteriores, ambos autores definen simulación como el análisis de un modelo mediante una computadora, y en términos generales nosotros definiremos simulación como:

Cualquier acción realizada para representar una situación, ya sea física, social, cultural, económica, ficticia, etc. Que puede ser representada por cualquier tipo de modelo

matemático, físico, heurístico, empírico o conceptual que puede ser analizado por medio de una computadora.

Como se puede observar con esta definición se puede abarcar cualquier tipo de situación que llegase o no a existir, ya que una simulación es un proceso por el cual se trata de dar respuesta a una pregunta.

En la actualizad existen una infinidad de simuladores, dentro de los cuales se pueden encontrar simuladores de carácter científico como los siguientes:

Mastercam: simula procesos de corte de materiales y maquinado como torno y fresa.

Unigraphics: simula esfuerzos y deformaciones en elementos de máquinas sujetos a esfuerzos.

Phoenix: simula fluidos en movimiento y transferencia de calor entre sustancias.

También existen simuladores de vida como:

The Sims: Es un juego que simula la vida de una familia (los Sims), cuya finalidad es satisfacer las necesidades de la familia.

Second Life: Simulador de vida virtual en línea, en el puedes crear una vida alterna, y cuenta con una economía virtual en la cual puedes realizar todo tipo de transacciones.

Simuladores de vuelo:

Microsoft Flight Simulator: Juego que proporciona una simulación realista del vuelo de aviones.

Simuladores de Robots:

EDISIM (Editor y simulador de robots manipuladores) crea, modifica y simula movimientos de robots manipuladores.

Microsoft Robotics Studio: simulador para la creación y depuración de aplicaciones robóticas, en donde la programación de los robot se puede realizar de forma visual a través de lenguajes como Visual C#, Visual Basic, JScript.

1.3 Robots móviles

1.3.1 Definición

El término robot tiene sus orígenes en la palabra “robota” que quiere decir “trabajo forzado” este término fue utilizado por primea vez en la obra de teatro Rossum's Universal

Robots (R.U.R) escrita por el checoslovaco Karen Capek en colaboración con su hermano Josef en el año 1920, la cual fue puesta en escena por primera vez en 1991.

Podemos definir a un robot como un dispositivo electromecánico construido para desarrollar una actividad específica, generalmente una serie de tareas repetitivas que abarcan manipulación de una infinidad de objetos y movimientos bajo control automático. Los robots no necesariamente tienen forma humana, pueden tener cualquier forma física. Se pueden clasificar dependiendo de cómo son controlados o por su forma [Her02].

1.3.2 Clasificación de los robots

Existen varias formas de clasificar a los robots, una de ellas puede ser por generaciones como se muestra a continuación:

Primera generación: Los robots están constituidos por dispositivos electromecánicos estacionarios, no programados y sin sensores [Ref05], la cual data de los años 70's.

Segunda generación: Los robots incluyen sensores y controladores programables, data de los años 80's.

Tercera generación: Estos robots cuentan con programación sofisticada, como reconocimiento y síntesis de voz, data de los años 90's.

Cuarta generación: Esta generación esta en la etapa de investigación e integra inteligencia artificial y robots nanométricos.

Dentro de las clasificaciones existen robots móviles y robots fijos. Los robots fijos como su nombre los menciona son los que se encuentran fijos a una base, éstos cuentan con extremidades para realizar tareas como los robots industriales, los cuales son utilizados principalmente en líneas de ensamblaje automotriz [Her02].

Los robots móviles como su nombre lo menciona son robots cuya fabricación les permite desplazarse por el ambiente en el que se encuentran, cuentan con dispositivos electromecánicos y sensores programables que les permiten conocer su medio ambiente, éstos pueden clasificarse en vehículos de ruedas dirigidas y de control diferencial. Los robots con configuración dirigida cuentan con una configuración de triciclo, esto es, un eje trasero y una rueda frontal. Los robots con configuración diferencial son de tipo tanque, lo que les permite realizar giros sobre su propio eje. Actualmente existen robots móviles que no utilizan ruedas, utilizan extremidades robóticas similares a las patas de los insectos, que les permiten moverse en terrenos no planos [Her02] [Mcc93].

El siguiente tema fue trascrito de [Her02].

1.3.3 Como controlar el movimiento del robot

La posición de un robot, regularmente está representada por tres parámetros: su posición (x, y) y su orientación, siempre y cuando se asuman movimientos planares. En un robot de tres ruedas sólo hay dos grados de libertad que son los correspondientes a los motores del eje trasero; controlar el movimiento de un robot móvil de esta configuración se limita a controlar los motores.

Los algoritmos de control de movimiento pueden ser divididos en dos partes: movimientos de trayectoria continua y movimientos punto a punto. El control de movimiento de trayectoria continua consiste en trasladar al robot sobre una línea continua, dicha línea no necesariamente tiene que ser recta. El tipo de movimiento punto a punto consiste en desplazar al robot desde su punto actual a otro punto, para ello se debe tener conocimiento de su posición en el mundo.

En general, hacer que un robot móvil se traslade sobre una recta requiere de muchos más cálculos que los movimientos punto a punto, pero el movimiento del robot es más suave cuando se mueve sobre una línea y por lo mismo puede ser más lento, facilitando de esta manera el control.

El principio para controlar un robot móvil se basa en el conocimiento de la posición del vehículo y su ambiente. Sin embargo, en la práctica estas dos características pueden ser desconocidas. Esta es la razón principal de contar con sensores. Los sensores recopilan información que puede ser procesada para obtener una posición estimada, detectar obstáculos y explorar el ambiente. Un robot puede contar con distintos tipos de sensores, los cuales se determinan por diversas razones como lo es la tecnología existente y el propósito del robot.

Los sensores pueden clasificarse en dos categorías: sensores de posición y sensores de ambiente. Los sensores de posición tienen como tarea proporcionar información al robot para que pueda estimar su posición actual. Los sensores de ambiente permiten al robot determinar las características del ambiente en el que se encuentra. Utilizando la información recopilada por sus sensores, un robot puede crear un mapa con las características del entorno en el que se encuentra, y con base en este mapa, puede planear sus movimientos para evitar obstáculos.

1.4 Técnicas de control de render

Render es el proceso de generar una imagen en dos dimensiones a partir de una escena tridimensional. Para poder generar un render es necesario realizar una serie de procesos por los cuales se tratará la escena para finalmente generar una imagen en dos dimensiones, a este proceso se le denomina render pipeline gráfica [Ref06]. A continuación podemos ver un diagrama del pipeline gráfico en la *Figura 1.1* [Bir07] [Ake91].

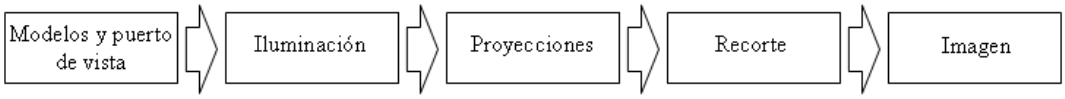


Figura 1.1 pipeline gráfico.

En la figura anterior se muestran los pasos para generar un render a partir de un modelo tridimensional, a continuación se explicarán éstos.

1.4.1 Modelos y transformaciones de vista

Un puerto de vista es un área en el dispositivo de salida (monitor) en el que se mapea una ventana que representa la vista de la cámara en un espacio tridimensional. Durante el proceso para generar una imagen en dos dimensiones a parir de una escena tridimensional los modelos se encuentran en un espacio de coordenadas mundial, al que se le debe de aplicar una serie de transformaciones para darles una orientación y posición con respecto a la cámara, y así poder visualizarlos en nuestro puerto de vista.

A este tipo de transformaciones se les denomina transformaciones de vista, son los cambios necesarios para que un espacio de coordenadas mundial se convierta al espacio de coordenadas de la cámara, donde podrá ser visualizado en el puerto de vista.

1.4.2 Iluminación

Para que una escena tridimensional resulte lo mas realista posible es necesario utilizar fuentes de iluminación.

En la vida real los objetos se ven afectados por una serie de fuentes de luz, dichos objetos pueden absorber o reflejar luz, dependiendo de la posición, orientación, forma del objeto y de las propiedades físicas del material con el que esté compuesto el objeto. Para obtener un efecto similar al que existe en la vida real, existen diferentes modelos de iluminación, con los cuales se puede generar el realismo deseado [Bir07].

Iluminación ambiental

Si una superficie no está expuesta a una fuente de luz, ésta será visible aún si los objetos circundantes están iluminados. En un modelo básico de iluminación se puede establecer un nivel general de brillantez para una escena, esta es una manera simple de modelar la combinación de varias superficies a fin de producir una iluminación uniforme que se conoce como luz ambiental [Bir07]. La ecuación de iluminación ambiental se puede expresar de la siguiente manera.

$$I = I_a K_a .$$

Donde I_a es la intensidad de la luz ambiental, que se supone constante para todos los objetos, la cantidad de luz ambiental reflejada por la superficie de un objeto está determinada por K_a, que es el coeficiente de reflexión ambiental, varía entre 0 y 1 y es una propiedad de los materiales.

Iluminación difusa

Con el modelo de iluminación ambiental se puede tener una escena iluminada con la misma intensidad de luz. En la vida real nunca sucede eso, los objetos se iluminan de diferente forma, dependiendo del tipo de material sobre el cual incide la luz, ésta se reflejará de diferente forma para cada material, por ejemplo, una superficie opaca refleja menos luz que una superficie lustrosa, así como una superficie rugosa refleja la luz de distinta forma que una superficie lisa.

La capacidad que tiene un material de reflejar parte de la luz en igual intensidad e igual dirección se le denomina reflexión difusa, la cual es una característica de los objetos mates como paredes, telas, papeles, tizas, etc [Bir07] [Fol96] [Hea95]. Para una superficie la brillantez depende únicamente del ángulo θ entre la dirección $\overline{L}$ a la fuente luminosa y la normal a la superficie $\overline{N}$, como se muestra en la *Figura 1.2*.

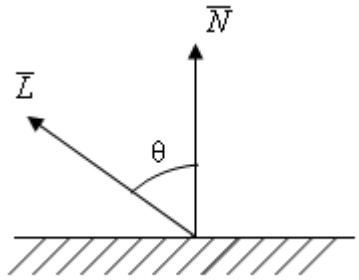


Figura 1.2 Diagrama de reflexión difusa.

De la figura 1.2 se obtiene la ecuación de iluminación de reflexión difusa:

$$I = I_p K_p \cos\theta.$$

Iluminación especular

Cuando se observa una superficie brillante iluminada, como metal pulido, una manzana o la frente de una persona, se ve un toque de luz o una mancha brillante en algunas direcciones de la vista. Este fenómeno se conoce como reflexión especular, es el resultado de la reflexión de la luz en la superficie que se observa [Bir07] [Fol96] [Hea95]. Un modelo muy utilizado para la representación de la luz especular es el modelo Phong [Ref07], desarrollado por Phong Bui-Tuong.

1.4.3 Proyecciones

Proyecciones de Perspectiva

En la vida real si se observa un objeto que se encuentra muy alejado, se podrá ver, dependiendo del tamaño y la distancia a la que se encuentre, pero mientras más cerca esté el objeto aumentará de tamaño, un sistema de proyecciones de perspectiva es simular a este efecto, como se muestra en la *Figura 1.3*.

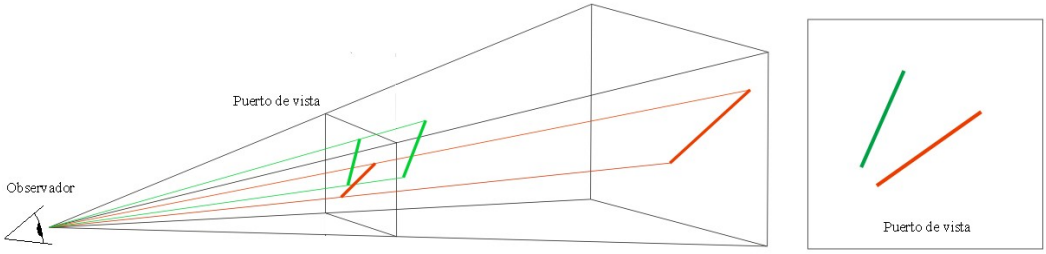


Figura 1.3 Proyección de Perspectiva.

Proyección Ortogonal

A diferencia de la proyección de perspectiva, las proyecciones ortogonales se hacen a través de un cubo, por lo tanto no se distingue la profundidad ni el volumen de los objetos, esto es porque un objeto no altera su tamaño en el puerto de vista, por ejemplo, una línea que se encuentra muy lejos se ve igual que a una que se encuentre próxima al observador. Como se muestra en la *Figura 1.4* [Hea95].

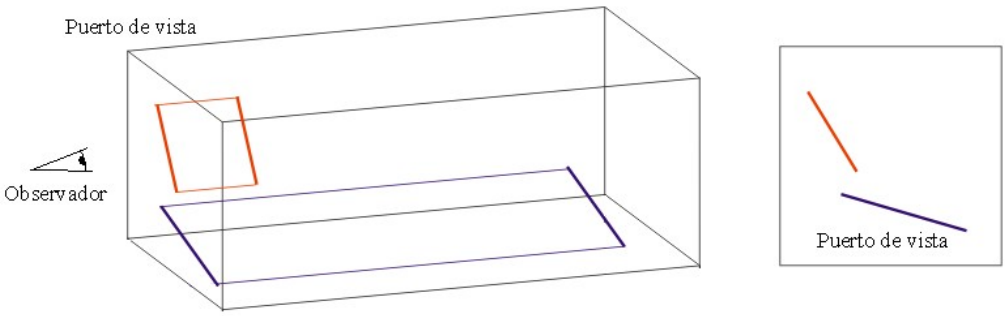


Figura 1.4 Proyección Ortogonal.

1.4.4 Recorte

Una cámara sólo es capaz de ver lo que se encuentra en su campo de vista, por lo que hay que identificar que objetos no se encuentran totalmente dentro del campo de vista de la cámara, para eliminarlos se les aplicará un algoritmo de recorte, entonces no es necesario que se dibujen, como se muestra a continuación.

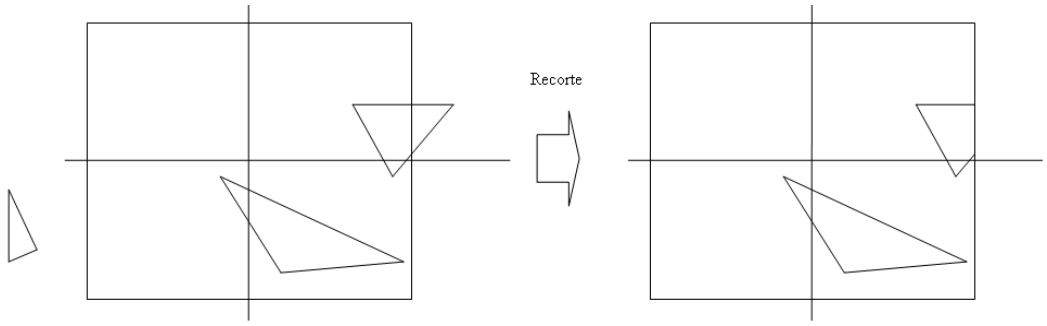


Figura 1.5 Operación de Recorte [Ref08].

En la *Figura 1.5* sólo existe un triángulo que se encuentra totalmente dentro del puerto de vista, por lo tanto el triángulo de la izquierda no es necesario dibujarlo, en cambio el triángulo que se encuentra a la derecha está parcialmente dentro del área de visualización, por lo que se deberá aplicar un algoritmo de recorte para eliminar las partes que no son visibles.

1.4.5 Imagen

Una vez realizados estos pasos se pueden aplicar otros métodos para obtener una imagen más nítida, como son los siguientes.

Sobre Muestreo (over-samplig)

Implica la recolección de más datos de los necesarios. Cuando un programa de render sobre muestrea una escena recoge más puntos o rayos de los que necesita para el número de pixeles [Ref09] que se emplearan en la imagen para obtener una mejor definición.

Render de alta resolución

Alternativa para obtener un mejor anti-aliasing [Ref10], consiste en generar una imagen con una resolución más alta, por ejemplo, si la imagen final tiene que ser de 720 pixeles de ancho es posible procesar a 1440 pixeles, usando un múltiplo de 200 ó 400 %, con lo cual se obtiene una imagen más suave, después de editar la toma, se escala la imagen para obtener el tamaño deseado.

Algoritmo REYES

Es el núcleo sobre el que trabaja RenderMan [Ref11] de Pixar [Ref12], se considera como el render de películas más importante [Bir07], fue diseñado para posibilitar un nivel de detalle elevado con superficies curvas y suaves, mapeando una superficie a nivel de píxel, evitando el desenfoque de movimiento y la profundidad de campo.

El algoritmo Reyes es capaz de procesar superficies curvas como las representadas por funciones NURBS [Ref13] o subdivisiones de superficies, dividiéndolos en micro polígonos, pequeños cuadriláteros del tamaño de un píxel e incluso más pequeños.

Render z-buffer

Generalmente es utilizado por las tarjetas gráficas para cálculos de tiempo real y render de prueba de animaciones.

Uno de los principales retos en los inicios de los gráficos 3D era encontrar algoritmos efectivos para eliminar superficies ocultas, esto consistía en asegurar que las superficies del primer plano aparecieran delante de la superficie que está detrás.

El render z-buffer es una solución para eliminar las superficies ocultas en tiempo real que permite que los objetos se dibujen con rapidez, independientemente del orden en el que se procesan sin que influya el ángulo de la cámara [Bir07]. El z-buffer es un área de la memoria que almacena la profundidad de la imagen.

Cuando se dibuja el primer polígono se añaden pixeles de color a un buffer de marco y se almacenan la información de profundidad en el z-buffer incluyendo la distancia de la cámara a cada punto del polígono que se está dibujando, cuando se dibuja otro polígono se compara la profundidad del z-buffer con la profundidad en cada píxel, si un polígono está más cerca de la cámara que alguno de los polígonos ya dibujados en esa área se trae el buffer de marco, si alguna parte del polígono está más alejado de la cámara, queda oculta basándose en una comparación entre la profundidad y el valor existente en z-buffer.

Render Scanline

Es un proceso que trabaja a nivel de píxel en lugar de polígono por polígono, una vez que se ha producido una línea horizontal de píxel (o scanline), el programa continua con la siguiente línea hasta que se completa la imagen.

Render por hardware

Se refiere al uso de las imágenes generadas por la tarjeta de gráficos como resultado final del proceso de render. Las tarjetas están diseñadas para generar imágenes en tiempo real para gráficos interactivos. Cuando se liberan de la limitación de generar 30 cuadros por segundo, son capaces de crear imágenes con una calidad más alta que la que se muestra en un video juego.

1.5 Colisiones

La detección de colisiones es una parte fundamental en los sistemas de computación gráfica, como en aplicaciones de realidad virtual, CAD/CAM, animación por computadora, modelado de sistemas físicos, simuladores científicos, simuladores de robots, y actualmente ha tomado una gran importancia en el desarrollo de juegos [Rui08].

La detección de colisiones verifica si dos objetos han chocado, ya sea por medio de ecuaciones u operaciones lógicas. La detección de colisiones se puede realizar en dos o tres dimensiones, en dos dimensiones las operaciones que se tienen que realizar son más simples mientras que la detección de colisiones en tres dimensiones resulta más compleja. Por lo que examinaremos algunos casos.

Para el caso de dos dimensiones se analizara un ejemplo de detección de colisiones en dos dimensiones de círculos con círculos, como se muestra a continuación.

Se dice que dos objetos colisionan cuando ocupan el mismo espacio o área, este caso se presenta cuando la distancia entre los centros de las circunferencias es menor o igual a la suma de los radios de éstas, como se muestra a continuación:

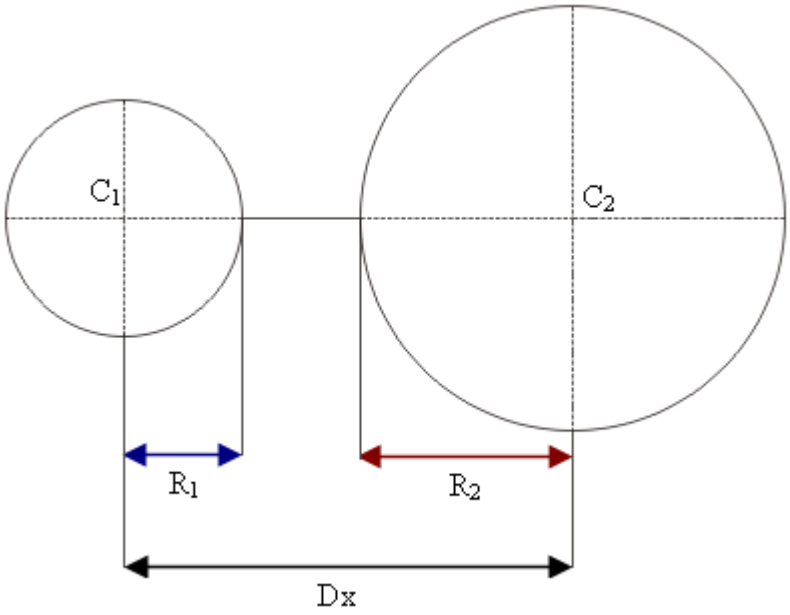
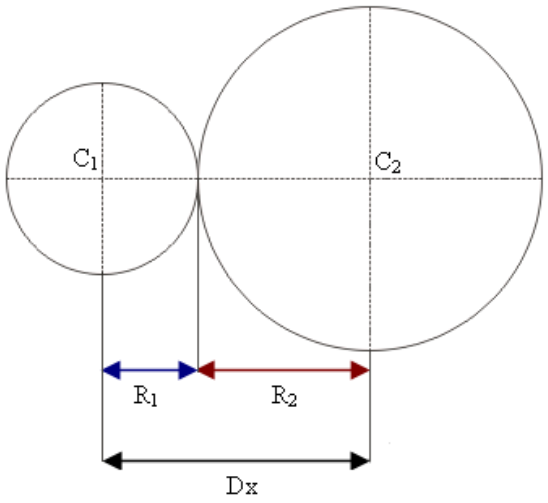


Figura 1.6 Ejemplo 1 de colisiones [Ref14].

En la *Figura 1.6* se pueden observar dos circunferencias con distintos radios R_1 y R_2, y centro C_1 y C_2 respectivamente, cuyos centros se encuentran separados uno del otro a una distancia Dx, como se muestra en la siguiente ecuación:

$$Dx > R_1 + R_2.$$

Si la expresión anterior se cumple podemos decir que las circunferencias no colisionan ya que la distancia Dx es mayor que la suma de los radios R_1 y R_2.
Ahora observemos los siguientes ejemplos



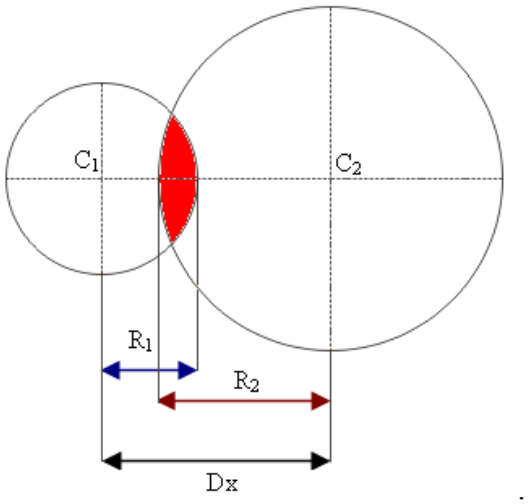


Figura 1.7 Ejemplo de colisiones 2. Figura 1.8 Ejemplo de colisiones 3.

En la *Figura 1.7* [Ref14] y *Figura 1.8* [Ref14] se puede observar las mismas circunferencias en dos situaciones de colisión, para verificar si dos circunferencias colisionan se tiene la siguiente expresión:

$$Dx \leq R_1 + R_2 .$$

Las circunferencias colisionan si Dx es menor o igual a la suma de los radios R_1 y R_2, de igual manera si la suma de los radios R_1 y R_2 es mayor o igual a la distancia Dx.

Para el caso en tres dimensiones: colisiones de esferas con esferas, la detección de colisiones se aplica de igual forma, lo único que cambiará será el cálculo de la distancia entre los radios ya que esta se deberá de realizar para puntos en el espacio.

Una vez definida la colisión entre esferas se puede ver la colisión entre cajas en el espacio de tres dimensiones. Para poder explicarlo con más detalle observe la siguiente figura.

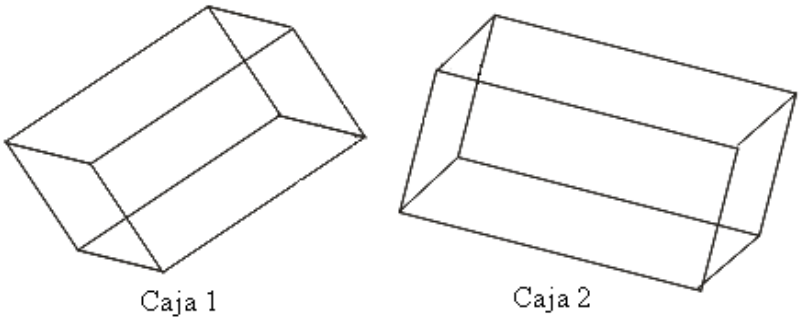


Figura 1.9 Ejemplo de colisiones 4.

En la *Figura 1.9* se pueden observar dos cajas distintas en el espacio, y que no existe colisión entre ellas. Esto es porque no existe contacto entre ellas. En la siguiente figura se observa qué sucede cuando colisionan.

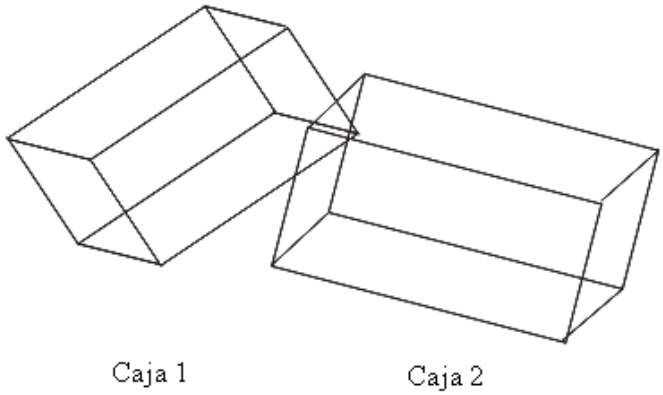


Figura 1.10 Ejemplo de colisiones 5

En la *Figura 1.10* es fácil observar que existe una colisión entre ambas cajas, esto se debe a que el ser humano cuenta con un nivel de abstracción superior al de una computadora, y con sólo observar la figura se puede saber si existe una colisión, pero para una computadora que no cuenta con la capacidad de abstracción no resulta tan fácil.

Para ello es necesario idear un método para detectar colisiones por medio de expresiones matemáticas y lógicas, como el que se muestra a continuación:

Se define un centro para cada caja, se dibujan líneas desde cada vértice al centro de la caja con la que se comprueba si existe colisión o intersección entre las líneas y las caras de la caja, con lo anterior se tiene un método para verificar colisiones entre cajas en el espacio. Mostrado en la siguiente figura.

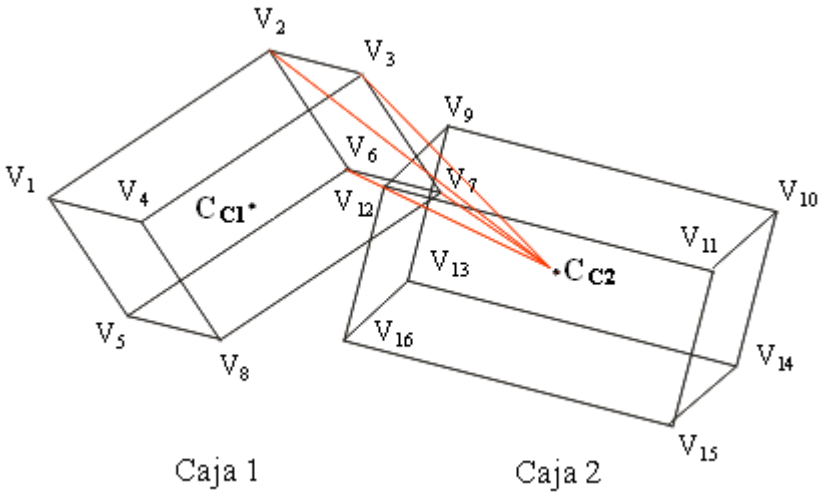


Figura 1.11 Ejemplo de colisiones 6.

En la *Figura 1.11* se puede observar la Caja 1 formada por los vértices V_1, V_2, V_3, V_4, V_5, V_6, V_7 y V_8, y el centro C_{C1}, y la Caja 2 con centro C_{C2} y vértices V_9, V_{10}, V_{11}, V_{12}, V_{13}, V_{14}, V_{15} y V_6. La detección de colisiones entre cajas se reduce a la detección de intersecciones entre el plano formado por los vértices V_9, V_{12}, V_{16} y V_{13}, y las líneas formadas por los vértices V_1 y C_{C2}, V_2 y C_{C2}, V_3 y C_{C2}, V_4 y C_{C2}, V_5 y C_{C2}, V_6 y C_{C2}, V_7 y C_{C2}, etc., respectivamente. Como se muestra en la figura.

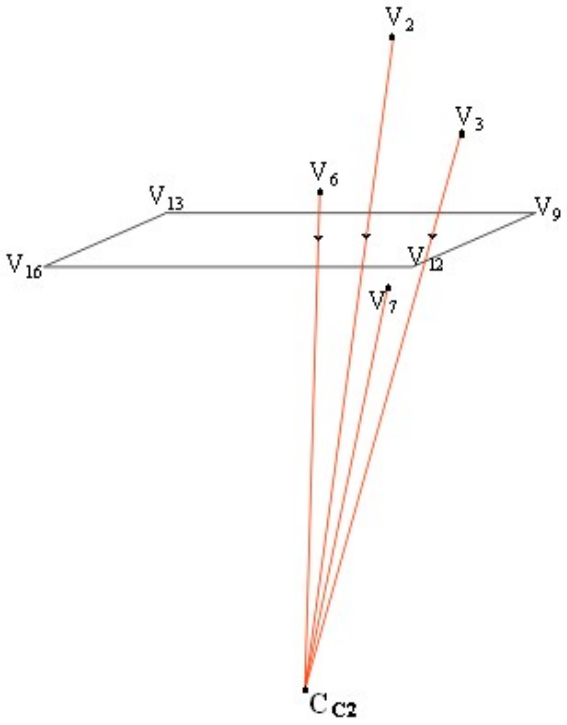


Figura 1.12 Detección de intersección entre líneas contra plano.

Como podemos observar en la *Figura 1.12* las líneas V_6 y C_{C2}, V_2 y C_{C2}, V_3 y C_{C2}, tienen una intersección con el plano formado con los vértices V_9, V_{12}, V_{16} y V_{13}, mientras que la línea formada por los vértices V_7 y C_{C2} no tiene intersección con el plano. Por lo que se dice que la Caja 1 colisiona con la Caja 2.

Para verificar si existe alguna intersección del plano con cualquier línea se tiene lo siguiente.

La ecuación de la línea se puede representar de la siguiente manera:

$$X = X_0 + X_d * t,$$
$$Y = Y_0 + Y_d * t,$$
$$Z = Z_0 + Z_d * t.$$

Donde:

$P_0 = [X_0, Y_0, Z_0]$ Punto de origen de la línea.

$V_d = [X_d, Y_d, Z_d]$ Vector director de la línea.

La ecuación general del plano es:

$$AX + BY + CZ + D = 0.$$

Donde:

$V_n = [A, B, C]$ Vector normal al plano.

$D = -(Ax_1 + By_1 + Cz_1)$ Constante.

$P_1 = [x_1, y_1, z_1]$ Punto sobre el plano.

Para encontrar la intersección entre la línea y el plano se sustituyen las ecuaciones de la línea en el plano, como se muestra a continuación.

$$A(X_0 + X_d * t) + B(Y_0 + Y_d * t) + C(Z_0 + Z_d * t) + D = 0 .$$

Despejando la variable t se tiene:

$$t = \frac{-(AX_0 + BY + CZ_0 + D)}{AX_d + BY_d + CZ_d}.$$

Simplificando la expresión:

$$t = \frac{-(V_n \bullet P_0 + D)}{V_n \bullet V_d}.$$

Por lo que la detección de colisiones entre cajas en el espacio se reduce a verificar la existencia de la variable t para cada línea formada por cada vértice de la Caja 1 con el centro de la Caja 2 y cada plano de la Caja 2.

1.6 Desarrollo multimedia

El término multimedia se refiere al uso de múltiples tipos de información de texto, gráficos, sonido, imágenes, videos, etc., que se muestran a través de cualquier medio electrónico [Ins97], y para que se consideren como una tecnología multimedia se deben cumplir tres aspectos importantes:

Encontrarse integrados coherentemente.
Proporcionar información a los usuarios en tiempo real.
Permitir la interacción del usuario con los elementos.

Las aplicaciones multimedia además de contener archivos de texto, gráficos, imágenes, sonido, video etc., deben ser presentadas en una forma armónica y lógica.

Una vez definido el término multimedia surge la interrogante ¿Qué es desarrollo multimedia?. El desarrollo multimedia se refiere a la acción de desarrollar aplicaciones (software) que utilicen multimedia para representar información que el usuario solicita, ya

sea por medio de sonido, imágenes, video, gráficos, animación etc., y que permiten una interacción del usuario con la aplicación [Fra94].

Para el desarrollo de aplicaciones multimedia existen una infinidad de librerías de desarrollo, que nos permiten incorporar multimedia, como:

Visual Studio: Es un entorno de desarrollo integrado (IDE) para las plataformas Microsoft Windows, facilita el desarrollo de aplicaciones que contienen multimedia a través de diferentes lenguajes de programación como son: Visual c++, Visual C#, Visual J#, ASP, Visual Basic. Integrando herramientas que pueden interactuar con audio, video, imágenes, gráficos, animación, etc.

GTK+: Significa GIMP Toolkit (Conjunto de herramientas GIMP), GIMP (Graphical Image Manipulation o Manipulación de Imágenes Gráficas), es una biblioteca utilizada para desarrollar aplicaciones que tengan una interfaz gráfica de usuario. GTK+ hoy en día es utilizada ampliamente para desarrollar aplicaciones multiplataforma ya que proporciona herramientas para la interacción del usuario con elementos de multimedia [Har99].

1.7 Librerías gráficas

La interfaz entre una aplicación y un sistema gráfico se puede especificar como un conjunto de funciones que residen en una librería gráfica, la cual es llamada Interfaz de Programación de Aplicaciones (API). El programa de aplicación puede ser modelado como se muestra en la *Figura 1.13* [Hea95].

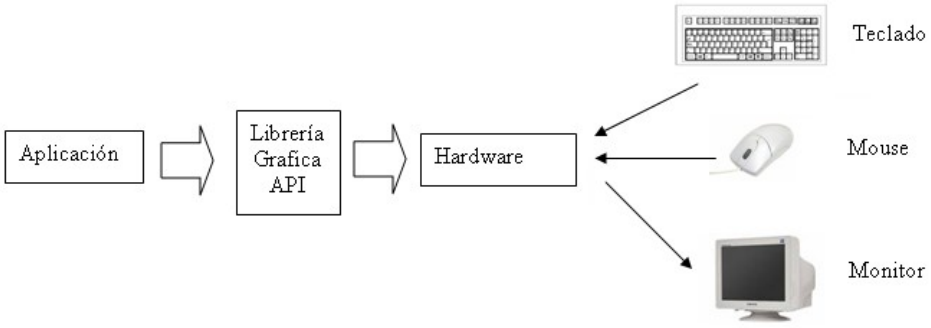


Figura 1.13 Representación del API.

La aplicación únicamente puede ver la API, no puede ver los detalles del hardware. La comunicación entre el hardware y la aplicación se hace a través de la API, por lo que las funciones disponibles de ésta deben combinarse con el hardware (tarjeta de video), para que los usuarios generen imágenes. El modelo cámara-sintética [Ref15] es la base para varios API's populares incluyendo OpenGL, PHIGS, DirectX, VRML y Java-3D [Ref16].

1.7.1 OpenGL

En el año de 1982 nació en la Universidad de Stanford el concepto de "Graphic Machine" el cual fue utilizado por Silicon Graphics (SGI) en su estación de trabajo Silicon IRIS para crear un renderiador con lo cual nació la librería gráfica GL, basada en una implementación del modelo cámara-sintética con una arquitectura pipeline. GL fue diseñado especialmente para el render de alta velocidad y en tiempo real. Debido a la popularidad de esta librería varias empresas se pusieron de acuerdo para desarrollar en conjunto una librería gráfica libre (o OpenGL por sus siglas en ingles). Entre esas empresas se encontraban SGI, Microsoft, IBM, Sun Microsystems, Digital Equipment Corporation (DEC), Hewlett-Packard, Intel [Ser08] [Ang00].

Debido a que se removieron funciones para la generación de ventanas y manejo de dispositivos de entrada enfocándose en los aspectos de render de la API. OpenGL emergió como una nueva API que es portable pero que retiene las características que hacían a GL una poderosa API. La característica de ser una librería open source [Ref17] garantizaba que un programa escrito para una plataforma podía ser fácilmente transportado a cualquier plataforma, obteniendo los mismos resultados, esta era la principal novedad OpenGL lo cual liberaba a los programadores de escribir un programa para un hardware específico, a continuación se muestran algunas implementaciones de esta librería.

OepnGL32: Implementación de Microsoft que se comenzó a incluir en Windows NT 4.0

SGI OpenGL para Windows: implementación de Silicon Graphics para Windows

3DFXGL: Se basa en la implementación OpenGL 1.1.

Mesa3D: Aunque es una librería gráfica basada en OpenGL pero de código abierto, no ha podido obtener la licencia de OpenGL de SGI, por lo tanto no se le puede llamar una implementación.

Debido a que OpenGL esta dedicada única y exclusivamente al render, no dispone de funciones adicionales para la simplificación de operaciones por lo que han surgido las siguientes librerías [Ser08]:

GLU: Incluye funciones para definir cilindros, discos y para trabajar con splines [Ref18] y matrices.

GLUT: La principal característica es que implementa funciones para la creación y manejo de ventanas, teclado y ratón.

GLAUX: Es la implementación de Microsoft para Windows de la librería GLUT

Entre otras.

1.7.2 Direct3D

Servan Keondjian fundó RenderMorphics, una compañía que desarrollaba una API de gráficos 3D llamada Reality Lab. Esta API se usaba en programas de CAD y representación de imágenes médicas. En el año de 1995 Microsoft compró RenderMorphics, incorporando a Keond jian a la compañía para implementar un motor gráfico para Windows 95. El resultado fue la primera versión de Direct3D [Wik02].

"Inicialmente, Direct3D se implementó sobre dos APIs: la *etained mode* y la *inmediate mode*. El modo retenido era una API de escenarios gráficos basada en el COM (*Computer Object Model*) de Microsoft, que tuvo escasa acogida. Los desarrolladores de juegos solicitaron un control más directo sobre las actividades del hardware del permitido en el *retained mode*. Solamente el juego *Lego Island* se basó en dicha API, por lo que Microsoft abandonó la evolución de dicho modo" [Wik02]

"La primera versión del modo inmediato de Direct3D consistía en un modelo de programación basado en un buffer de ejecución. Microsoft confiaba en que dicho buffer fuera soportado directamente por los vendedores de hardware pretendiendo que se almacenaran en memoria y fueran parseados [Ref19] por hardware, con el objetivo de realizar renderización 3D. Dichos buffers resultaron ser muy difíciles de programar, frenando la adopción de la nueva API y generando una corriente de opinión que solicitaba la adopción de OpenGL como la oficial para renderización 3D en Microsoft. Microsoft decidió seguir mejorando Direct3D, no solo para ser competitivos con OpenGL, sino también para competir de forma más efectiva contra otras APIs propietarias como Glide de 3dfx. Se encargó a un equipo de Redmond hacerse cargo del desarrollo del modo inmediato de Direct3D, mientras el equipo de RenderMorphics continuaba el trabajo sobre el modo retenido" [Wik02].

Direct3D se encuentra incluida dentro de la plataformas Microsoft Windows desde la versión Windows 95, también se encuentra dentro de SDK (Kit de Desarrollo de Software) que es un conjunto de herramientas que facilitan el desarrollo de software, en este caso el SDK DirectX esta enfocado al desarrollo de videojuegos, Microsoft lo distribuye gratuitamente.

Entre las principales características del SDK es que proporciona herramientas para manipular dispositivos de entrada como son ratón, teclado, cámaras, y dispositivos de juego, también incorpora herramientas que manipulan el audio. Actualmente Direct3D se encuentra en su versión 10 que esta únicamente disponible para Windows Vista.

1.8 Motor de física

Un motor de física (physics Engine) es un software diseñado para simular cuerpos rígidos por medio de la implementación de física newtoniana, asignando propiedades físicas a los objetos que se desean simular, como: masa, velocidad, fricción, etc. [Ram08] Con esto se puede predecir como reaccionará un objeto en diferentes condiciones al aplicarle gravedad y fuerzas externas, el principal uso esta en la de simulación científica y video juegos.

Los motores físicos se pueden clasificar en motores en tiempo real y motores de alta precisión o dinámicos, donde los motores de alta precisión son utilizados en simulaciones científicas y en películas animadas, ya que es en estas áreas donde se requiere que la simulación sea lo más precisa posible, ya que son aplicaciones que manejan grandes cantidades de datos y realizan gran cantidad de operaciones, por lo que es necesario computadoras que puedan manejar esa cantidad de procesos [Anm08]. Mientras que los simuladores de tiempo real son utilizados en los video juegos, ya que se requiere disminuir la calidad de la simulación para aumentar la velocidad de ésta.

Los motores de física están conformados principalmente por dos componentes, un sistema de detección de colisiones y un sistema de física responsable de aplicar las fuerzas a los objetos y generar las reacciones de éstos.

Los motores en tiempo real son los utilizados principalmente en video juegos, donde la velocidad de simulación es más importante que la exactitud, es por esto, que generalmente los modelos tridimensionales están representados por dos tipos de objetos, el primero que representa el modelo 3D en el mundo y es el que se muestra al usuario, por lo que este modelo está conformado generalmente por una gran cantidad de polígonos, para obtener modelos lo más realista posible [Anm08]. El segundo objeto llamada comúnmente como malla o geometría de colisión, que puede ser una simplificación del objeto tridimensional o bien una representación de éste por medio de geometría básica como cilindros, conos, cajas, esferas. Que son utilizadas para la detección de colisiones y aplicación de la física, con esto se busca disminuir el número de posibles colisiones entre objetos y así aumentar el rendimiento de la simulación, pero disminuyendo el realismo de la misma.

La principal limitación de los motores de tiempo real es el tipo de datos que utilizan para representar fuerzas y vectores, ya que al tener una limitación en el almacenamiento de la información los datos se tienen que redondear, lo cual produce errores de precisión, ya sea en la posición de los objetos o en la cantidad de fuerza que se le aplica a un objeto [Anm08]. Estos redondeos de la información pueden llegar a producir una acumulación de energía en el objeto, produciendo después de un cierto tiempo que el objeto reaccione diferente ha lo esperando, produciendo efectos como el de que el objeto comience a temblar sin razón aparente.

Otro problema que presenta los motores de tiempo real es en el calculo de los frames [Ref20], debido a que estos se generan cada cierto periodo de tiempo, lo cual quiere decir que la representación del tiempo no es continuo, y la simulación solo se obtiene para ciertos instantes, así que si no se escoge un tiempo suficientemente pequeño entre cada calculo de frame y frame se pueden obtenerse una simulación errónea. Y a pesar de que hayamos escogido un intervalo de tiempo lo suficientemente pequeño se pueden presentar efectos no deseados, como en el caso de objetos que se mueven a grandes velocidades, ya que debido a esto puede que se presente un efecto de tele transportación ya que el tiempo entre el calculo de frame y frame no es lo suficiente mente pequeño para obtener una secuencia continua del desplazamiento de dicho objeto.

Actualmente existen varios motores físicos comerciales y de distribución libre, entre los más conocidos se encuentran los siguientes:

Open Dynamic Engine (ODE): Motor de físico con licencia BSD [Ref21] y diseñado por Russell Smith. Entre las principales características de este motor físico es que es multiplataforma e incorpora simulación de cuerpo rígido. Diferentes tipos de joins (uniones entre diferentes objetos) como bola y Socket, bisagra, deslizables, uniones universales en las que uno puede indicar los grados de libertad, también incorpora fricción y colisiones, así como un sistema de detección de colisiones independiente. Una desventaja es que no incorpora detección de colisiones de cilindros con cilindros.

PhysX: Motor físico con licencia EULA [Ref22] y diseñado por AGEIA: entre las principales características de este motor físico es que incorpora simulación de cuerpo rígido así como simulación de cuerpo blando, fricción, joins (uniones entre diferentes objetos), simulación de partículas e implementación de sensores entre otras.

Bullet Physics: Motor de físico con licencia ZLib [Ref23]. Proyecto dirigido por Erwin Coumans: entre las principales características es que es una librería multi-hilos y multiplataforma dentro de las cuales se encuentran PlayStation 2 y 3, XBox 360, Nintendo Wii. Principal desventaja, la documentación es casi nula.

Tokamak: Motor de física con licencia BSD y desarrollada por David Lam: entre las principales caracterizas es que es una librería muy estable y fácil de utilizar. La principal desventaja es que solo da soporte para la plataforma Windows y no se le ha dado soporte a la librería desde 2004.

Newton Game Dynamic: Motor de física con licencia Copyright [Ref24] y desarrollada por Julio Jerez y Alain Suero: entre las principales características son que es multiplataforma, incorpora colisiones entre cilindros, fricción con diferentes tipos de materiales colisiones de cilindros con cilindros. Desventajas no cuenta con mucha documentación.

1.9 Scripting

Un script (guión) es una secuencia de una o más instrucciones que pueden ser ejecutados en un contexto sin la necesidad de una previa compilación. Lo que quiere decir que es un programa que fue escrito utilizado un lenguaje interpretado [Char05]. Por lo que la palabra scripting se refiere a la elaboración de un programa utilizando un lenguaje interpretado.

Para que un programador pueda desarrollar una aplicación (software) tiene que utilizar un lenguaje de programación, esto se debe a que el lenguaje que utiliza el ser humano es de un nivel de abstracción muy superior al que una computadora pudiera llegar a utilizar y entender, por lo que es necesario utilizar otro lenguaje para comunicarnos con las computadoras.

Existen una infinidad de lenguajes de programación que se pueden utilizar para el desarrollo de aplicaciones, así como diferentes niveles de lenguajes de programación, y cuando nos referimos a niveles de lenguajes de programación, quiere decir que mientras el lenguaje de programación utilizado para comunicarnos con el ordenador tiene un grado de abstracción mayor (se asemeje mas a nuestro idioma) implicara que el lenguaje es de un

mayor nivel. Lo cual quiere decir que para indicarle al ordenador que realice una acción se requerían un menor número de instrucciones.

El lenguaje que utilizan las computadoras resulta ser demasiado analítico y complejo para los seres humanos, por lo que para indicarle a una computadora que realice una acción, el número de instrucciones utilizado es mayor debido a que el nivel de abstracción que utilizan las computadoras es menor a nuestro idioma, así que es necesario utilizar un mayor numero de instrucciones para comunícanos con la computadora. Entonces se dice que el lenguaje de programación es de bajo nivel.

Una vez que el programador escogió un lenguaje para comunicarse con la computadora, es necesario utilizar un programa encargado de traducir el lenguaje de programación a un lenguaje que la computadora pueda entender (lenguaje maquina compuesto de ceros y unos), y que generalmente solo se puede realizar en un solo sentido del lenguaje de programación al lenguaje maquina, y esto se debe a que son lenguajes muy complejos y resulta muy difícil realizar la operación inversa de lenguaje maquina al lenguaje de programación.

Los lenguaje de programación también reciben el nombre de lenguajes compilados o interpretados, dependiendo del tipo de programa que utilicen para traducir el lenguaje de programación al lenguaje maquina. Como su nombre lo dice los lenguajes compilados utilizan un compilador para traducir el programa, mientras que los lenguajes interpretados utilizan un intérprete, para realizar dicha operación.

Los compiladotes traducen el programa completo a otro lenguaje (lenguaje maquina) que la computadora es capas de interpretar, a este proceso se le conoce como compilación. El computador genera un archivo denominado ejecutable que puede ser utilizado todas las veces que se desee, así que si se quiera utilizar el programa ya no será necesario utilizar el compilador para generar nuevamente el ejecutable.

Los interpretes toman el código escrito en x lenguaje, traducen la primera instrucción para obtener un código maquina que la computador pede entender y permite que la computadora la ejecute, acto seguido traduce la siguiente instrucción a código maquina y permite que la computadora la ejecute, repitiendo el proceso hasta llegar al final del programa. Lo que quiere decir que cada vez que se quiera ejecutar el programa se tendrá que hacer uso del intérprete, por lo que aquí el intérprete no genera ningún archivo ejecutable.

Entre los lenguajes compilados se encuentran: Fortran, C, C++, Ada, Pascal, Delphi, algol, entre otros.
Entre los lenguajes interpretados o de script se encuentran: Perl, PHP, Java, Javascript, Logo, ASP, Pitón, Robel entre otros.

Entre las ventajas y desventajas de un lenguaje compilado se encuentran las siguientes:

La principal ventaja de los programas compilados es la velocidad, es verdad que al principio puede ser muy tardada la compilación, pero una vez obtenido el programa compilado, solo es necesario ejecutar este las veces que se desee.

Otra ventaja es el manejo de errores debido a que el compilador verifica primero toda la estructura del programa, y muestra todos los errores que encuentra.

Una desventaja de los lenguajes compilados es que es necesario tener un compilador para cada plataforma en la que se desee utilizar el programa.

Entre las ventajas y desventajas de un lenguaje interpretado se encuentran las siguientes:

Una ventaja es que el programa se puede depurar de una forma más rápida y sencilla.

Otra ventaja es que no es necesario tener un compilador para cada plataforma en la que se desee utilizar el programa.

Una desventaja seria la velocidad, ya que al no generarse un archivo ejecutable, cada vez que se desee utilizar el programa este tendrá que ser interpretado.

Capítulo II

Elección de las librerías de desarrollo

2.1 Introducción

El objetivo de esta tesis es realizar una actualización del simulador ROC2 obtenido como resultado de la tesis “INTERFAZ GRÁFICA 3D PARA SIMULAR Y CONTROLAR ROBOTS MÓVILES USANDO REALIDAD VIRTUAL” presentada por Emmanuel Hernández Hernández y Gabriel Vázquez Rodríguez. Que esté disponible para sistemas operativos tipo win32, como sistemas operativos tipo Unix, Por lo que se decide utilizar librerías de programación multiplataforma que sean compatibles con el lenguaje de programación C, ya que éste es el lenguaje que se escogió para realizar la implementación de la nueva versión.

2.2 Elección de la librería para la interfaz gráfica de usuario

GTK+ o The GIMP Toolkit [Ref24] es un conjunto de herramientas para el desarrollo de interfaces gráficas de usuario (GUI), se encuentra bajo la licencia LGPL [Ref25]. Originalmente fue desarrollada por Peter Mattis, Spencer Kimball y Josh MacDonald para el sistema de ventanas X Windows y Gimp, actualmente es utilizada para el desarrollo de una infinidad de aplicaciones que requieren de una interfaz gráfica de usuario. Es una librería orientada a objetos escrita en C que es soportada en lenguajes como C++, C#, java, Python, Perl, PHP, etc., y se encuentra disponible para las siguientes plataformas. [Har99].

- GNU/Linux y Unix
- Windows
- Mac OS X

La librería de desarrollo GTK+ está compuesta por las librerías Glib, Atk, Pango, cairo, en donde:

Glib es una librería ampliamente utilizada por GTK+ ya que proporciona sustituciones a funciones estándar para el manejo de listas enlazadas, mecanismos para el tratamiento de errores, gestión de memoria, etc., éstas se utilizan para aumentar la portabilidad de GTK, ya que varias de las funciones incluidas en algunos sistemas operativos no se encuentran disponibles en otros.

"ATK o Accessibility Toolkit es una biblioteca de la API GTK+ que permite a los programadores de aplicaciones del entorno GNOME [Ref27] crear interfaces con características de gran accesibilidad, importante para personas discapacitadas o minusválidas. Se pueden utilizar lupas de aumento, lectores de pantalla, o entradas de datos alternativas al clásico teclado o mouse." [Wik03]

"Cairo es una biblioteca gráfica de la API GTK+ proporciona imágenes basadas en gráficos vectoriales. Aunque cairo es una API independiente y no hace uso directo de los dispositivos de la máquina en la que se encuentre**,** está diseñada para usar aceleración por hardware cuando esté disponible. Cairo deja disponibles numerosas primitivas de imágenes de segunda dimensión." [Wik04]

"Pango es una biblioteca de código abierto para el diseño y dibujo de texto internacional como parte del conjunto GTK+, por lo tanto del entorno gráfico GNOME para sistemas operativos GNU/Linux [Ref27]. "[Wik05]

GTK+ como muchas librerías para el desarrollo de interfaces gráficas de usuarios, está orientada para trabajar con eventos y señales, es decir, las aplicaciones que se desarrollan con la librería siempre están a la espera de una señal, donde las señales son emitidas por el sistema operativo cuando el usuario realiza alguna acción como: maximizar, minimizar, mover la ventana, presionar un botón, etc., por lo que la aplicación al recibir ésta señal realizará una tarea específica previamente programada. La construcción de las interfaces gráficas de usuario con está librería se hace a través de widgets [Ref28], éstos son componentes de control que el programador utiliza y son todos los objetos integrables en

nuestra aplicación como ventanas, cuadros combinados, cuadros de texto, botones, menús, combobox, etc. En la *Figura 2.1* se muestran algunos widgets que proporciona la librería GTK+

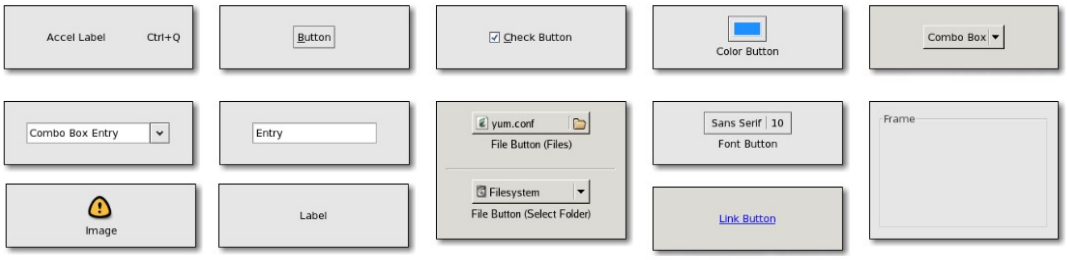


Figura 2.1 Algunos widgets que proporciona GTK+.

El siguiente ejemplo muestra cómo utilizar widgets.

```
#include <gtk/gtk.h>

int main (int argc, char *argv[]){
        GtkWidget *window;
        GtkWidget *button;

        //Inicializamos GTK
        gtk_init (&argc, &argv);

        //Creamos una ventana con titulo
        window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
        gtk_window_set_title (GTK_WINDOW (window),"GTK");

        //Creamos un boton con titulo (Salir) y lo agregamos a la ventana
        button = gtk_button_new_with_mnemonic ("Salir"));
        gtk_container_add (GTK_CONTAINER (window), button);

        //Agregamos la señal destroy al botón
        g_signal_connect ((gpointer) button, "destroy",  G_CALLBACK(gtk_main_quit), NULL);

        //Mostramos la ventana y todo los elementos que contiene
        gtk_widget_show_all(window);

        //Entramos al ciclo de GTK+
        gtk_main();

        return 0;
}
```

Analizando el ejemplo anterior:

Para utilizar GTK+ en una aplicación es necesario indicarle al compilador que se hará uso funciones y widgets que se encuentran en las librerías de GTK+ de la siguiente manera:

#include <gtk/gtk.h>

Ya que se creará una ventana con un botón dentro de ella, es necesario tener dos variables que almacenen la información de los widgets (window y button):

```
GtkWidget *window;
GtkWidget *button;
```

Para hacer uso de los componentes de GTK+ lo primero que se debe de hacer es: inicializar la librería con una llamada a la función gtk_init, está función recibe los argumentos de la función principal (main) como se muestra a continuación:

```
gtk_init (&argc, &argv);
```

Para crear una ventana es necesario hacer uso de la función "gtk_window_new" que recibe como dato, el tipo de ventana que se creará, en este caso "GTK_WINDOW_TOPLEVEL", que quiere decir que la ventana será una ventana principal. Para colocar un título en la ventana se hace uso de la función "gtk_window_set_title", los datos que recibe la función son: la ventana a la que se le colocara el título y una cadena que se mostrara como título.

```
//Creamos una ventana con titulo
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
gtk_window_set_title (GTK_WINDOW (window),"GTK");
```

Para crear un botón que muestre el texto "Salir", se hace uso de la función "gtk_button_new_with_mnemonic", el dato que recibe es el texto que mostrara el botón. Creado este es necesario colocarlo en el interior de a la ventana, esto lo hacemos con la función "gtk_container_add", los datos que recibe son: la ventana en la que estará el botón y el segundo dato es la variable que tiene los datos del botón, en este caso la variable button.

```
//Creamos un boton con titulo (Salir) y lo agregamos a la ventana
button = gtk_button_new_with_mnemonic ("Salir"));
gtk_container_add (GTK_CONTAINER (window), button);
```

Una vez que se ha colocado el botón en la ventana es necesario conectar una señal a este, en este caso la señal es "destroy", está señal se emite cuando se cierra una aplicación, así que al conectar la señal al botón, está se emitirá cuando se presione. Para conectar la señal con el botón se hace uso de la función "g_signal_connect", está conecta una señal con un widget y recibe cuatro datos: el primero es el widget que emite la señal (el botón), el segundo es el nombre de la señal que se emite (destroy), el tercero es la función encargada de manejar la señal que se emite, la función "gtk_main_quit" proporcionada por la librería. El último es un apuntador a los datos que recibe la función que maneja la señal, "NULL" [Ref29] ya que no recibe ningún dato, como se muestra a continuación.

```
//Agregamos la señal destroy al boton y a la ventana
g_signal_connect ((gpointer) button, "destroy",  G_CALLBACK(gtk_main_quit), NULL).
```

Creada la ventana es necesario hacer uso de la función "gtk_widget_show_all" que se encarga de mostrar la ventana y su contenido. Finalmente se debe pasar el control a GTK+ que se encargara de manipular la ventana por medio de la función "gtk_main()".

```
//Mostramos la ventana y todo los elementos que contiene
gtk_widget_show_all(window);

//Entramos al ciclo de GTK+
gtk_main();
```

El resultado final se muestra en la *Figura 2.2*.

Figura 2.2 Ejemplo de una ventana desarrollada con GTK+.

2.3 Elección de la librería gráfica

ROC2 es un simulador de robots móviles, donde el dibujado de las escenas tridimensionales se hace mediante la librería gráfica DirectX 8 [Ref30], que es propiedad de Microsoft y exclusiva para la plataforma Windows en sus versiones Windows NT/2000/ XP, debido a esto y a que se requiere que la nueva versión de ROC2 sea multiplataforma, se decidió escoger la librería gráfica OpenGL [Ref31], ya que proporciona las siguientes ventajas.

- Es una librería multiplataforma soportada en prácticamente todos los sistemas operativos.
- Existe una extensa documentación sobre esta librería gráfica en muchos idiomas.
- Es independiente del Hardware de la máquina donde se utilice.
- La utilización de ésta librería en aplicaciones es relativamente fácil.
- Proporciona soporte para una infinidad de lenguajes de programación.
- Se pueden realizar implementaciones de OpenGL y GTK+ de una forma sencilla.

OpenGL es una librería gráfica multilenguaje y multiplataforma enfocada esencialmente en el render, por lo que es soportada en una gran cantidad de sistemas operativos, ésta librería fue diseñada para ocultar lo complejo de la interfaz, de la programación directa con las tarjetas gráficas, proporcionando una interfaz única y uniforme entre la aplicación y la tarjeta gráfica que resulta aún más poderosa al hacerla independiente de la plataforma.

El funcionamiento básico de OpenGL consiste en proporcionar una serie de funciones para el dibujado de primitivas, como puntos, líneas y polígonos, con los cuales se pueden dibujar objetos tridimensionales más complejos.

OpenGL es una librería que trabaja como una máquina de estados. Lo que quiere decir es que OpenGL activa y desactiva opciones y realiza operaciones de dibujado, traslación, etc., que dan como resultado una representación en pantalla en forma de imágenes. Esto quiere decir que dependiendo del estado en que se encuentra, será el resultado observado en la pantalla, ya que no será lo mismo dibujar un triángulo y activar una textura, que activar una textura y dibujar un triángulo, de igual manera no será lo mismo trasladar y rotar un objeto

que rotar y trasladar el mismo objeto. En otras palabras dependiendo del orden de las acciones que se realicen será el resultado que se obtendrá.

Tomando en cuenta lo anterior, los pasos para dibujar un objeto son:

1. Activar todas las opciones que van a ser persistentes a la escena (colocamos la cámara en su posición, activamos la iluminación, etc.)
2. Activar las opciones que establecen el estado de un objeto específico (color, posición, texturas, etc.)
3. Dibujar el objeto.
4. Desactivar las opciones que se establecieron para este objeto.
5. Regresar al punto 2 e iniciar el proceso para dibujar el siguiente objeto, así hasta dibujar todos los objetos.

Como se puede observar es un esquema sencillo de implementar, que permite la agrupación en jerarquías y proporciona gran flexibilidad a la hora de programar.

Aunque OpenGL es una librería independiente de GTK+, existe una librería llamada GtkGLExt, que es una extensión de OpenGL para GTK+, proporciona funciones para la implementación de OpenGL sobre aplicaciones desarrolladas con GTK+, que permite la utilización de todas las funciones de OpenGL sin problema alguno.

A continuación se muestra un ejemplo sencillo de una implementación de GTK+ con OpenGL y el resultado que se obtiene a compilar y ejecutar el programa se muestra en la figura 2.3.

```
#include <stdio.h>
#include <stdlib.h>
#include <gtk/gtk.h>
#include <gtk/gtkgl.h>

#ifdef G_OS_WIN32
#define WIN32_LEAN_AND_MEAN 1
#include <windows.h>
#endif

#include <GL/gl.h>
#include <GL/glu.h>

gboolean configure_event (GtkWidget *,GdkEventConfigure *,gpointer );
gboolean expose_event (GtkWidget *,GdkEventExpose *,gpointer );

int main(int argc, char *argv[]){
        GtkWidget *window;
        GtkWidget *drawing_area;
        GdkGLConfig *glconfig;

        //inicializamos GTK
        gtk_init(&argc, &argv);

        //Inicializamos opengl
        gtk_gl_init (&argc, &argv);
```

```
	//Creamos la ventana de la aplicacion
	window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
	gtk_window_set_title(GTK_WINDOW (window),"gtk y opengl");
	gtk_window_set_default_size (GTK_WINDOW (window),200,200);
	gtk_container_set_reallocate_redraws (GTK_CONTAINER (window), TRUE);
	g_signal_connect (G_OBJECT (window), "destroy",  G_CALLBACK(gtk_main_quit),NULL);

	//confuguracion de openGL,Seleccion del modo y buffer
	glconfig = gdk_gl_config_new_by_mode
(static_cast<GdkGLConfigMode>(GDK_GL_MODE_RGB|GDK_GL_MODE_DOUBLE|
GDK_GL_MODE_DEPTH));

	//Verificamos que se pudo configurar openGL
	if (glconfig == NULL){
		g_print ("*** Cannot find the double-buffered visual.\n");
		g_print ("*** Trying single-buffered visual.\n");

		glconfig = gdk_gl_config_new_by_mode
(static_cast<GdkGLConfigMode>(GDK_GL_MODE_RGB|GDK_GL_MODE_DEPTH));

		if (glconfig == NULL){
			printf("*** No appropriate OpenGL-capable visual found.\n");
			return false;
		}
	}

	//Creamos el widget para el viewport
	drawing_area=gtk_drawing_area_new ();

	//Configuramos el widget para opengl
	gtk_widget_set_gl_capability (drawing_area,glconfig,NULL,TRUE,GDK_GL_RGBA_TYPE);

	//Evento que se lanza cuando se necesita redibujar la ventana
	g_signal_connect (G_OBJECT (drawing_area), "expose_event",G_CALLBACK (expose_event),
NULL);

	//Evento que se lanza cuando una ventana cambia de tamaño
	g_signal_connect (G_OBJECT (drawing_area), "configure_event",G_CALLBACK
(configure_event), NULL);

	//Agregamos el viewport a la ventana
	gtk_container_add (GTK_CONTAINER (window), drawing_area);

	//Mostromos la ventana y su contenido
	gtk_widget_show_all(window);
	gtk_main();

	return 0;
}

//Funcion que se llama cuando se necesita redimencionar la ventana
gboolean configure_event (GtkWidget *widget,GdkEventConfigure *event,gpointer data){

	//Obtenemos el contexto de opengl
	GdkGLContext *glcontext = gtk_widget_get_gl_context (widget);
	//Obtenemos widget del viewport
	GdkGLDrawable *gldrawable = gtk_widget_get_gl_drawable (widget);
```

```
	//Iniciamos openGL para modificarlo
	if (gdk_gl_drawable_gl_begin (gldrawable, glcontext)){

		glEnable (GL_DEPTH_TEST);

		//Definimos las dimenciones del viewport y sus parametros
		glViewport (0, 0,widget->allocation.width,widget->allocation.height);
		glMatrixMode(GL_PROJECTION);
		glLoadIdentity();
		gluPerspective(45.0f, (GLfloat) widget->allocation.width/ (GLfloat)widget->alloca-
tion.height, 1.0f,1000.0f);

		//Definimos la posicion de la camara
		glMatrixMode(GL_MODELVIEW);
		glLoadIdentity();
		gluLookAt(0, 0, 20.0,0, 0, 0.0,0.0, 1.0, 0.0);

		//Cerramos openGL una vez que lo modificamos
		gdk_gl_drawable_gl_end (gldrawable);
	}

	return TRUE;
}

//Funcion que se llama cuando se necesita redibujar la ventana
gboolean expose_event (GtkWidget *widget,GdkEventExpose *event,gpointer data){

	//Obtenemos el contexto de opengl
	GdkGLContext *glcontext = gtk_widget_get_gl_context (widget);
	//Obtenemos widget del viewport
	GdkGLDrawable *gldrawable = gtk_widget_get_gl_drawable (widget);

	//Verificamos que podamos iniciar openGL para poder modificarlo
	if (gdk_gl_drawable_gl_begin (gldrawable, glcontext)){

		glClearColor(1,1,1,0);
		glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

		//Dibujamos una esfera de color rojo en el origen
		glColor3f(1,0,0);
		gdk_gl_draw_sphere(true,3,50,50);

		if (gdk_gl_drawable_is_double_buffered (gldrawable))
			gdk_gl_drawable_swap_buffers (gldrawable);
		else
			glFlush ();

		gdk_gl_drawable_gl_end (gldrawable);
	}

	return TRUE;
}
```

Figura 2.3 Ejemplo de una ventana desarrollada con GTK+ y OpenGL.

2.4 Elección de un motor de física

Newton Game Dynamic (NGD) [Ref32] integra una serie de funciones para simular física en tiempo real, proporciona detección de colisiones, así como dinámica del comportamiento de cuerpos rígidos.

Las principales ventajas de esta librería son:

- Es una librería multiplataforma soportada en sistemas operativos win32, Unix y MAC.
- Proporciona un sistema de simulación de Física.
- Es independiente del motor gráfico.
- Proporciona un sistema independiente para el manejo de colisiones.
- Proporciona funciones para el manejo de articulaciones

NGD es una librería multiplataforma, por lo que las implementaciones que se realicen utilizando la librería en una plataforma serán totalmente funciónales en cualquier otro sistema operativo que sea soportado por la librería. [New08]

Es independiente del motor gráfico, es decir, la simulación que se realice con NGD será totalmente independiente de la representación visualización de ésta, así que se deja al desarrollador la implementación del dibujado de los objetos. Para facilitar esta parte al desarrollador, NGD proporciona una matriz de transformaciones de 4x4 para cada objeto que se esté simulando, en donde se encuentran todas las transformaciones necesarias para localizar el objeto en la escena gráfica, como son rotación en el eje x, y, z, y posición del objeto en el mundo virtual.

La librería también proporciona funciones para el uso del sistema de colisiones independiente del sistema de física, es decir, no es necesario tener una simulación de un mundo virtual para realizar la detección de colisiones entre objetos, sólo es necesario definir una serie de objetos denominados primitivas de colisión, como: esferas, conos,

cajas, cilindros, modelos tridimensionales, así como objetos complejos compuestos por objetos primitivos o modelos tridimensionales. Con los que se pueden verificar las colisiones entre ellos a partir de funciones definidas en la librería NGD.

También proporciona funciones para el manejo de articulaciones, se pueden crear uniones entre objetos como son: uniones rígidas, bisagras, uniones deslizables, unión bola socket, etc. Con las cuales se crean una serie de efectos, como el de una puerta que al abrir o cerrar gira sobre uno de sus ejes, este efecto se puede simular con una unión tipo bisagra.

Realizar simulaciones usando NGD es sencillo, ya que lo único que se necesita es crear un mundo en el que se colocaran los objetos a simular, dentro de éste se pueden crear una infinidad de objetos, ya sean geometrías básicas como cilindros, conos, cajas, esferas, así como geometrías complejas como son objetos tridimensionales u objetos compuestos de geometrías básicas. Una vez que se ha creado el mundo virtual con los objetos que se desean simular, es necesario realizar un paso de simulación, para conocer la posición de los objetos en un instante de tiempo.

En la vida real los objetos en todo momento están interactuando entre sí, reaccionando a fuerzas del medio ambiente en el que se encuentran, lo que quiere decir que la física siempre esta activa. Simular esto por medio de ordenadores resulta imposible. Por lo que los motores de física dividen la simulación en intervalos de tiempo lo más pequeño posible, en los que sólo en estos instantes de tiempo es donde interactúan los objetos entre sí y están sujetos a fuerzas del medio ambiente que se encuentran como la gravedad. Por ejemplo, con intervalos de tiempo de 0.003 segundos los objetos que se encuentran en el mundo virtual solo interactúan por un instante entre sí y con las fuerzas presentes en el medio ambiente, con lo que los objetos pueden cambiar de posición y orientación.

Debido a que NGD no proporciona mecanismos para dibujar los objetos que se simulen en el mundo virtual ficticio, el programador es el encargado de realizar esta operación entre cada paso de simulación, para poder visualizar los cambios que se realicen en la simulación.

Capítulo III

Simulador

3.1 Introducción

La interfaz gráfica de usuario no es la parte esencial de la aplicación, sin embargo, es una parte importante del desarrollo de software, ya que en ocasiones la aceptación de una aplicación no depende de que tan útil sea o que tan eficiente resuelva un problema, sino que tan amigable resulte para el usuario final.

Una interfaz gráfica de usuario busca tener las siguientes características [Ref33]:

- Niveles de capacidad múltiple. Quiere decir que es posible realizar una misma acción de dos o más formas diferentes, debido a que existen usuarios con diferentes niveles de capacidad.
- Rapidez de aprendizaje. Tiene que ver con el tiempo que le toma a un nuevo usuario aprender a utilizar la interfaz de forma eficiente.

- Consistencia. Deben preservarse los significados de los elementos durante toda la interfaz, así como su presentación.
- Mínimo de memorización. Determina que se debe planear la interfaz de forma que no sea necesario memorizar una gran cantidad de pasos para realizar tareas comunes o sencillas.
- Respaldo y manejo de errores. Indica que debe ser posible cancelar una acción indicada así como deshacerla, en caso que sea imposible deshacer la operación se debe generar una advertencia.
- Retroalimentación. Quiere decir que la interfaz no sólo recibe información del usuario sino que muestra el progreso de las acciones que realiza en cada paso; esto es de especial importancia cuando el tiempo de respuesta es alto.

Lo que se pretende solucionar con una interfaz gráfica de usuario es cubrir una necesidad específica de información usando los recursos que se tienen.

3.2 Estructura interna del simulador

La interfaz que se ha desarrollado para esta actualización Roc2008 es similar a la que se cuenta la versión anterior de ROC2, sigue conservando las características de la versión anterior, como son el control de robots virtuales por medio del teclado e instrucciones específicas proporcionadas por el usuario. Por lo que la aplicación puede ser utilizada como simulador de robots o como medio de visualización de resultados.

La siguiente lista muestra características que se conservan de la versión anterior.

• Uso de 3 ventanas para visualizar resultados de simulación.
• Navegar libre del ambiente tridimensional.
• Consola de comandos.
• Control sobre uno o más robots virtuales.
• Posibilidad de cambiar el ambiente de simulación.
• Manipular los robots de forma local como remota.
• Compatibilidad con comandos de Robel [Ref34].

Los comandos de Robel que son soportados se pueden ver en la *Tabla B.6*.

La *Figura 3.1* muestra de forma general el funcionamiento del simulador.

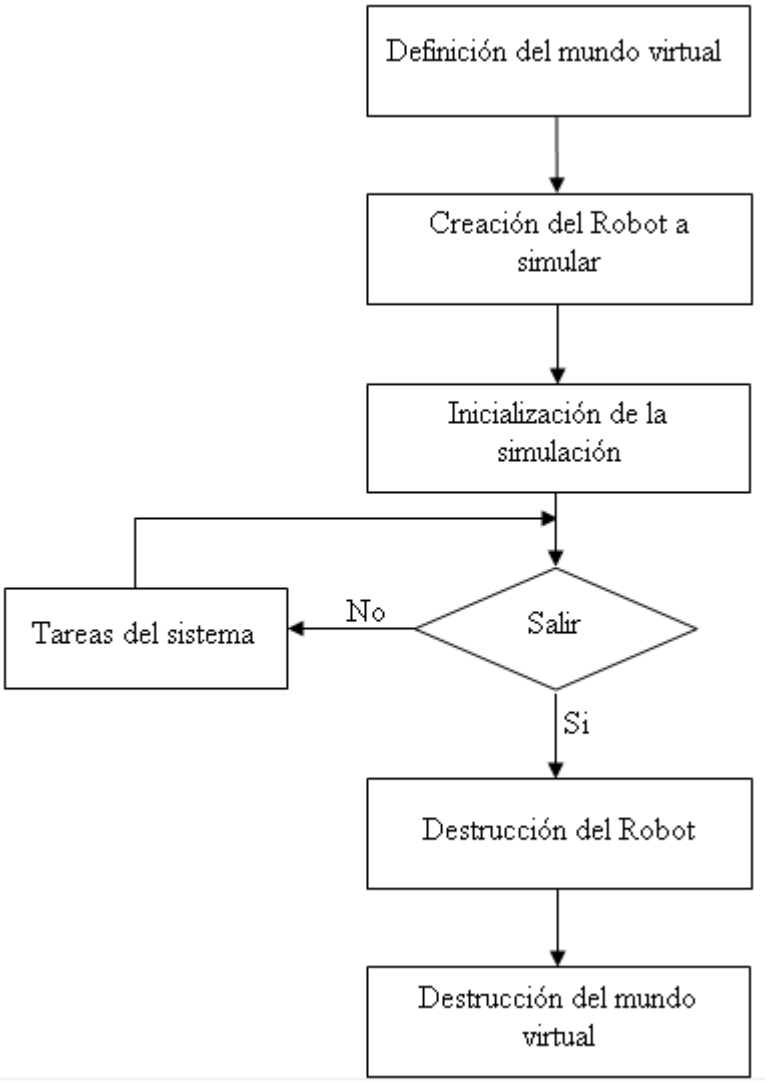


Figura 3.1 Estructura interna del simulador.

Definición del mundo virtual

El mundo virtual se define mediante una escena gráfica formada por un conjunto de geometrías, luces y un mapa de descripción.

Un mapa de descripción es un archivo de texto con extensión wrl, que puede ser creado en cualquier editor de texto, en donde se describe la estructura de la escena por medio de ciertas reglas las cuales se muestran en el tema 5.4 (Obstáculos tipo mapa). La ventaja de utilizar mapas de descripción es que permiten especificar las dimensiones del mundo que se desea cargar, así como la ubicación de los obstáculos que lo conforman, además se pueden hacer representaciones de un mundo virtual con un mínimo de geometrías, por lo que se reflejará en el desempeño de la simulación, la desventaja que se presenta es que no resulta sencillo hacer representaciones de mundos demasiado complejos.

Creación de un robot a simular

Una vez que se tiene el entorno virtual en el que se realizarán las simulaciones es necesario crear el robot a simular, el cual deberá contar con una estructura lógica para manipular el robot en la escena gráfica. La creación de un robot se describe con mayor detalle en el tema 4.2 (Creación de un robot).

Inicialización de la simulación

Esta etapa consiste en procesar la escena gráfica, formada por geometrías, imágenes y luces y añadir las funciones que permitirán al usuario interactuar con el simulador.

Tareas del sistema

En esta etapa se realizan todas actividades relevantes del simulador, es la encargada de realizar las tareas solicitadas por el usuario, como manipular la interfaz gráfica o los robots que se estén simulando en ese momento.

Destrucción del robot y del mundo virtual

Una vez que se decide terminar con la simulación y cerrar la aplicación, es necesario destruir los robots que se estén simulando, así como cualquier objeto que se encuentre dentro de la simulación, debido a que cada uno de los elementos que se encuentran está representado por una estructura lógica que consume memoria de la máquina. Por lo que es necesario liberar los recursos que se estén utilizando para que cualquier aplicación pueda hacer uso de éstos.

3.3 Ejecución de tares del sistema

Por medio de éstas se controla la mayoría de las funciones de la aplicación, entre los que se encuentran: mover la ventana de la aplicación, la cámara, ejecutar un comando, agregar, mover, seleccionar y eliminar objetos, etc.

Debido a la forma en que se programo el sistema, las tareas que se realizan en él tienen una secuencia específica, en la *Figura 3.2* se puede observar por medio de un diagrama de flujo en el que se muestra el orden en el que se ejecuta cada tarea.

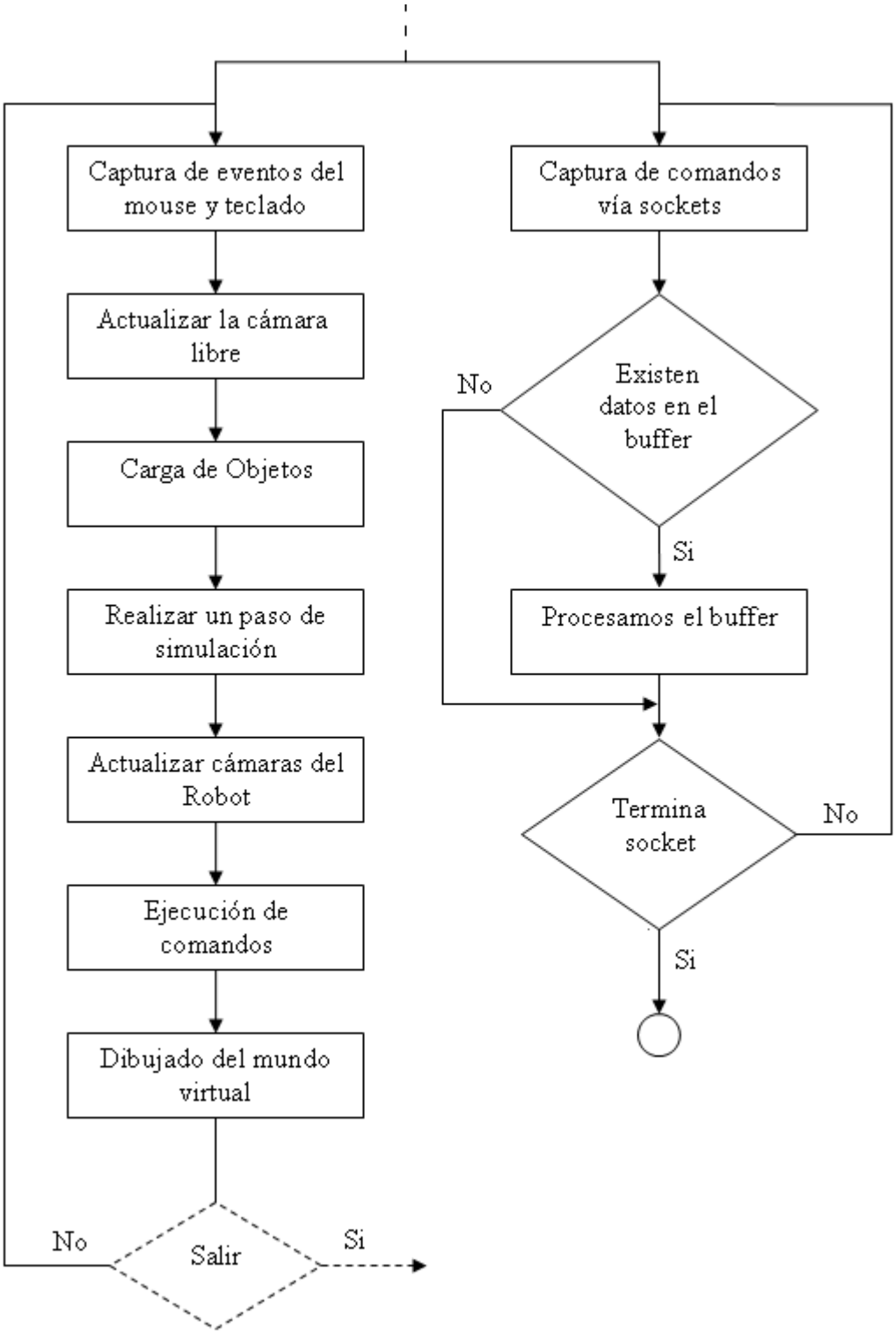


Figura 3.2 Orden de ejecución de las tareas dentro del simulador.

Debido a que es posible manipular los robots a través de funciones definidas por el usuario, que a su vez pueden hacer uso de sockets [Ref35] para la comunicación con otros maquinas, es necesario tener un hilo [Ref36] en el que siempre se este preguntando si existe algún mensaje en los sockets.

Captura de eventos del mouse y teclado

Esta tarea es la encargada de leer el buffer [Ref37] del teclado y el estado del mouse, ya que por medio del teclado se pueden manipular los robots mediante el uso de comandos, así que cuando se detecta un comando proveniente del teclado, éste no es ejecutado de inmediato, debido a que el robot puede estar realizando otras tareas, por lo que el comando es colocado en la lista de comandos del robot para su futura ejecución. Con el mouse es posible cambiar la posición y orientación de la cámara libre, para mayor información ir al tema 3.5 (Cámara libre), también es posible manipular la interfaz gráfica, ya sea redimensionar, mover o minimizar la ventana, etc.

Actualizar la cámara libre

Cuando se utiliza el mouse para cambiar la posición y orientación de la cámara libre es necesario mover y hacer uso de los botones del mouse, dependiendo de la tarea que se desea realizar, para mayor información ir al tema 3.5 (Cámara libre), por lo que la tarea de mover la posición y orientación de la cámara no se realiza en ese momento, si no hasta el siguiente paso de simulación, ya que si se realizaran los cambios solicitados en ese momento no se tendría el efecto de movimiento continuo, si no un efecto de movimiento en pasos, por lo que las acciones realizadas como presionar y mover el mouse persistirán hasta que se realice otra acción diferente con el mouse, dando el efecto de movimiento continuo.

Carga de objetos

Cuando en usuario solicita agregar un objeto en la escena a través de funciones de usuario, el simulador puede estar realizando alguna tarea, así que las instrucciones para cargar objetos se almacenan en una lista de objetos a cargar. Cuando el simulador termina las tareas que esté realizando da inicio nuevamente el ciclo de simulación, al pasar por esta etapa verifica si existe algún objeto que deba cargar, si es así, realiza los pasos necesarios para cargar los objetos y elimina las instrucciones de la lista.

Realizar un paso de simulación

Los objetos que se encuentran en el mundo virtual así como en la vida real se encuentran inmersos en un entorno en el que existen varias fuerzas como la de gravedad, por lo que al interactuar las fuerzas con los objetos provocan que se muevan, esto se simulan a través de la librería NGD. Debido que para el simulador resulta imposible hacer esto para cada instante de tiempo, la simulación de la física sólo se realiza cada cierto instante de tiempo.

Un paso de simulación se realiza cuando se simulan las fuerzas que actúan sobre un objeto como la gravedad, sólo para un instante de tiempo, con lo que se obtiene una matriz de transformaciones para cada objeto que se encuentre el mundo virtual, que contiene la posición y orientación del objeto para ese instante de tiempo.
Por ejemplo se desea mover un robot de la posición (x,y) a la posición (x1,y1), el movimiento no se realiza en un solo paso, si no en varios pasos, en el primer paso de

simulación el robot se mueve a la posición (x',y'), en el siguiente paso se mueve a la posición (x'',y''), así hasta que x^n=x1 y y^n=y1. Para mayor detalle sobre los movimientos del robot ir al tema 4.2.2 (Movimiento del robot).

Actualizar las cámaras del robot

Una vez que se realizó un paso de simulación, los robots que se encuentran en el mundo virtual pueden haber cambiado de posición y orientación, por lo que es necesario actualizar la posición de las cámaras asociadas a los robots que se estén simulando.

Ejecución de comandos

En esta etapa se realizan las tareas más importantes del simulador, que son ejecutar los comandos del robot, para mayor información sobre los comandos que puede ejecutar un robot ir a la *Tabla B.6*.

Debido a que algunos comandos requieren que el robot realice algún tipo de movimiento de rotación o traslación, es necesario que estos comandos se realicen en más de un paso de simulación, esto quiere decir que si el robot debe moverse de la posición (x,y) a la posición (x1,y1), debe de moverse en (n) pasos, ya que si se ejecutara en un solo paso no se tendría el efecto de movimiento continuo, si no un efecto de tele transportación.

Existen algunos comandos que no requieren más de un paso de simulación como son las lecturas de los sensores, ya que esta operación no implica realizar un movimiento del robot.

Dibujado del mundo virtual

Una vez que se realizaron todos los pasos anteriores los objetos y robots que se encuentran en el mundo virtual pueden haber cambiado de posición u orientación, por lo que es necesario dibujar nuevamente el mundo para poder observar los cambios que se han realizado.

Captura de comandos vía socket

Debido a que los comandos que ejecutan los robots pueden provenir de funciones de usuario, de la línea de comandos o de Internet, es necesario tener un hilo que siempre esté verificando si existe algún comando proveniente de Internet, ya que el simulador puede estar ocupado realizando alguna tarea importante.

Los sockets (programa que permite el flujo de datos entre dos aplicaciones) soportados para realizar la conexión con otros dispositivos son de tipo UDP y TCP.
Sockets UDP es un socket que está basado en el envío y/o recepción de información, no es necesario establecer una conexión fija entre las dos aplicaciones para realizar el envío y/o recepción de la información y no establece ningún control de errores.

Sockets TCP es un socket que está basado en el envío y/o recepción de información, con conexión fija entre las dos aplicaciones. Lo que quiere decir que es necesario mantener una conexión entre las dos aplicaciones para que se puedan enviar o recibir información. Este tipo de socket si proporciona mecanismos de control de errores.

3.4 Ventanas y puntos de vista

El simulador cuenta con una ventana principal donde se muestran cuatro puntos de vista (viewports) [Ref38], los cuales muestran el mundo virtual desde una cierta posición y orientación, estas ventanas tienen integradas funciones proporcionadas por GTK+ con las que se manipulan acciones básicas, como: redimensionar la ventana o mover la orientación y posición de la cámara.

Cada vez que se crea un mundo virtual es necesario crear un punto de vista o cámara por el cual se observara dicho mundo. Debido a que el medio donde se despliegan las imágenes generadas por la simulación es un dispositivo bidimensional (monitor), en el que sólo se pueden observar representaciones de imágenes en dos dimensiones, por lo que para observar una representación de una escena tridimensional es necesario definir un punto de vista, en el cual se define la posición y orientación de un observador en la escena tridimensional desde la que todas las geometrías asociadas con la simulación serán proyectas en la ventana y representados en imágenes bidimensionales.

Un punto de vista proporciona un medio de visualización donde se proyectaran imágenes 2D, que son la representación de imágenes 3D en el mundo virtual después de realizar la proyección de estas sobre el punto de vista. Y que son las imágenes que se ven a través del lente de una cámara imaginaria colocada con una cierta posición y orientación sobre el mundo tridimensional.

3.5 Cámara libre

En el primer punto de vista está asignada la ventana de la cámara libre, que es una cámara que el usuario puede usar para navegar por todo el mundo virtual. Ésta puede ser manipulada por el usuario a través del mouse, y dispone de seis grados de libertad para navegar en la escena.

En la *Figura 3.3* se muestra un diagrama de cómo se divide la ventana en 9 zonas y dependiendo de la zona en la que se encuentre el mouse y que botones se presionen, será el movimiento que se realicé, esto es para ayudar a entender cómo se realizan los movimientos de la cámara

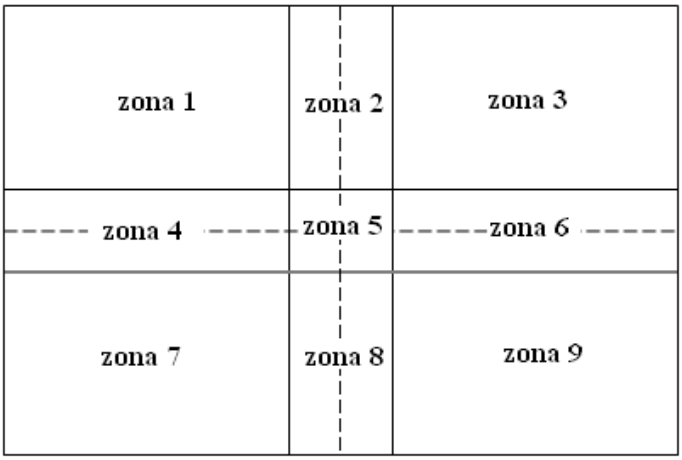


Figura 3.3. Representación de las zonas en las que se divide la ventana.

En la *Figura 3.4* se muestra la orientación de los ejes coordenados utilizados para el sistema.

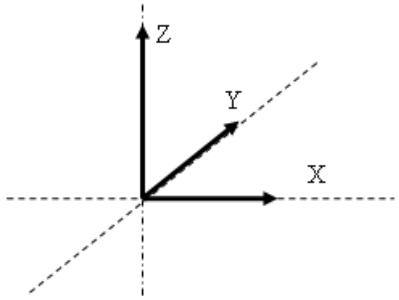


Figura 3.4 Orientación de los ejes coordenados del sistema.

En la *Tabla 3.1* muestra los efectos sobre la cámara libre dependiendo del botón del mouse que se ha presionado y la zona en la que este se encuentre.

zona	Evento del mouse	Efectos sobre la cámara
1	Ningún botón se presiona	Ninguno
2	Se presiona botón izquierdo	La cámara se mueve hacia delante
8	Se presiona botón izquierdo	La cámara se mueve hacia atrás
6	Se presiona botón izquierdo	La cámara rota a la izquierda sobre el eje z
4	Se presiona botón izquierdo	La cámara rota a la derecha sobre el eje z
2	Se presiona botón derecho	La cámara se mueve hacia arriba
8	Se presiona botón derecho	La cámara se mueve hacia abajo
6	Se presiona botón derecho	La cámara se mueve hacia la derecha

4	Se presiona botón derecho	La cámara se mueve hacia la izquierda
2	Se presiona el botón izquierdo y derecho	La cámara rota hacia arriba sobre el eje x
8	Se presiona el botón izquierdo y derecho	La cámara rota hacia abajo sobre el eje x
6	Se presiona el botón izquierdo y derecho	La cámara rota hacia la izquierda sobre el eje y
4	Se presiona el botón izquierdo y derecho	La cámara rota hacia la derecha sobre el eje y

Tabla 3.1 Efectos del mouse sobre la cámara libre.

3.6 Cámara del robot

Esta cámara está localizada sobre el robot y orientada en la dirección del vector director del robot, cada vez que se realiza un movimiento ya sea de rotación o traslación es necesario redefinir la posición y orientación de la cámara, en la *Figura 3.4* y la *Figura 3.5* se muestra la vista superior y la vista del robot respectivamente.

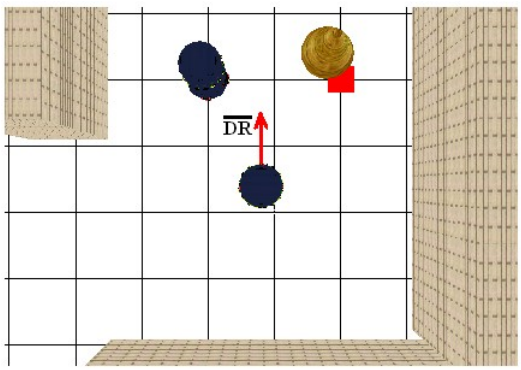


Figura 3.4 Vista superior del Robot.

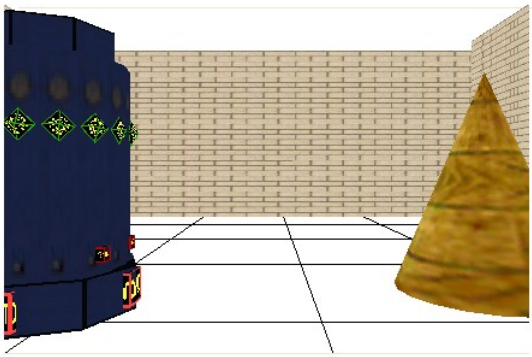


Figura 3.5 Imagen capturada de la cámara del robot.

En donde el vector $\overline{DR}$ es el vector director del robot.

Para representar una cámara del robot son necesarios tres datos.

- Coordenadas en formato (x,y,z) donde se encuentra posicionada la cámara.
- Coordenadas en formato (x,y,z) del punto que observa la cámara.
- Vector director del eje que indica la posición hacia arriba de la cámara.

A continuación se muestra el pseudocódigo [Ref39] para actualizar la posición de la cámara del robot.

Función actualiza_la_vista_del_robot()
    Obtener la posición actual del robot.
    Obtener el ángulo de rotación actual del robot sobre el eje z.
    Obtención de las coordenadas del punto que observa el robot.
    Obtención del vector director del eje hacia arriba de la cámara.
    Actualizar la cámara.
Fin función.

3.7 Cámara superior

Esta cámara se encuentra localizada sobre el robot a una cierta distancia, el punto que observa esta cámara es el centro del robot y el vector director del eje hacia arriba de la cámara es el vector director del robot, por lo que la cámara siempre sigue los movimientos del robot desde una vista superior, en la *Figura 3.6* se muestra la vista de la cámara superior.

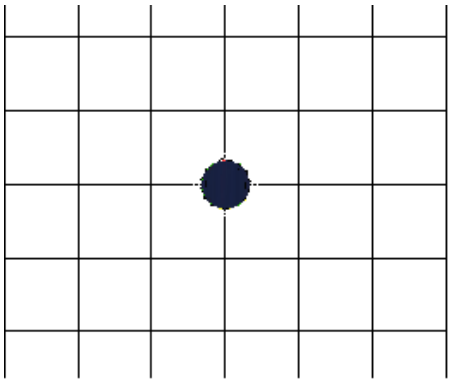


Figura 3.6 Imagen capturada de la cámara superior del robot.

A continuación se muestra el pseudocódigo, con el que se actualiza la cámara superior.

Función actualiza_la_vista_superior_del_robot()
    Obtener la posición actual del robot.
    Obtener la posición donde se encuentra la cámara.
    Obtener el vector director del robot.
    Actualizar la cámara.
Fin función.

3.8 Cámara lateral

Además de las tres cámaras anteriores se proporciona una cuarta cámara, cámara lateral que al igual que la cámara superior esta siempre sigue al robot, por lo que el punto que ve esta cámara es el centro del robot, y se encuentra posicionada a un costado del robot, como se muestra en la *Figura 3.7*.

Figura 3.7 Imagen capturada de la cámara lateral del robot.

A continuación se muestra el pseudocódigo, con el que se actualiza la cámara lateral.

```
Función actualiza_la_vista_lateral_del_robot()
        Obtener la posición actual del robot.
        Obtener la posición donde se encuentra la cámara.
        Obtención del vector director del eje hacia arriba de la cámara.
        Actualizar la cámara.
Fin función.
```

3.9 Selección de objetos

Consiste en elegir objetos que se encuentran en el mundo virtual haciendo uso del mouse, mediante la selección de un punto sobre la ventana bidimensional, con el que se tendrá que elegir un objeto que se encuentre en el mundo virtual.

El primer problema que se presenta cuando se trata de elegir un objeto a través de una ventana, es que en una ventana se manejan coordenadas bidimensionales mientras que en un mundo tridimensional se manejan coordenadas tridimensionales, en la *Figura 3.8* se muestra este problema.

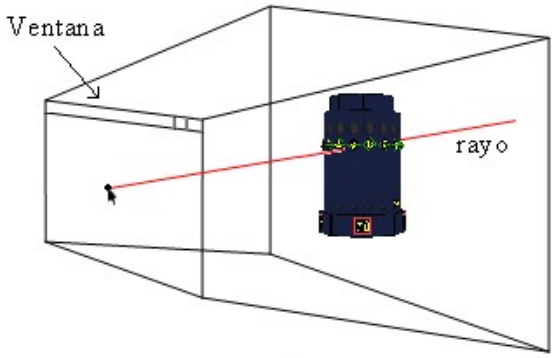


Figura 3.8 Representación visual del proceso del selección.

El proceso que se realiza es la operación inversa del dibujo de una escena tridimensional, en este proceso lo que se hace es pasar de una representación tridimensional a una representación bidimensional, por lo que muchos pixeles pueden caer sobre un solo punto de la pantalla bidimensional, pero sólo se muestra el píxel que se encuentre más cerca al observador.

Afortunadamente OpenGL proporciona la función gluUnProject, que permite obtener un punto de la escena tridimensional a partir de las coordenadas bidimensionales de un punto sobre la ventana de visualización, con lo que se pueden obtener una serie de puntos dentro del mundo virtual con los que se genera una línea en el espacio tridimensional.

Una vez que se obtienen los puntos con los que se crea una línea, es necesario verificar que objetos del espacio tridimensional se intersecan con la línea, para descartar aquellos objetos que no podrán seleccionarse. Dentro de los objetos que se intersecan con la línea se elige el que se encuentre a menor distancia del observador.

El realizar las pruebas para verificar si una línea se interseca con un objeto dentro de la escena tridimensional puede consumir demasiados recursos de la computadora, por lo que existen diferentes métodos para realizar este tipo de pruebas. Entre los cuales se encuentra la detección de colisiones asignando niveles de detalle, lo que quiere decir que se pueden definir varios niveles de detalle para un polígono, por ejemplo el primer detalle seria una esfera que contenga a todo el polígono, y el siguiente nivel de detalle podría ser una caja mínima que contenga el polígono, por lo que para verificar intersección entre el objeto y la línea, se verificara la intersección con el primer nivel de detalle que sería la esfera, si no existe intersección entre la línea y la esfera no será necesario verificar el siguiente nivel de detalle, ya que quiere decir que no existe intersección entre el objeto y la línea, lo cual resulta una operación mas eficiente que el verificar si la línea cruza con alguno de los n polígonos que forman el objeto.

Afortunadamente NGD proporciona la función “NewtonWorldRayCast”, que puede verificar si un objeto simulado colisiona con una línea, lo que facilita la selección de objetos en el mundo virtual.

A continuación se muestra el pseudocódigo para la selección de un robot dentro del mundo virtual.

```
Función selecciona_robot_con_mouse()
        Obtener las coordenadas del punto sobre la ventana.
        Obtener los puntos de la línea.
        If (verificar si la línea colisiona con algún robot)
                Se selecciona el robot
                Cambiamos la posición y orientación de las cámaras asociadas al robot
                seleccionado
        End if
Fin función
```

Capítulo IV

Robots

4.1 Creación de robots

En la versión anterior del ROC2 existía la posibilidad de crear dos tipos de robots, empleando modelos 3D, a través de un archivo en formato 3ds [Ref40], y utilizando un archivo en formato RBT, los cuales están representados por medio de primitivas básicas como: triángulos, cilindros, cubos, etc. Ahora, para esta nueva versión (roc2008) existe la posibilidad de crear un sólo tipo de robot, éste es una representación en 3D del robot B14, el cual se encuentra en el laboratorio de biorobotica de la Facultad de Ingeniería, dentro del simulador el nombre que tendrá el modelo es RTX8 , en la *Figura 4.1* se muestra el modelo del robot.

Figura 4.1 Modelo 3D del robot RTX8.

Este modelo cuenta con 16 sensores tipo sonar, dos sensores infrarrojos, cuatro sensores de contacto y un sensor reflectivo.

4.2 Estructura geométrica

El Robot RTX8 está compuesto por un cuerpo, consta de tres cilindros, 16 sensores de tipo sonar, dos sensores infrarrojos, cuatro sensores de contacto, un sensor reflectivo y cuatro llantas, de las cuales solo dos son utilizadas para mover el robot, como se muestra en la *Figura 4.2*y la *Figura 4.3*.

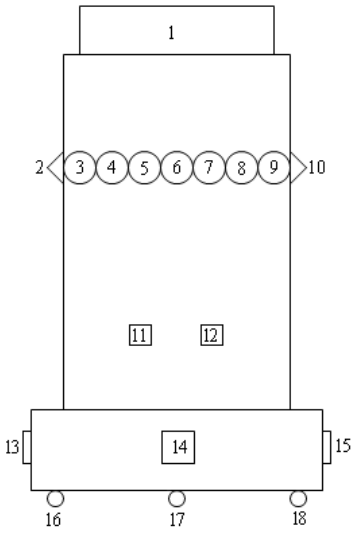


Figura 4.2 Vista frontal del robot RTX8.

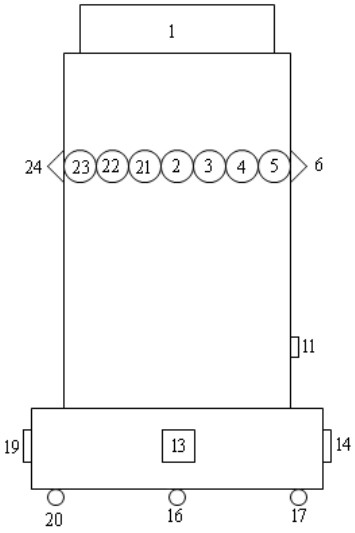


Figura 4.3 Vista lateral del robot RTX8.

Donde:

1.- cuerpo del Robot
2.- sonar
3.- sonar
4.- sonar
5.- sonar
6.- sonar

7.- sonar
8.- sonar
9.- sonar
10.- sonar
11.- sensor infrarrojo
12.- sensor infrarrojo
13.- sensor de contacto
14.- sensor de contacto
15.- sensor de contacto
16.- llanta izquierda
17.- llanta delantera
18.- llanta derecha
19.- sensor de contacto
20.- llanta trasera
21.- sonar
22.- sonar
23.- sonar
24.- sonar

4.2.1 Estructura lógica

Para controlar y conocer el estado del robot, se cuenta con una serie de estructuras de control y variables, utilizadas internamente por el simulador para manipular al robot.

Nombre: Nombre con el cual se identifica el robot dentro del mundo virtual.

Registro de datos: Estructura de datos de tipo flotante, entero y cadena para uso interno del robot.

Sensores: Estructuras tipo sonar para almacenar la posición, orientación, número y tipo de sensor.

Posición del robot: Esta representada por una terna (x,y,z).

Lista de comandos: Es una estructura donde son almacenados los comandos que recibe el robot ya sea desde la línea de comandos, funciones de usuario o Internet, para su futura ejecución.

Banderas de estado del robot. Permite conocer en qué estado lógico se encuentra un robot.

4.2.2 Movimiento del robot

El movimiento del robot se controla por medio de cuatro ruedas las cuales están unidas al cuerpo del robot, con las que es posible simular dos movimientos rotación y traslación. Los movimientos de rotación se realizan alrededor del eje central del robot, con lo que se rota a la derecha y a la izquierda. Los movimientos de traslación permitidos son: traslación hacía delante y hacia atrás.

De las cuatro llantas del robot, sólo dos de ellas tienen movimiento: la derecha y la izquierda, las dos llantas restantes, trasera y delantera proporcionan estabilidad al robot. Para poder realizar los cuatro posibles movimientos del robot, movimiento hacia delante y atrás, giro a la izquierda y derecha, las llantas que se simulan (derecha e izquierda) en el robot, sólo tienen dos posibles movimientos: giro hacia delante y hacia atrás, con los cuales

se pueden realizar los cuatro posibles movimientos, en la Figura 4.3 se muestra la organización de las llantas del robot.

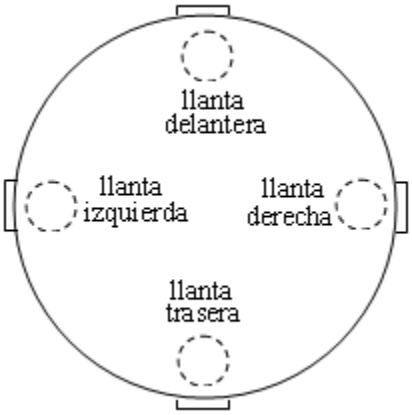


Figura 4.3 Vista inferior del Robot RTX8.

Para realizar los cuatro movimientos del robot es necesario mover hacia delante o hacia atrás las llantas derecha e izquierda según sea necesario, como se muestra a continuación.

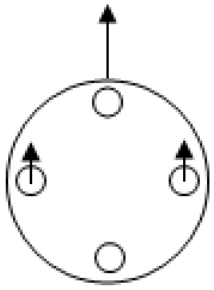


Figura 4.4 Moviendo hacia delante, las llantas derechas e izquierdas se mueven al mismo tiempo hacia delante.

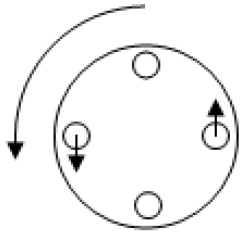


Figura 4.5 Rotación a la izquierda, la llanta izquierda se mueve hacia atrás y la llanta derecha hacia delante.

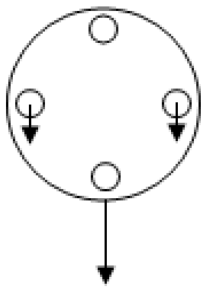


Figura 4.6 Moviendo hacia atrás, las llantas derechas e izquierdas se mueven al mismo tiempo hacia atrás.

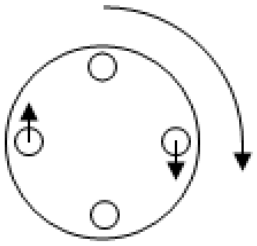


Figura 4.7 Rotación a la derecha, la llanta izquierda se mueve hacia adelante y la llanta derecha hacia atrás.

.

4.3 Tipos de sensores

Los sensores son los encargados de recibir las señales que serán las encargadas de estimular al robot.

Un sensor es un dispositivo capaz de transformar una magnitud ambiental física o química en una señal eléctrica, la cual puede ser manipulada por un dispositivo para representar la señal eléctrica en alguna variable cuantificable, ya sea distancia, presión, temperatura, etc.

Los sensores con los que cuenta el robot para las lecturas de su entorno son los siguientes.

Sensores de contacto.
Sensores reflectivos
Sensores infrarrojos.
Sensores de ultrasonido

4.3.1 Sensores de contacto

La finalidad de un sensor de contacto en un robot es determinar si algún obstáculo ha colisionado con el robot, considerando que un obstáculo es cualquier elemento tridimensional dentro del simulador, ya sea una geometría básica como una esfera, cubo, rectángulo, cono, cilindro o un modelo tridimensional, así como cualquier otro robot representado en la escena.

Las consideraciones que se tomaron en la versión anterior de Roc2 para la presentación de un sensor de contacto y de cualquier objeto dentro de la escena son las siguientes:

Los objetos son prismas de 6 caras.
Las cajas son convexas.
Los ángulos de las caras de los prismas son rectos
La normal de una cara está alineada a un eje coordenado del sistema.

La Figura 4.7 muestra un par de prismas que cuentan con las condiciones antes descritas.

Estas consideraciones garantizaban que todas las caras del cuerpo podían ser proyectadas sobre planos, cuyas normales son paralelas a los ejes del sistema de coordenadas, por lo que cada proyección mostrará dos caras del prisma, una cara oculta a las que son paralelas.

Con lo cual la detección de colisiones entre dos cuerpos se reducía a un problema de colisión de cajas orientadas o por sus siglas en ingles AABB (axis aligned bounding boxes).

A continuación se muestra un ejemplo extraído de [Her02] que muestra la detección de colisiones por medio del método AABB.

En la Figura 4.7 se muestran dos poligonos de x dimensiones que no colisionan, mientras que en la Figura 4.8, se muestran las respectivas proyecciones de los cubos sobre los ejes de coordenadas.

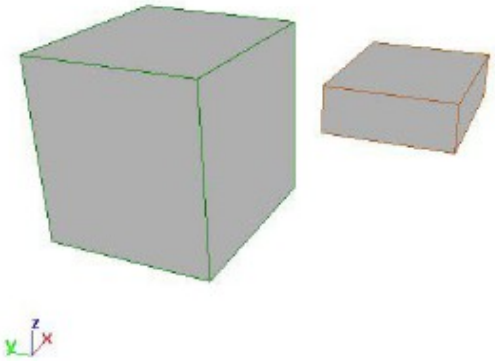


Figura 4.7 Prismas utilizados para representar objetos que pueden colisionar [Ref41].

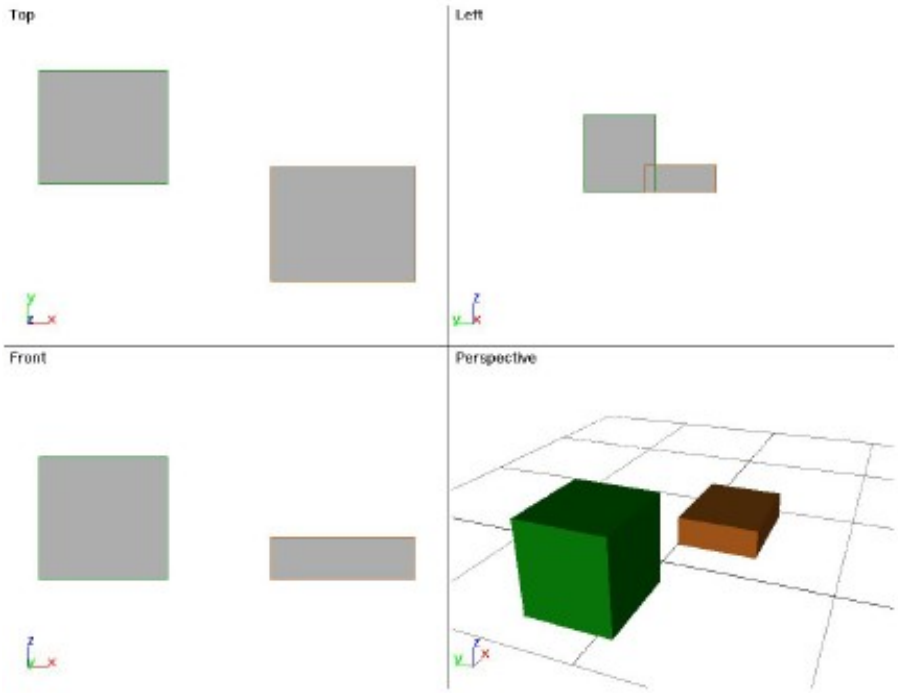


Figura 4.8 Proyecciones de los prismas [Ref41].

Las proyecciones de un prisma (P) sobre un plano, como el mostrado en la Figura 4.8, forman figuras planas de cuatro vértices (v); además, las componentes de cada vértice en el espacio 3D son de la forma (vx,vy,vz). Para determinar cuál es la proyección de un vértice (v) del prisma sobre un plano paralelo, ortogonal a un eje coordenado, se hace lo siguiente (ver tabla 4.1).

Plano ortogonal	Proyección (u,w)
X	(vy, vz)

Y	(vx, vz)
Z	(vx, vy)

Tabla 4.1 Proyección de un vértice sobre los planos paralelos.

Una vez que se obtienen las figuras planas que se forman al proyectar dos prismas sobre un plano, se determinará si ambas figuras se intersecan. Los datos con los que se cuenta hasta el momento son:

Vértices de la figura 1	vértices de la figura 2
V11	V21
V12	V22
V13	V23
V14	V24

Tabla 4.2 Datos de los vértices.

Donde Vab indica vértice b de la figura a (ver tabla 4.2).

Así, cada vértice Vij tiene dos componentes; los ejes pertenecientes a la proyección serán U y W. Para cada una de las figuras es necesario identificar cuáles serán las componentes Umínima, Umáxima, Wmínima y Wmáxima, como lo muestra la Figura 4.9.

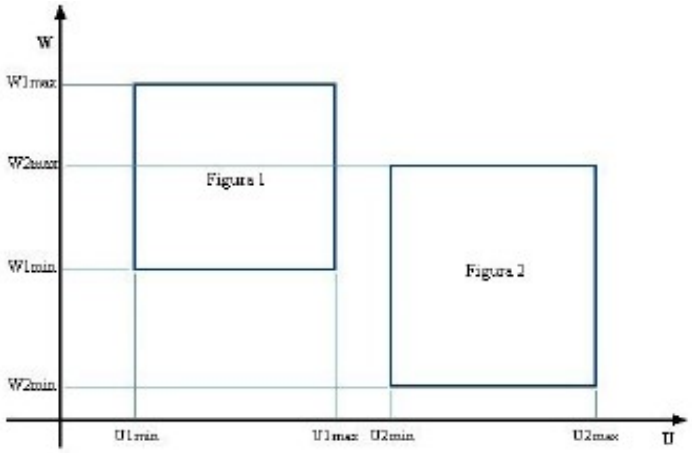


Figura 4.9 Determinación de la componentes U y W [Ref41].

Las figuras son proyectadas sobre líneas paralelas a los ejes U y W para obtener las componentes máximas y mínimas de cada eje (ver tabla 4.3).

Figura 1	Figura 2
U1min	U2min
U1max	U2max
W1min	W2min
W1max	W2max

Tabla 4.3 Componentes U y W de cada figura.

Utilizando los datos de la proyección de las figuras sobre los ejes U y W se pueden obtener los casos mostrados en la *Figura 4.10*; donde X es un eje arbitrario.

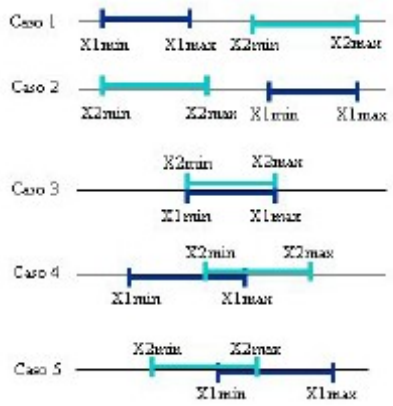


Figura 4.10 Casos de proyección de dos figuras sobre una recta [Ref41].

De la *Figura 4.9* se observa que si se detecta intersección en las proyecciones U y W, se considera que las figuras se encuentran intersecadas.

La siguiente condición refleja cuando una figura se interseca con otra haciendo uso de la proyección de sus vértices sobre los ejes.

```
If ( ( U1min ≤ U2min && U1max ≤ U2max && U2min ≤ U1max ) | |
        ( U2min ≤ U1min && U2max ≤ U1max && U1min ≤ U2max ) ) &&
        ( ( W1min ≤ W2min && W1max ≤ W2max && W2min ≤ W1max ) | |
        ( W2min ≤ W1min && W2max ≤ W1max && W1min ≤ W2max ) )
                Existe intersección
Else
                No hay intersección
End if
```

De manera práctica es necesario encontrar intersección en sólo dos proyecciones para garantizar que un cuerpo se interseca con otro.

Debido a que la detección de colisiones entre objetos en la versión anterior del sistema (Roc2) se realiza mediante el proceso antes descrito, no se necesita gran capacidad de procesamiento para estas operaciones y se ve reflejado en el rendimiento del simulador.

Pero existe un gran inconveniente en este algoritmo, cualquier objeto que se simule dentro de la escena deberá de ser representado por una caja mínima [Ref42], la cual encierra el modelo tridimensional y las colisiones se realizarán con ésta caja mínima, como se muestra en la *Figura 4.11*.

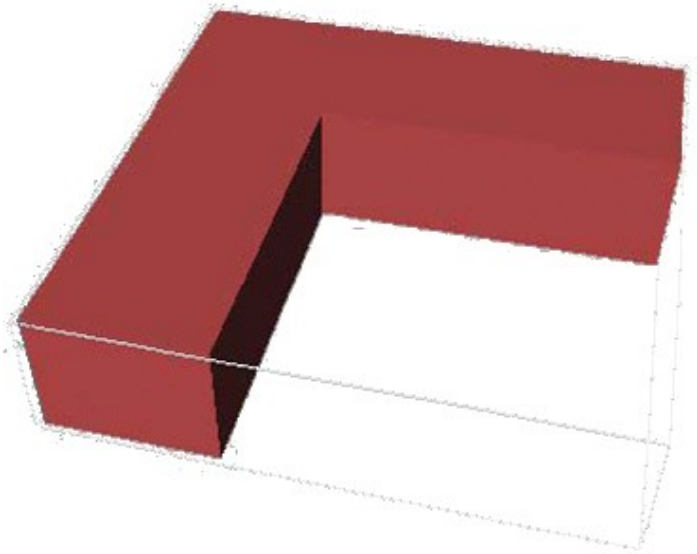


Figura 4.11 Prisma con volumen fantasma [Ref41].

Por lo que el inconveniente que se presenta durante la detección de colisiones, es que estas se realizan con la caja mínima, lo que produce colisiones con un objeto diferente del real o el que se esta simulando, como se muestra en la *Figura 4.12*.

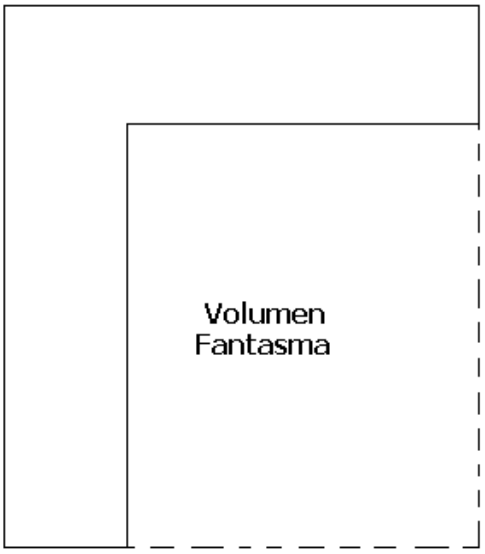


Figura 4.12 Vista superior del prisma con volumen fantasma.

Existen otras técnicas para resolver este tipo de problemas, como la de estructuras jerárquicas basadas en esferas, entre los cuales se encuentran el método de Philip M. Hubbard, y el método de Sean Quilan, que tienen la finalidad de tener una cantidad de representaciones del modelo con un cierto nivel de detalle, ya que para la primer representación se puede tener una esfera que contenga a todo el modelo, mientras que para el siguiente nivel de detalle se puede tener un mayor número de esferas que representen el objeto y así sucesivamente, como se muestra en la *Figura 4.13*.

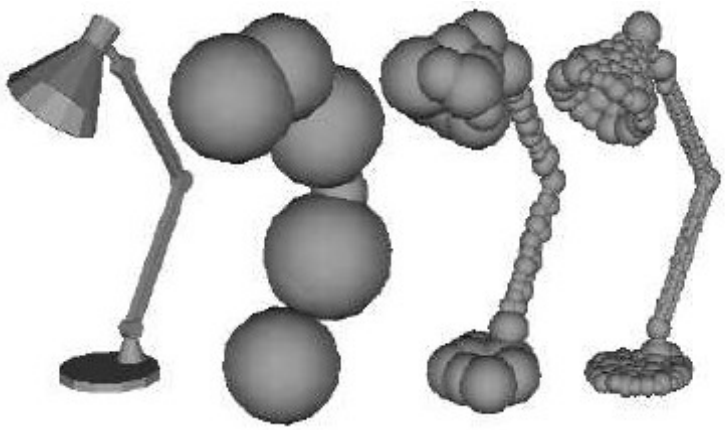


Figura 4.13 Lámpara con diferente estructura jerárquica basaba en esferas [Ref43].

Para solucionar este problema, la librería NGD incluye un sistema de detección de colisiones, por lo que el problema se reduce a integrar el sistema de colisiones al simulador. Para realizar la detección de colisiones entre dos objetos, NGD proporciona la función "NewtonCollisionCollide", que recibe una serie de datos, como son los dos objetos con los que se quiere verificar si existe colisión y las matrices que contiene la información de posición y orientación de los objetos. Si los dos objetos colisionan la función regresa el número de contactos que existe entre los dos objetos, de lo contrario regresa un cero. Por lo que el problema de detección de colisiones se reduce a verificar que al menos exista un punto de contacto entre los dos objetos.

A continuación se muestra el pseudocódigo para realizar la detección de colisiones entre dos geometrías.

```
if(NewtonCollisionCollide(…)>=1)
        El robot ha colisionado
else
        El robot no ha colisionado.
```

4.3.2 Sensores reflectivos

El funcionamiento de un sensor reflectivo consiste en emitir un haz de luz infrarroja sobre la superficie que se desea censar así como capturar el haz de luz reflejado por la superficie. Dependiendo de las propiedades físicas del material será la intensidad del haz reflejado sobre el sensor. Para superficies blancas el porcentaje del rayo reflejado será muy próximo al 100%, mientras que para superficies más obscuras el porcentaje ira disminuyendo acercándose a 0%, porcentaje reflejado de superficies negras.

Para poder obtener una lectura de una superficie por medio de un sensor reflectivo implica que se debe obtener el índice de refracción del material que se desea censar. Por lo tanto los robots deben de determinar el valor promedio de iluminación que es reflejada sobre un área de la pista.

Para el caso del simulador lo que se obtiene es el promedio de los pixeles adyacentes al punto de incidencia de luz emitida por el sensor dentro de una pista previamente establecida. Para nuestro caso el sensor se encuentra localizado en la parte baje del robot, como se muestra en la *Figura 4.14*.

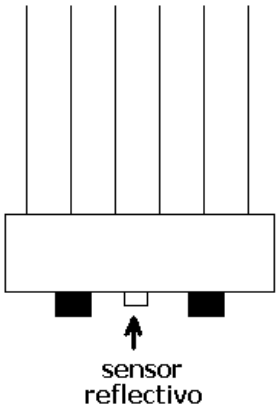


Figura 4.14 Sensor reflectivo del robot RTX8.

Para realizar la lectura de un sensor reflectivo dentro de una pista, se obtiene el promedio de los pixeles adyacentes al punto de incidencia de luz emitida por el sensor dentro de una pista previamente establecida.

Para obtener la posición de un sensor dentro de la pista como se muestra en la *Figura 4.15*.

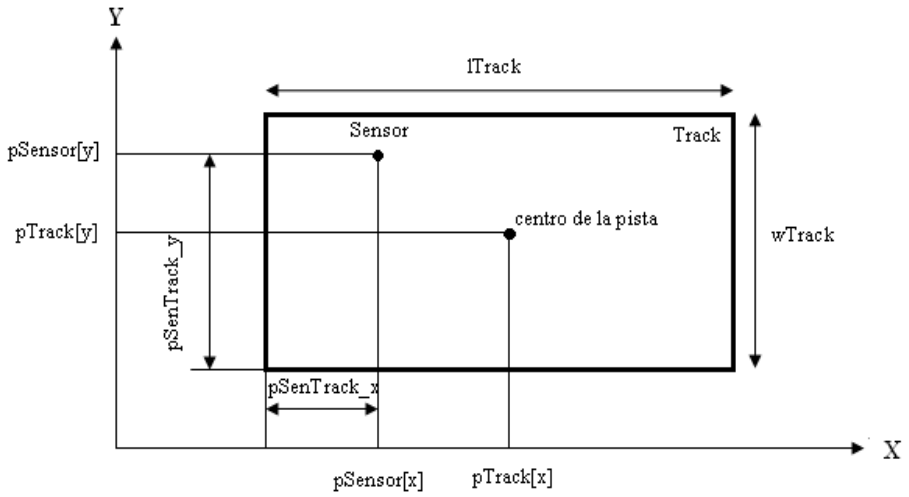


Figura 4.15 Obtención de la posición relativa del sensor reflectivo sobre el mundo virtual [Ref41].

Son necesarios los siguientes datos:

- Posición del sensor pSensor
- Posición de la pista pTrack
- Ancho de la pista wTrack
- Largo de la pista lTrack.

Para determinar si un sensor se encuentra dentro de la pista se usa el siguiente discriminante:

(pSensor[X] ≤ (pTrack[X]+lTrack/2)) && (pSensor[X] ≥ (pTrack[X]-lTrack/2))
&& (pSensor[Y] ≤ (pTrack[Y]+wTrack/2)) && (pSensor[Y] ≥ (pTrack[Y]-wTrack/2))

Para determinar la posición del sensor sobre la pista (pSenTrack_x, pSenTrack_x) se utiliza la siguiente ecuación de mapeo.

pSenTrack_x=-pSensor[X]+ pTrack[X]+lTrack/2;
pSenTrack_y= pSensor[Y] - pTrack[Y]+wTrack/2;

Una vez verificado que el sensor se encuentra dentro de la pista, se deben obtener los pixeles que se encuentran dentro del área del sensor, como se muestra en la *Figura 4.16*.

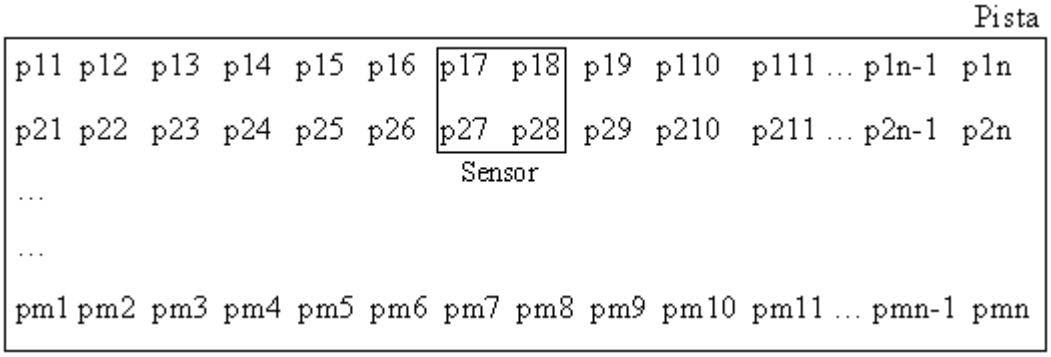


Figura 4.16 Representa del sensor sobre la pista.

Para calcular los píxeles que se tomarán en cuenta, para realizar la lectura, es necesario considerar las dimensiones del sensor y los píxeles por unidad, como se muestra a continuación.

pixelsX= sensorWidth* imaPixelsWidth/wTrack;
pixelsY= sensorLong * imaPixelsLong/lTrack;

Finalmente para obtener la lectura del sensor se considera un promedio de los píxeles que se encuentran dentro del área de sensor, y se le agrega ruido para simular un sensor real.

4.3.3 Sensores infrarrojos

Para la adecuada simulación de un sensor infrarrojo se requiere de un dispositivo emisor, éste es el encargado de emitir un haz de luz infrarroja, y un sensor infrarrojo que es el encargado de captar el haz de luz infrarroja emitida por el emisor. A continuación se describe como se realizó la simulación del emisor y el sensor infrarrojo.

4.3.3.1 Emisor de luz infrarroja

Como se mencionó anteriormente, un emisor es el encargado de emitir la señal (haz de luz infrarroja) que captara el sensor infrarrojo.

Físicamente un emisor de luz infrarroja es un diodo emisor de luz (LED), dispositivo semiconductor que al aplicarle una cierta corriente produce un espectro de luz, todo esto depende de los materiales de los cuales este compuesto el diodo.

Para el caso que se está estudiando, la simulación de un emisor dentro del ambiente virtual, se realiza a partir de un cono como se muestra en la *Figura 4.17*.

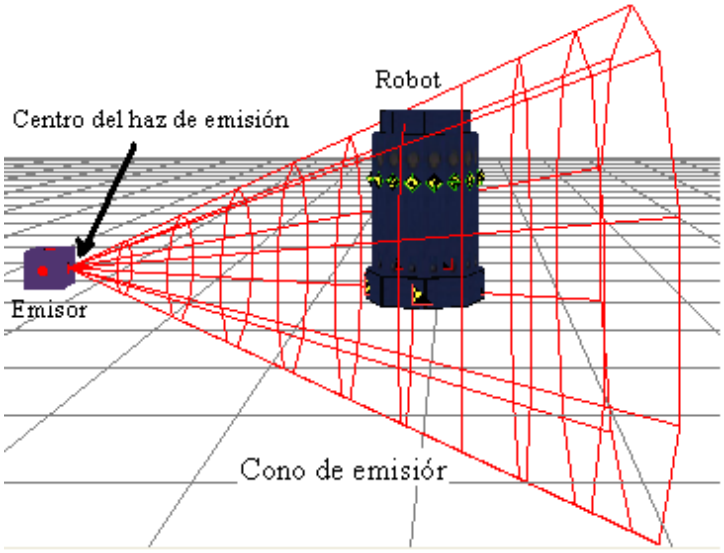


Figura 4.17 Cono de emisión de luz infrarroja.

Los rayos infrarrojos provenientes del emisor están delimitados y contenidos por el cono de emisión, que tiene un radio y una altura, como se muestra en la *Figura 4.18*.

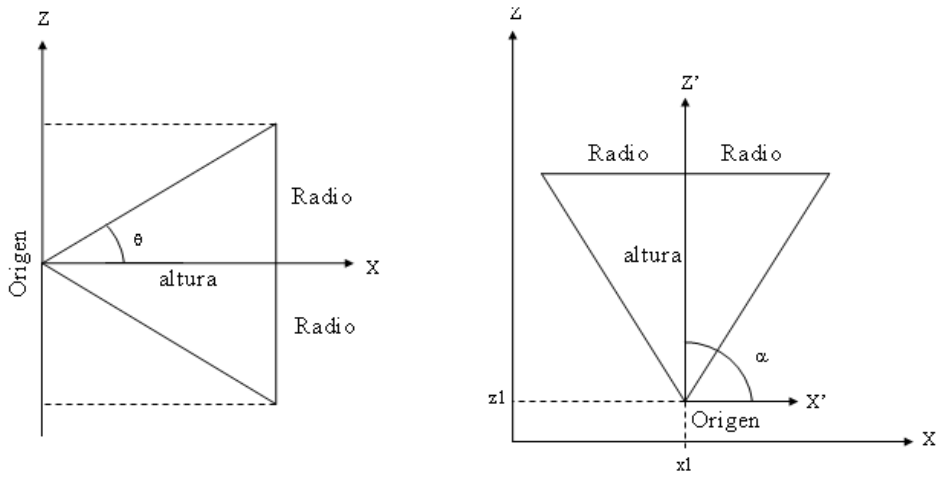


Figura 4.18 Esquema del cono de emisión de luz infrarroja.

Todo receptor que se encuentre dentro del cono de emisión, es susceptible a ser excitado por el emisor como se muestra en la *Figura 4.19*. La intensidad del haz de luz varia de acuerdo a la distancia del receptor con respecto al origen de emisión del haz infrarrojo, ya que un receptor que se encuentre a menor distancia del origen del haz de emisión tendrá un valor mayor de intensidad que uno que se encuentra a mayor distancia. El cálculo de la intensidad del haz infrarrojo se calcula de la siguiente forma.

$$I = \frac{h-d}{h}.$$

$$d = \left|\overline{OeR}\right| = \sqrt{(Oex-Rx)^2 + (Oey-Ry)^2 + (Oez-Rz)^2}.$$

d = distancia del origen de emisión del emisor al sensor del robot.

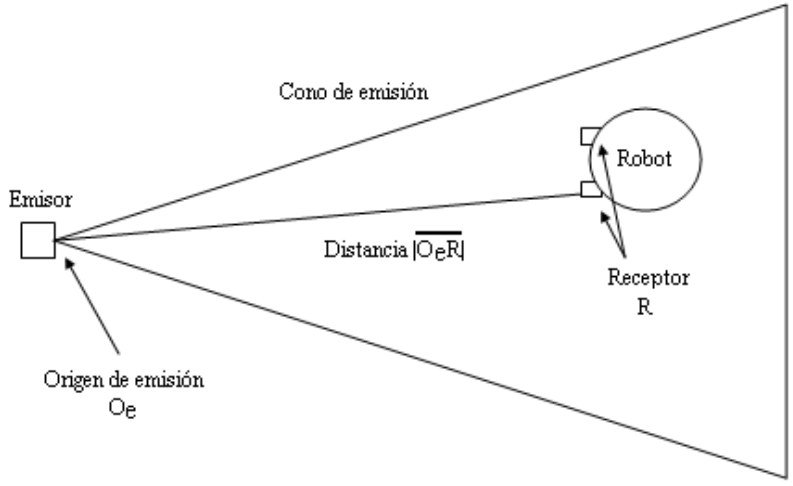


Figura 4.19 Diagrama del haz de luz emitida por el emisor de luz infrarroja.

4.3.3.2 Receptor de luz infrarroja

Físicamente un receptor de luz infrarroja (sensor) es un dispositivo semiconductor que al ser excitado con un cierto espectro de luz es capaz de representar la excitación por medio de una diferencia de corriente.

Este dispositivo es el encargado de censar el medio ambiente para captar cualquier estimulo proveniente de un emisor de luz infrarroja y transformarlo en una diferencia de voltaje, y el robot se encargara de interpretarlo para saber la distancia a la que se encuentra el emisor.

Los receptores de luz infrarroja dentro del simulador están representados por medio de un cubo montado en el robot como se muestra en la *Figura 4.20*. Para poder realizar la lectura del sensor es necesario conocer su posición y orientación dentro de la escena virtual, con lo

que se determinara si el receptor esta dentro del cono de emisión y si el ángulo de visión es el adecuado para realizar la lectura.

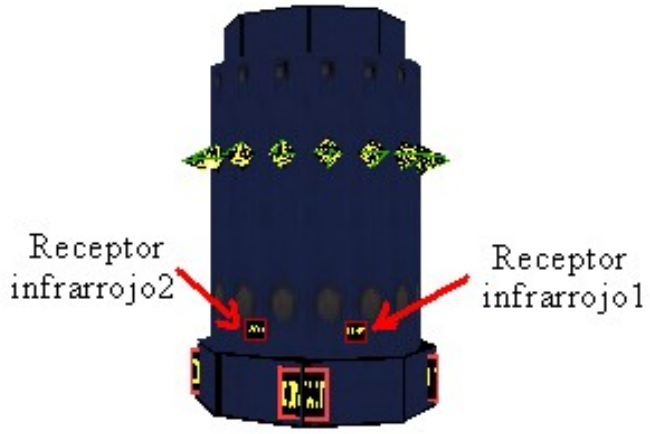


Figura 4.20 Receptores de luz infrarroja.

El primer paso que se debe realizar para poder tomar una lectura del receptor (sensor infrarrojo) es validar que el receptor se encuentre dentro del cono de emisión de luz infrarrojo.

El ángulo de visión del receptor: es el ángulo entre el vector director del emisor y el vector director del receptor de haz infrarrojo, como se muestra en las *Figuras 4.21* y la *Figura 4.22*.

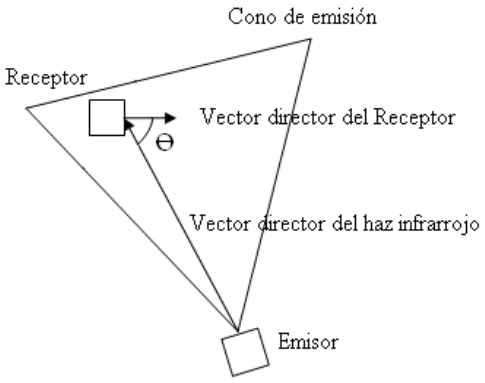


Figura 4.21 Vista superior del cono de emisión del emisor de luz infrarroja.

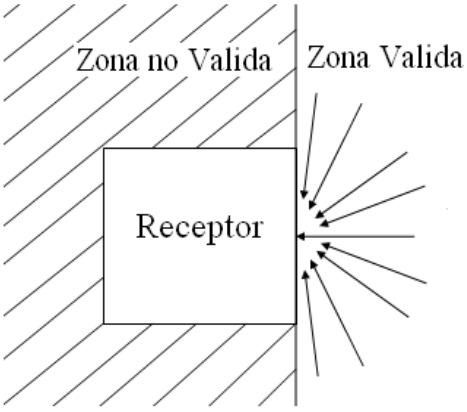


Figura 4.22 Zonas validas para la recepción del haz de luz infrarroja.

El ángulo valido entre el vector director del receptor y el vector director del haz infrarrojo se encuentra entre 0^o y 90^o, el ángulo entre los dos vectores se calcula de la siguiente forma.

$$\cos(uv) = \frac{u \cdot v}{|v||u|}.$$

Donde:

u = vector director del receptor
v = vector director del haz infrarrojo.

Para validar el ángulo de recepción de sensor de luz infrarroja se utiliza el siguiente discriminante:

$$((\text{ángulo} > 0^o) \ \&\& \ (\text{ángulo} < 90^o))$$

Para detectar si el receptor de luz infrarroja se encuentra dentro del cono de emisión de luz infrarroja se hace uso de la función "NewtonCollisionCollide", que sirve para verificar si dos objetos colisionaron, en este caso los objetos son el receptor de luz infrarroja y el cono de emisión de luz infrarroja.

4.3.4 Sensores de ultrasonido

La mayoría de los sensores de ultrasonido se basan en dos etapas, la primera de ellas es la emisión de un pulso de alta frecuencia generalmente de 40 kHz (ultrasonido), y cuyo campo de acción es de forma cónica, la segunda consiste en medir el tiempo entre la emisión del sonido y la percepción del eco, con lo que se puede conocer la distancia a la que se encuentran los objetos con la siguiente formula.

$$d = \frac{Vt}{2}.$$

Donde:

V es la velocidad del sonido en el aire
t es el tiempo trascurrido entre la emisión y recepción del pulso.

Pasa simular un sensor de ultrasonido se considera lo siguiente:

- Los sonares tenían forma cilíndrica.
- El campo de acción del sonar se limita a un rayo, se definía mediante un punto fijo y una dirección, donde el punto fijo es la posición del sonar en la escena, y la dirección es la orientación del sonar en el instante de la lectura.
- Para calcular qué elemento está enfrente del robot, se prueban intersecciones entre el rayo y todos los polígonos de los posibles obstáculos con los que el robot puede colisionar. Los obstáculos con los que se podría colisionar son todos aquellos elementos que se encuentren en la escena.

- La lectura entregada por el sonar representa la distancia que hay de la posición del sonar al polígono del obstáculo más próximo al robot.

La ventaja que se tiene al utilizar una línea como representación del haz de acción del sonar es que la detección de colisiones entre una línea y un objeto tridimensional, es más eficiente y rápida, ya que la operación se reduce a realizar una detección de colisión entre una línea y un plano como se muestra en la *Figura 4.23*.

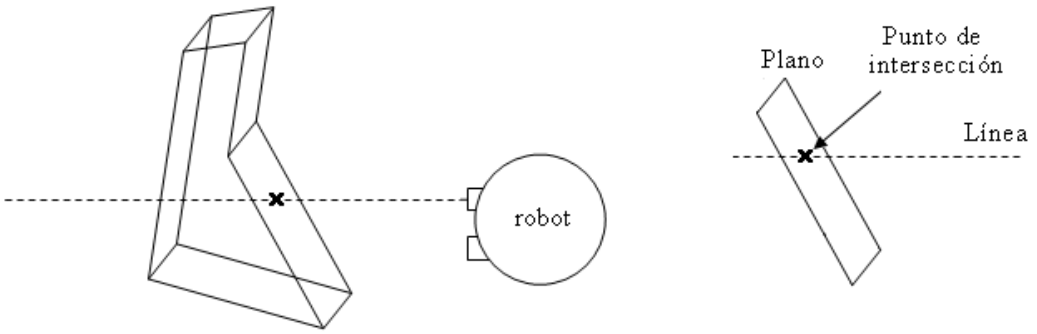


Figura 4.23 Detección de colisión del rayo y el plano de un obstáculo.

La desventaja que se presenta al utilizar una línea como haz de acción de un sonar es que no siempre se puede encontrar el objeto mas cercano al sensor, ya que se puede tener un objeto que colisiona y se encuentra a una distancia Dx, pero se puede encontrar otro obstáculo a menor distancia pero que no colisiona con la línea que representa el haz de acción del sonar como se muestra en la siguiente *Figura 4.25*.

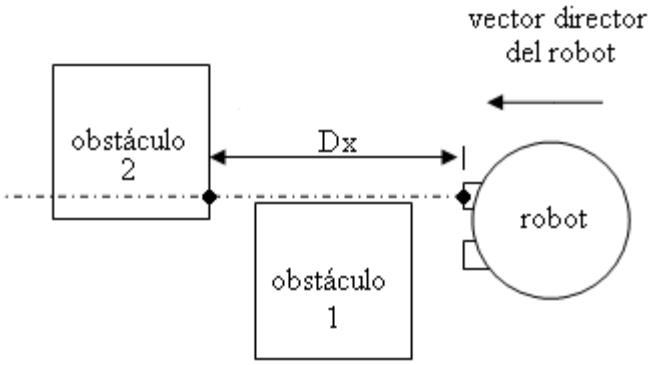


Figura 4.24 Detección del obstáculo mas próximo.

Para detectar que objeto se encuentra a menor distancia del sonar, se verifica que objetos colisionan con una línea de longitud infinita, cuyo origen se encuentra en el centro del sonar, del que se toma la lectura como se muestra en la *Figura 4.25*.

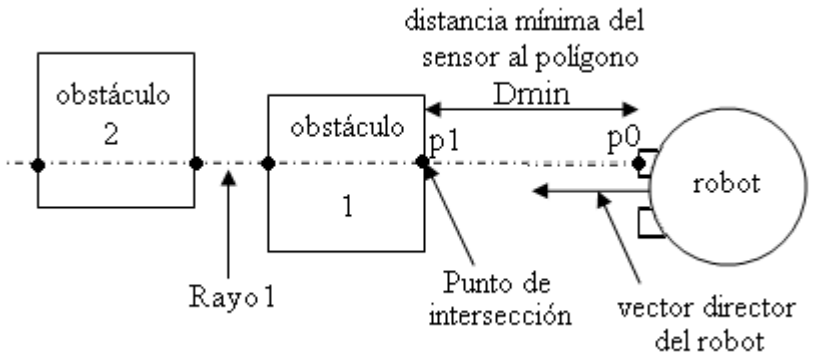


Figura 4.25 Detección de colisión del rayo del sonar con los obstáculos.

Como se puede observar en la *Figura 4.25*, la distancia mínima (Dmin) se encuentra entre p0 y p1 que se obtiene al colisionar el Rayo 1 con el obstáculo 1. Por lo que para obtener la lectura del sonar, sólo es necesario calcular la distancia entre el punto p0 y p1.

A continuación se encuentra el pseudocódigo utilizado para realizar la lectura del sonar.

```
Float getNearObject(numSonar){
        Obtener matriz de transformaciones del sonar (matriz)
        Obtenemos la posición del punto origen de la línea (matriz, p0)
        Obtenemos la posición del punto destino de la línea (matriz, p1)
        Obtenemos la lista de objetos que colisionan con la línea (lista, p0, p1)
        Obtenemos el objeto más cercano (objeto)
        Obtenemos el punto de intersección de la línea con el objeto (p0,p1,p2);
        Calculamos la distancia entre p0 y p1
}
```

La lectura del sensor se obtiene al calcular la distancia de p0 que se encuentra sobre el sonar y el punto p1, punto de intersección entre la línea que representa el haz del sonar y el objeto más cercano al sonar.

$$\text{Distancia} = \sqrt{(p1x - p0x)^2 + (p1y - p0y)^2}.$$

Capítulo V

Obstáculos

5.1 Introducción

Un obstáculo es cualquier objeto estático o dinámico que impida el libre movimiento de un Robot dentro del mundo virtual. Por lo que todo objeto que se encuentre en el mundo virtual se considerara como un obstáculo.

Un objeto estático, es aquel que al aplicarle un vector fuerza de amplitud y magnitud cualquiera reacciona con un vector fuerza con igual amplitud y dirección contraria al vector que se le aplico, cancelándose las fuerzas e impidiendo que el objeto se mueva de su posición inicial.

Un objeto dinámico, es aquel objeto que al aplicarle un vector fuerza de amplitud y magnitud cualquiera reaccionara moviéndose en la dirección que se aplico la fuerza, modificando su posición inicial.

Los obstáculos que existen en la escena se pueden clasificar de la siguiente forma:
Obstáculos tipo geometría básica.
Obstáculos tipo modelo 3DS.
Obstáculos tipo mapa.
Obstáculos tipo pista.

5.2 Obstáculos tipo geometría básica

Son geometrías básicas como esferas, cubos, rectángulos, cilindros y conos, que pueden tener cualquier dimensión y pueden ser estáticos o dinámicos, como se muestra en la *Figura 5.1*.

Figura 5.1 Geometrías básicas.

5.3 Obstáculos tipo modelo 3DS

Son modelos hechos con algún software de modelado tridimensional, que son guardados en un archivo con formato 3ds. Estos modelos son cargados en el mundo virtual a través del archivo en formato 3ds, la carga de los obstáculos se realiza mediante una librería desarrollada por Matthew Fairfax para la carga de modelos 3ds. Por lo que para poder cargar obstáculos con este tipo de formato se tienen las siguientes consideraciones.

Sólo puede existir un objeto dentro del archivo 3ds.
El modelo no debe de tener un sistema de luces.
No debe tener cámaras.

Una vez cargado el obstáculo a partir del archivo 3ds, es necesario crear el cuerpo de colisión (modelo utilizado para verificar colisiones con otros objetos) con la que se creara el cuerpo rígido (modelo utilizado para realizar la simulación de física) del modelo. Esta cuerpo de colisión se crea por medio de funciones de la librería NGD, debido a las propias restricciones de la librería no se pueden crear objetos dinámicos cóncavos, únicamente objetos estáticos cóncavos, así que cuando se crea un objeto cóncavo a través de una modelo en 3D, la librería hace un ajuste y genera un objeto convexo.

Decimos que un objeto es convexo, si y solo si dados dos puntos cualesquiera P_1 y P_2 que se encuentren dentro del objeto, al trazar una línea del punto P_1 al punto P_2 esta no cruza ninguna cara del objeto, como se muestra en la *Figura 5.2*.

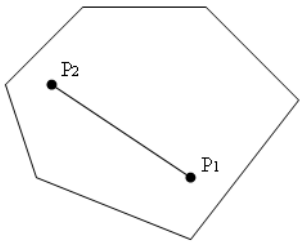


Figura 5.2 Polígono convexo.

Se dice que un objeto es cóncavo, si y solo si dados dos puntos cualesquiera P_1 y P_2 que se encuentren dentro del objeto, al trazar una línea del punto P_1 al punto P_2 esta cruza al menos por una cara del objeto, como se muestra en la *Figura 5.3*.

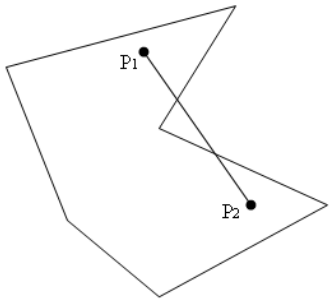


Figura 5.3 Polígono cóncavo.

Para ilustrar esto, se puede observar el siguiente modelo cóncavo de una nave espacial en la *Figura 5.4*.

Figura 5.4 Modelo de una nave espacial.

El modelo de la nave espacial de la *Figura 5.4* es un objeto cóncavo, por lo que la librería NGD al crear su cuerpo rígido, creará una malla mínima que contendrá el modelo tridimensional y esta malla se utilizara para crear el objeto de colisión del sistema, como se muestra en las *Figuras 5.5* y *5.6*.

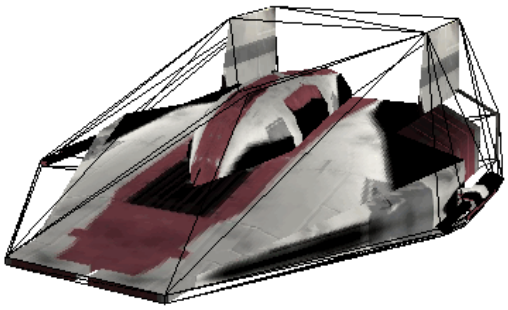


Figura 5.5 Modelo de la nave con su malla mínima.

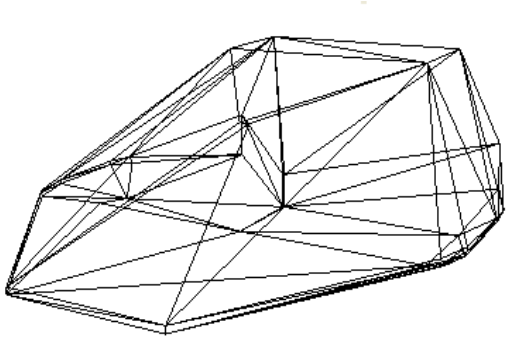


Figura 5.6 Malla mínima de la nave.

5.4 Obstáculos tipo mapa

Dentro del simulador es posible definir un ambiente virtual por medio de un mapa, los mapas pueden estar compuestos por geometrías básicas como cubos, rectángulos y objetos tridimensionales definidos por reglas preestablecidas, que impedirá el libre movimiento de los robots, los mapas se consideran como un conjunto de obstáculos agrupados en un obstáculo tipo mapa. Los mapas se guardan en archivos de texto que describen los elementos que contienen el ambiente virtual, así como la posición y dimensiones de los objetos que lo conforman.

Las ventajas que se tienen al utilizar este tipo de archivos para describir entornos virtuales son:

No es necesario el modelado 3D para la creación de los entornos.
Menor consumo de recursos.
No es necesario software de terceros para la creación o modificación de los entornos.

5.4.1 Formato de los mapas

Los mapas están formados por directivas, cuyas reglas son las siguientes.

- Cada directiva debe de estar contenida entre paréntesis.
- Se debe indicar a que mundo virtual pertenece.
- Solo es posible definir una directiva por línea.
- Si los datos de la directiva son erróneos o incompletos, no se cargara el mapa.
- Los comentarios deberán iniciar con (;).

Todo mapa debe tener las siguientes tres directivas:

- limit_area (límite del contorno).
- dimensions (dimensiones).

- middle-point (punto medio).

limit-area. Indica la posición de cada uno de los vértices que forman el límite o contorno del ambiente virtual. Los vértices se deben especificar siguiendo el sentido de las manecillas del reloj. Además es necesario indicar el nombre del ambiente, como se muestra a continuación:

(limit_area LII 0.00 0.00 0.00 77.00 74.50 77.00 74.50 0.00).

En este caso, el nombre del ambiente es LII y sus dimensiones están dadas por los vértices que le siguen. Estos vértices forman un polígono cuadrado cuyos vértices son:

v1 = (0.00, 0.00),
v2 = (0.00, 77.00),
v3 = (74.50, 77.00),
v4 = (74.50, 0.00).

dimensions. Indica el largo y el ancho del ambiente virtual, como se muestra a continuación:

(dimensions LII 74.50 77.00).

En este ejemplo se definió las dimensiones para el ambiente virtual LII; los valores de largo y ancho aparecen después del nombre del ambiente.

middle-point. Contiene el punto medio del ambiente virtual, como se muestra a continuación:

(middle-point LII 5.0 3.0).

En este ejemplo se definió el punto medio para el ambiente virtual LII, que se encuentra en el punto (5,3).

5.4.2 Creación de elementos geométricos

Para crear objetos dentro del ambiente virtual se utiliza la directiva *polygon (polígono)*. El formato de esta directiva permite especificar los vértices que forman parte de la base de un prisma recto, su formato se describe a continuación:

(polygon tipo ambiente cuarto vertex1 vertex2....)

donde:

tipo	indica el tipo de elemento geométrico que se está creando, usando este parámetro es posible indicarle al sistema que el elemento a crear se encuentra en la clasificación indicada.
cuarto	indica a qué entorno virtual pertenece el elemento.
nombre	indica el nombre del elemento dentro del mundo virtual.

vertex especifica los vértices del polígono que forman al objeto, ordenados siguiendo el sentido de las manecillas del reloj.

Utilizando las propiedades tipo y nombre, podremos crear una organización lógica para representar el ambiente virtual. Es posible crear cuartos diferentes con el mismo nombre del elemento; por ejemplo, podríamos especificar un elemento silla1 para la cocina y también un elemento silla1 para la sala, como se muestra a continuación:

```
(polygon cocina LII silla1 1.50 22.90 1.50 30.50 32.00 30.50 32.00 22.90).
(polygon sala LII silla1 1.50 31.50 1.50 61.40 8.90 61.40 8.90 31.50).
```

Por último, es posible añadir puntos de referencia dentro de los mapas del mundo virtual, el formato de la directiva para los puntos de referencia es el siguiente:

```
(reference-point cuarto posX posY).
```

donde:

cuarto indica el nombre del cuarto al que pertenece el punto de referencia.
posX, posY indica la posición del punto de referencia dentro del cuarto.

Es importante mencionar que los archivos de descripción utilizados en el simulador también son utilizados en un sistema experto CLIPS [Ref44], para tener una representación simbólica del medio ambiente en donde se navega.

El siguiente archivo es un ejemplo de un mapa que representa el Laboratorio de Interfaces Inteligentes de la Facultad de Ingeniería de la UNAM, en la *Figura 5.7* se muestra la representación tridimensional del mapa.

```
;*****************************************************************
;* File: lii.wrl                                                 *
;* Purpose: Definition of the forbidden and allowed areas in the Robot's  *
;* world. These areas are derivated from the objects in the      *
;* world.                                                        *
;*****************************************************************
(limit_area    LII   0.00  0.00  0.00 77.00 74.50 77.00 74.50  0.00)
(dimensions LII 74.50 77.00)
(middle-point LII 5. 3.)
(reference-point LII 10.0 10.0)
(reference-point LII 60.0 10.0)
(reference-point LII 20.0 50.0)
(reference-point LII 60.0 50.0)

(polygon object-wall LII mesa_1 1.50 22.90 1.50 30.50 32.00 30.50 32.00 22.90)
(polygon object-wall LII mesa_2 1.50 31.50  1.50 61.40  8.90 61.40  8.90 31.50)
(polygon object-wall LII mesa_3 6.40 68.20  6.40 75.50 36.40 75.50 36.40 68.20)
(polygon object-wall LII mesa_4 42.00 68.20 42.00 75.50 73.00 75.50 73.00 68.20)
(polygon object-wall LII mesa_5 65.40  31.20 65.40 61.60 73.00 61.60 73.00 31.20)
(polygon object-wall LII mesa_6 42.70 22.90 42.70 30.50 73.00 30.50 73.00 22.90)
```

(polygon object-wall LII mesa_7 36.50 45.50 36.50 74.30 42.50 74.30 42.50 45.50)
(polygon object-wall LII est_1 6.40 67.50 1.50 67.50 1.50 75.00 6.40 75.00)
(polygon object-wall LII est_2 65.40 66.30 73.00 66.30 73.00 61.60 65.40 61.60)

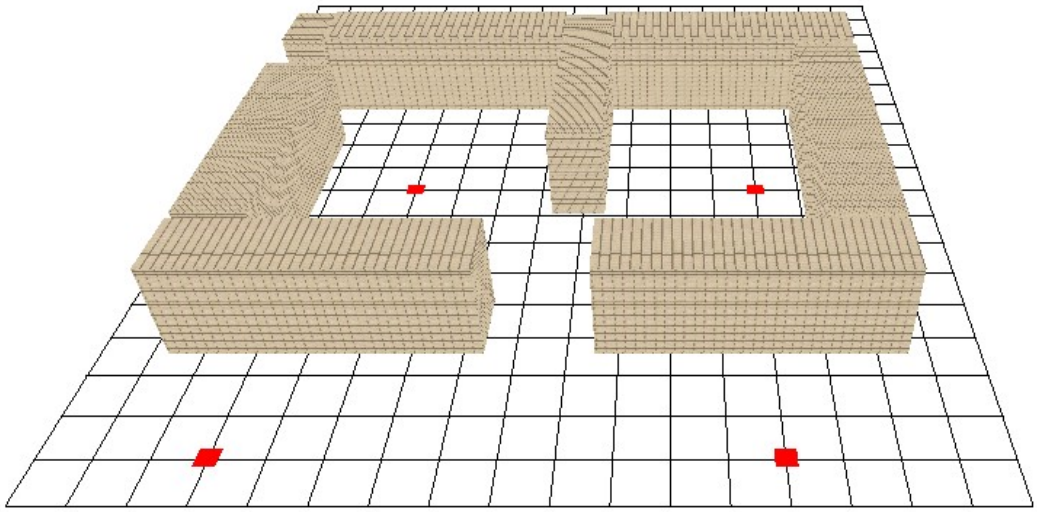


Figura 5.7 Cuarto formado por el archivo lii.wrl

5.5 Pistas

Las pistas son mapas de bits elaborados por el usuario, su representación dentro del mundo virtual es un plano a nivel del suelo que muestra la imagen de la pista. Como se muestra en la *Figura 5.8*.

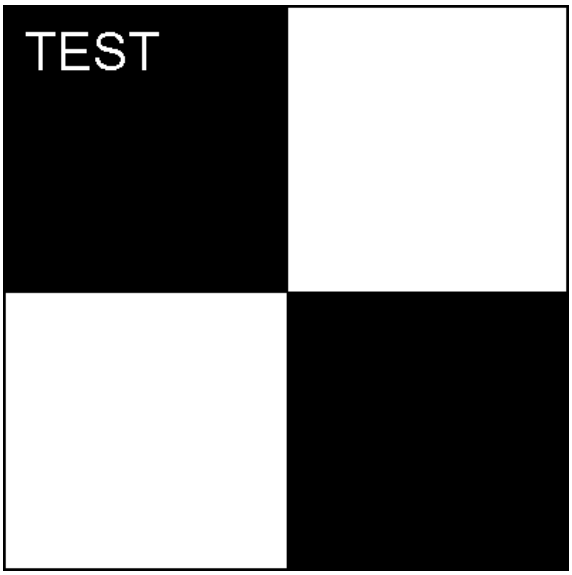


Figura 5.8 Ejemplo de una pista [Ref41].

Debido a que una pista requiere una representación visual dentro del simulador, es necesaria una estructura lógica que permita manipular la información contenida en el mapa de bits de la pista.

Estructura lógica:

- Nombre. Es con el que se identifica la pista dentro del mundo virtual
- Nombre de la imagen. Es el del archivo que contiene el mapa de bits que forma a la pista.
- Dimensiones de la imagen. Tamaño en píxeles del mapa de bits (imagen) asociado a la pista.
- Dimensiones de la pista. Dentro del mundo virtual la pista puede o no tener el mismo tamaño que el mapa de bits; por ello, es necesario almacenar el tamaño de la pista independientemente del tamaño del mapa de bits.
- Posición. Posición de la pista dentro del mundo 3D.
- Dimensiones del sensor. Son las dimensiones mínimas que debe tener el sensor utilizado para realizar una lectura sobre la pista, que corresponde al área sobre el mapa de bits, que se considera para calcular el porcentaje de luz reflejada por la superficie (pista).

5.5.1 Creación de una pista

Para crear una pista dentro del mundo virtual es necesario seguir cuatro pasos, como se muestra a continuación.

Figura 5.9 proceso para crear una pista.

Validación de la imagen. El mapa de bits que va formar la pista debe ser analizado para verificar que cumpla con las especificaciones requeridas; es decir, la imagen debe estar

dada en formato PNG, JPEG, GIF, TIFF, XPM y TGA y el tamaño de la imagen en pixeles debe de ser de $2^n x 2^m$ donde n y m son un número par.

Obtención de la matriz de datos. Para poder realizar lecturas en la pista es necesario conocer la información de los pixeles, como color y posición dentro de la imagen y cuyos datos se guardarán en una matriz que contendrá todos los pixeles de la imagen.

Obtención de los datos de la pista. Una vez guardada la matriz de pixeles de la imagen es necesario guardar los datos lógicos antes mencionados para poder realizar futuras lecturas sobre la pista.

Creación del modelo gráfico. Una vez que se guardaron todos los datos necesarios para realizar futuras lecturas sobre la pista, es necesario crear un modelo visual para mostrarlo en la simulación, mediante un polígono con la textura de la imagen seleccionada para la pista.

Para conocer con mayor detalle cómo se realizan las lecturas sobre una pista ir al tema 4.3.2 (Sensores reflectivos).

Capítulo VI

Pruebas del sistema

Para verificar el buen funcionamiento del sistema Roc2008 fue necesario realizar una serie de pruebas que consisten en:

1. Probar el conjunto de instrucciones de Robel sobre el robot virtual de manera local y remota.
2. Probar las interfaces que permiten interactuar con el sistema, como mouse y teclado.
3. Probar la interacción de las funciones de usuario con el sistema.

Las pruebas que se realizaron al conjunto de instrucciones de Robel consistieron en probar de forma individual cada instrucción y en conjunto por medio scripts, de manera local y remota. De manera local las instrucciones se probaron introduciendo cada una de ellas por medio del teclado y verificando que el robot ejecutara las acciones correctas. Las pruebas

de forma remota consistieron en enviarle las instrucciones al sistema desde otra computadora a través de Internet.

Las pruebas de interacción del usuario con el sistema a través del mouse y teclado consistieron en, manipular la interfaz del sistema por medio del teclado y mouse. La interacción con el sistema a través del teclado, consistió en introducir instrucciones a la línea de comandos y verificar que las instrucciones se realizaran. Las pruebas del mouse con el sistema consistieron, en manipular los puertos de vista, los menús desplegables, y la cámara libre que nos permite moverse por el mundo virtual, y verificar que se realizaran las acciones solicitadas.

Por último se verificó la interacción del sistema a través de las funciones de usuario, mandando instrucciones al robot a través de las funciones de forma local como de forma remota haciendo uso de sockets, que el mismo sistema proporciona.

Las pruebas que se realizaron de forma local y remota se realizaron sobre cinco equipos con distintos sistemas operativos, con lo que se verificó que el sistema desarrollado fuese soportado en diferentes plataformas, las características de los equipos utilizados para las pruebas se muestran a continuación.

Equipo 1.

Sistema Operativo: Windows 2000
Procesador: Celeron 1.8 Ghz
Memoria: 128 MB
Tarjeta de Gráficos: Tarjeta integrada Intel 82845G con 32MB

Equipo 2.

Sistema Operativo Wndows XP
Procesador: Pentium IV Duo1.6 Ghz
Memoria: 1 GB
Tarjeta de Gráficos: Tarjeta integrada Intel 945GM con 128MB

Equipo 3

Sistema Operativo: Ubuntu 8.04 LTS
Procesador: Pentium IV Core Duo 1.6 Ghz
Memoria: 1 GB
Tarjeta de Gráficos: Tarjeta integrada Intel 945GM con 128MB

Equipo 4

Sistema Operativo: Ubuntu 6.06 LTS
Procesador: Pentium IV Core Duo 1.6 Ghz
Memoria: 1 GB
Tarjeta de Gráficos: Tarjeta integrada Intel 945GM con 128MB

Equipo 5

Sistema Operativo Wndows Vista Bussiness
Procesador: Intel Core Duo 2.33 Ghz
Memoria: 2 GB
Tarjeta de Gráficos: Tarjeta integrada Intel Q35 con 128MB

Además de realizar pruebas individuales sobre cada comando para verificar su correcto funcionamiento de forma local y remota, también se probaron dos scripts que fueron utilizados para realizar pruebas sobre la versión anterior Roc2 y que ahora se probaron sobre esta nueva versión. Los script utilizados se muestran a continuación.

Script 1. Consiste en recorrer la siguiente trayectoria mostrada en la *Figura 6.1* dentro del ambiente virtual del Laboratorio de Interfaces Inteligentes.

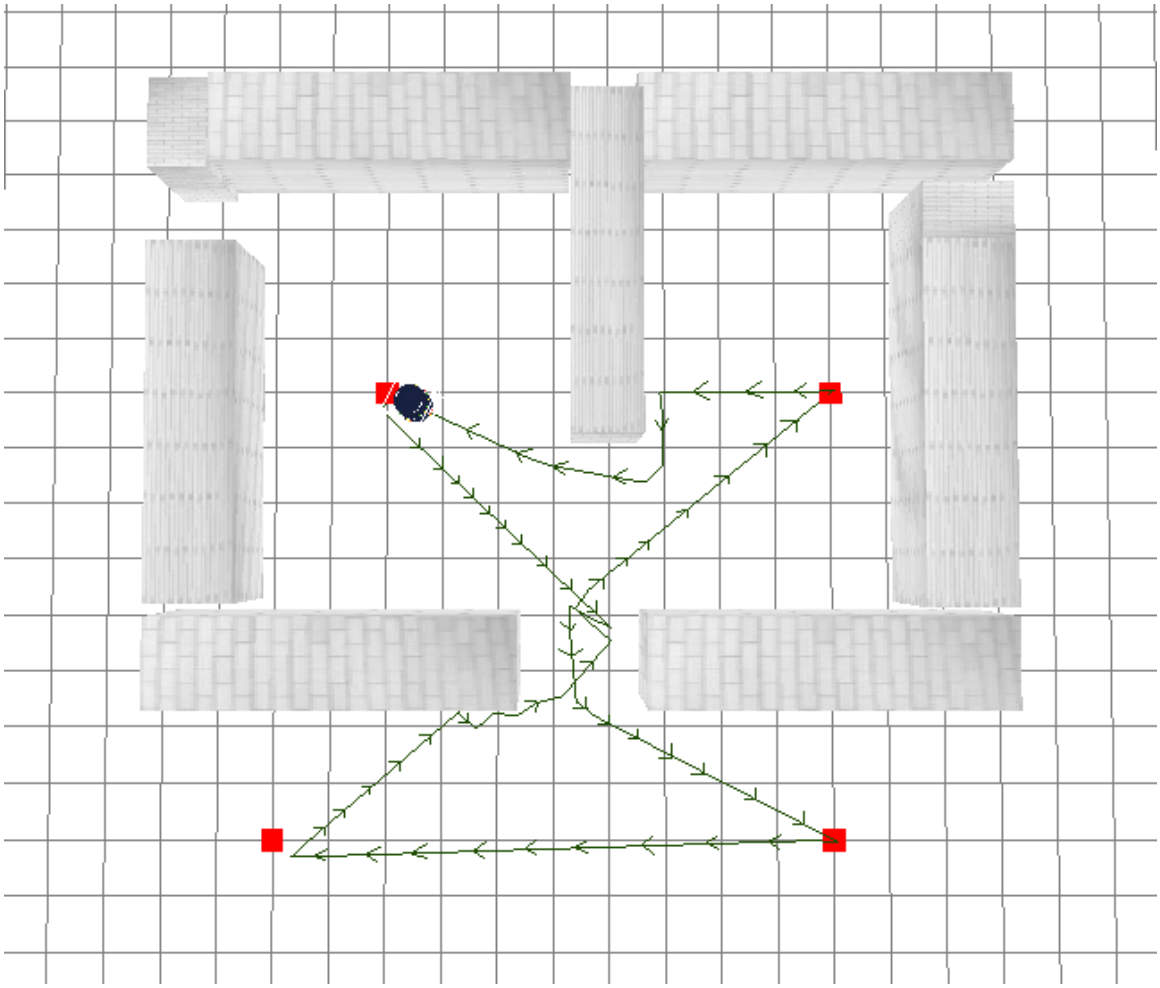


Figura 6.1 Trayectoria del Robot definida por el script 1

El script utilizado para recorrer esta trayectoria es el siguiente.

putbot 20.5 49.5 0.0 @0
mvto 19.397 48.489 1 @1
mvto 20.993 46.976 1 @2
mvto 22.590 45.462 1 @3
mvto 24.187 43.949 1 @4
mvto 25.783 42.435 1 @5
mvto 27.380 40.922 1 @6
mvto 28.977 39.408 1 @7
mvto 30.573 37.895 1 @8
mvto 32.170 36.381 1 @9
mvto 33.766 34.868 1 @10
mvto 35.363 33.354 1 @11
mvto 36.960 31.841 1 @12
mvto 38.582 30.354 1 @13
mvto 40.186 28.849 1 @14
mvto 40.189 28.846 1 @15
mvto 40.192 28.842 1 @16
mvto 40.194 28.839 1 @17
mvto 40.197 28.835 1 @18
mvto 40.200 28.832 1 @19
mvto 40.203 28.828 1 @20
mvto 40.205 28.825 1 @21
mvto 40.208 28.822 1 @22
mvto 40.211 28.818 1 @23
mvto 40.213 28.815 1 @24
mvto 40.216 28.811 1 @25
mvto 38.234 29.765 1 @26
mvto 36.344 30.891 1 @27
mvto 36.474 28.695 1 @28
mvto 36.672 26.504 1 @29
mvto 36.871 24.313 1 @30
mvto 37.071 22.122 1 @31
mvto 38.875 20.863 1 @32
mvto 40.836 19.865 1 @33
mvto 42.793 18.861 1 @34
mvto 44.749 17.854 1 @35
mvto 46.705 16.846 1 @36
mvto 48.661 15.839 1 @37
mvto 50.617 14.832 1 @38
mvto 52.573 13.825 1 @39
mvto 54.528 12.818 1 @40
mvto 56.484 11.810 1 @41
mvto 58.440 10.803 1 @42
*DESTINE REACHED 1
mvto 60.396 9.796 1 @43
mvto 57.801 9.931 1 @44
mvto 55.602 9.861 1 @45
mvto 53.403 9.792 1 @46
mvto 51.204 9.722 1 @47
mvto 49.005 9.653 1 @48
mvto 46.807 9.583 1 @49
mvto 44.608 9.514 1 @50
mvto 42.409 9.445 1 @51
mvto 40.210 9.375 1 @52
mvto 38.011 9.306 1 @53
mvto 35.812 9.236 1 @54
mvto 33.613 9.167 1 @55
mvto 31.414 9.097 1 @56
mvto 29.215 9.028 1 @57
mvto 27.016 8.958 1 @58
mvto 24.818 8.889 1 @59
mvto 22.619 8.820 1 @60
mvto 20.420 8.750 1 @61
mvto 18.221 8.681 1 @62
mvto 16.022 8.611 1 @63
mvto 13.823 8.542 1 @64
*DESTINE REACHED 2
mvto 11.624 8.472 1 @65
mvto 13.217 9.902 1 @66
mvto 14.888 11.333 1 @67
mvto 16.558 12.765 1 @68
mvto 18.229 14.197 1 @69
mvto 19.899 15.629 1 @70
mvto 21.569 17.060 1 @71
mvto 23.240 18.492 1 @72
mvto 24.910 19.924 1 @73
mvto 26.595 21.339 1 @74
mvto 28.120 19.754 1 @75
mvto 29.726 21.258 1 @76
mvto 31.910 20.994 1 @77
mvto 33.710 22.258 1 @78
mvto 35.863 22.713 1 @79
mvto 37.281 24.395 1 @80
mvto 38.729 26.051 1 @81
mvto 40.187 27.699 1 @82
mvto 40.189 27.702 1 @83
mvto 40.192 27.706 1 @84
mvto 40.194 27.710 1 @85
mvto 40.197 27.713 1 @86
mvto 40.199 27.717 1 @87
mvto 40.202 27.720 1 @88
mvto 40.204 27.724 1 @89
mvto 40.207 27.728 1 @90
mvto 40.210 27.731 1 @91
mvto 40.212 27.735 1 @92
mvto 40.215 27.738 1 @93
mvto 38.492 29.107 1 @94
mvto 36.883 30.608 1 @95
mvto 38.248 32.334 1 @96
mvto 39.956 33.719 1 @97
mvto 41.685 35.080 1 @98
mvto 43.391 36.470 1 @99
mvto 45.096 37.859 1 @100
mvto 46.802 39.249 1 @101
mvto 48.508 40.638 1 @102
mvto 50.213 42.028 1 @103
mvto 51.919 43.417 1 @104
mvto 53.625 44.807 1 @105
mvto 55.330 46.196 1 @106
mvto 57.036 47.586 1 @107
mvto 58.742 48.975 1 @108
*DESTINE REACHED 3
mvto 60.447 50.365 1 @109
mvto 57.800 50.000 1 @110
mvto 55.600 50.000 1 @111
mvto 53.400 50.000 1 @112
mvto 51.200 50.000 1 @113
mvto 49.000 50.000 1 @114
mvto 46.800 50.000 1 @115
mvto 44.600 50.000 1 @116
mvto 44.596 49.998 1 @117
mvto 44.593 49.995 1 @118
mvto 44.589 49.993 1 @119
mvto 44.585 49.990 1 @120
mvto 44.582 49.987 1 @121
mvto 44.579 49.984 1 @122
mvto 44.575 49.981 1 @123
mvto 44.572 49.978 1 @124
mvto 44.569 49.975 1 @125
mvto 44.566 49.972 1 @126
mvto 44.564 49.968 1 @127
mvto 44.879 47.791 1 @128
mvto 44.841 45.592 1 @129
mvto 44.847 43.392 1 @130
mvto 43.284 41.843 1 @131
mvto 41.120 42.240 1 @132
mvto 38.959 42.649 1 @133
mvto 36.795 43.050 1 @134
mvto 34.625 43.411 1 @135
mvto 32.569 44.192 1 @136
mvto 30.571 45.114 1 @137
mvto 28.574 46.037 1 @138
mvto 26.577 46.960 1 @139
mvto 24.580 47.883 1 @140
mvto 22.583 48.806 1 @141
*DESTINE REACHED 4

Script 2. Este script lee el estado de los sensores de contacto de un robot y con base en ello determina qué acción ejecutar. Por ejemplo, si los sensores frontales (sensor 1 y 2) no están haciendo contacto con ningún objeto, entonces el robot avanza hacia adelante; si el sensor frontal 1 o el sensor lateral derecho (sensor 3) están haciendo contacto, entonces el robot retrocede una pequeña distancia y gira hacia su izquierda; y si el sensor frontal 2 o el sensor lateral izquierdo (sensor 4) están haciendo contacto, el robot retrocederá cierta distancia y girará hacia la derecha. Este script continuará ejecutándose hasta que el usuario lo detenga, el script se muestra a continuación.

```
*CONTACT TEST
float r1, r2, r3, r4
Ini:
*READ CONTACT SENSORS
shs contact 1 @0
r1 <- RegF1 @1
shs contact 2 @2
r2 <- RegF1 @3
shs contact 3 @4
r3 <- RegF1 @5
shs contact 4 @6
r4 <- RegF1 @7

if r3 = 1.0 @8
goto LeftTurn @9
endif @10

if r4 = 1.0 @11
goto RightTurn @12
endif @13

if r1 = 1.0 @14
goto LeftTurn @15
endif @16

if r2 = 1.0 @17

goto RightTurn @18
endif @19

if r1 = 0.0 @20
if r2 = 0.0 @21
forward @22
endif @23
endif @24

goto Ini @25

LeftTurn:
backward @26
left @27
left @28
goto Ini @29

RightTurn:
backward @30
right @31
right @32
goto Ini @33
```

Capítulo VII
Conclusiones

El sistema Roc2008 fue desarrollado utilizando librerías multiplataforma, para el diseño de interfaces de usuario e implementación de aplicaciones que hacen uso de gráficos 3D, conservando las características principales de la versión anterior, como interacción con robots virtuales a través del lenguaje de programación Robel, e interacción del sistema a través de librerías desarrolladas por los usuarios mediante lenguaje C. Por lo que se ha cumplido el objetivo del proyecto, realizar una actualización del sistema conocido como Roc2 para que sea posible ejecutarse en distintas plataformas como Windows XP, Windows Vista y sistemas tipo Unix como GNU/Linux.

Al realizar la actualización del sistema se hicieron mejoras, las que se mencionan a continuación.

Mejora del sistema

Roc2008 es un sistema multiplataforma soportado en Windows 2000/XP/Vista así como en sistemas tipo Unix y GNU/Linux, para sistemas operativos tipo win32 se proporciona una archivo comprimido que contiene el ejecutable así como otros archivos necesarios para la ejecución del sistema. Para los sistemas operativos tipo GNU/Linux se proporciona un archivo comprimido que contiene el código fuente necesario para que el usuario compile el programa especialmente para su plataforma. Aunque no se realizaron pruebas sobre sistemas tipo OSX, también es posible compilar el código fuente para estos sistemas, ya que existe una versión de GTK+ y OpenGL compatibles con este tipo de sistemas.

Se realizaron mejoras en el sistema de colisiones, en la versión anterior Roc2 la detección de colisiones se basa en un sistema de cajas alineadas a los ejes de coordenadas o por sus siglas en ingles AABB (Axis Aligned Bounding Box) y como su nombre lo indica, cada cuerpo que se simula está representado físicamente por una caja, en la que al menos uno de sus lados está alineada a algún eje de coordenadas, por lo que la detección de colisiones se limita a verificar la colisión entre cajas. Para está nueva versión las colisiones se realizan tomando la malla que forma el objeto y se verifica las colisiones a través de está, si se trata de un objeto cóncavo se aproxima a un objeto convexo y se realizan las colisiones con éste nuevo objeto, lo cual permite realizar una simulación mas exacta.

Se incorpora la simulación de física en tiempo real lo que quiere decir que los cuerpos que se simulan en el entorno virtual tienen propiedades físicas que les permiten interactuar entre si.

Soporte de diferentes formatos de imágenes, en la versión anterior el sistema sólo soportaba los formatos tga e imágenes binarias, para está nueva versión es posible utilizar imágenes en formato bmp, tga, tiff, png, jpeg y gif.

Es posible simular un mayor numero de robots, en la versión anterior sólo era posible simular 5 robots al mismo tiempo, para está versión no existe un limite en el número de robots que se pueden simular, aunque hay que tomar en cuenta que a mayor número de objetos que se encuentren en la simulación el rendimiento del sistema será menor.

Mejoras en la interfaz de usuario, ya que además de contar con cuatro puertos de vista por los cuales se pueden seguir los movimientos del robot y una ventana de comandos, se añadió una venta con la cual se pueden agregar robots, esferas, cubos, cilindros cajas, y conos, así como modelos tridimensionales desde archivos 3DS, pistas con texturas y emisores infrarrojos, ya sea haciendo uso del mouse para colocar los objetos o indicando las coordenadas directamente mediante el uso del teclado.

El software se encuentra bajo licencia GPL (Licencia Pública General de GNU) lo que quiere decir que los usuarios tienen derecho a obtener, modificar, y redistribuir el código fuente del software.

Desventajas con respecto a la versión anterior

Aunque el sistema es portable, es necesario tener instalado el runtime environment de GTK+ (entorno de tiempo de ejecución de GTK) en su versión 2.8 o posterior, y si el usuario desea realizar alguna modificación tendrá que compilar el software con sus modificaciones y será necesario instalar el development environment de GTK+ (entorno de desarrollo de GTK+).

A diferencia de la versión anterior (Roc2), está nueva versión no cuenta con un sistema de procesamiento de voz por lo que al usar el comando “say” no genera sonidos únicamente se limita a mostrar el texto en la ventana de comandos que desea que el robot pronuncie.

Por el momento sólo es posible simular un tipo de robot RTX8, que es una representación en 3D del robot B14 fabricado por Real World Interface, que se encuentra en el Laboratorio de Biorobotica de la Facultad de Ingeniería de la UNAM.

El software fue desarrollado usando librerías multiplataforma que permiten que el código utilizado para compilar el software en las distintas plataformas sea el mismo, esto no es del todo cierto ya que para Windows Vista fue necesario realizar modificaciones en el código fuente, ya que las librerías utilizadas para crear la interfaz de usuario (GTK+ y GtkGLExt) no se comportan de igual forma en Windows Vista, pero aunque fue necesario realizar algunos cambios en el código fuente para esta plataforma, el sistema se sigue comportando de igual manera para Windows Vista.

El desempeño del simulador es menor por lo que la simulación es más lenta con respecto a la versión anterior.

No se cuenta con una aplicación para crear definiciones de robots en formato rbt como con la que cuenta la versión anterior (Roc2) con el nombre de Robot Builder.

Trabajo a futuro

Este sistema debe seguir desarrollándose para que el proyecto sigua creciendo y no caer en la obsolescencia, ya que actualmente el software que es desarrollado hoy, dentro de seis meses es necesario actualizarlo, para que no se vuelva obsoleto es necesario seguir mejorándolo y si es necesario eliminar o agregar mas funciones.

A continuación se mencionan algunas mejoras que se pueden hacer al sistema:

Mejorar el desempeño del sistema y del código, para hacerlo mas eficiente.

Agregar un mayor número de robots a simular.

Crear una aplicación similar a Robot Builder con la cual el usuario pueda crear sus propios robots para simularlos en el sistema.

Mejorar el formato de definición de mapas (entornos virtuales)

Agregar un mayor número de instrucciones al lenguaje Robel.

Agregar la posibilidad de controlar los robots virtuales a través de comandos vía voz.
Y mejorar la calidad de los gráficos de la simulación.

Apéndice A

Bibliotecas de enlace dinámico

Una biblioteca de enlace dinámico o DLL por sus siglas en ingles (Dynamic-Link Library) es un archivo en el que se encuentran funciones o subrutinas que pueden ser llamadas o ejecutadas desde otro programa o aplicación. Por lo que este archivo es previamente creado (compilando), ligando y almacenando todas las funciones que posteriormente cualquier proceso podrá utilizar. Como las funciones no son compiladas con el programa original se hace el enlace en tiempo de ejecución, lo que quiere decir, que las funciones que se requieran serán cargadas en memoria, de donde el programa podrá acceder a ellas.

Dentro de las librerías dinámicas existen funciones externas o exportables e internas, en donde las funciones internas solo pueden ser utilizadas dentro de la misma librería, esto quiere decir, que sólo las funciones definidas dentro ella pueden hacer uso de estás.

Mientras que las funciones externa o exportable las pueden utilizar tanto funciones definidas dentro de la librería como otras funciones fuera de está.

Dentro de los Sistemas W32 las librerías dinámicas comúnmente tienen la extensión .dll, mientras que en sistemas tipo GNU/Linux tienen la forma libnombre.so, a continuación se muestra un ejemplo de cómo crear una librería dinámica tanto en Windows como en GNU/ Linux.

A.1 Crear una librería dinámica para Windows

El siguiente ejemplo muestra como crear una librería dinámica haciendo uso de Visual Studio .NET 2003 (VS).

Pasos a seguir para crear la librería:

- Crear un nuevo proyecto tipo librería DLL.
- Agregar código fuente
- Crear del archivo de funciones de exportación
- Crear la librería.

A.1.1 Crear un nuevo proyecto tipo librería DLL

Dentro del VS hay que crear un proyecto de DLL seleccionando File/New/Project, nos mostrará una ventana con los diferentes tipos de proyectos que podremos crear. En el panel tipo de proyecto, en Visual C++ seleccionamos Win32 y en el panel de plantillas seleccionamos Aplicación de Consola Win32. Como se muestra en la figura.

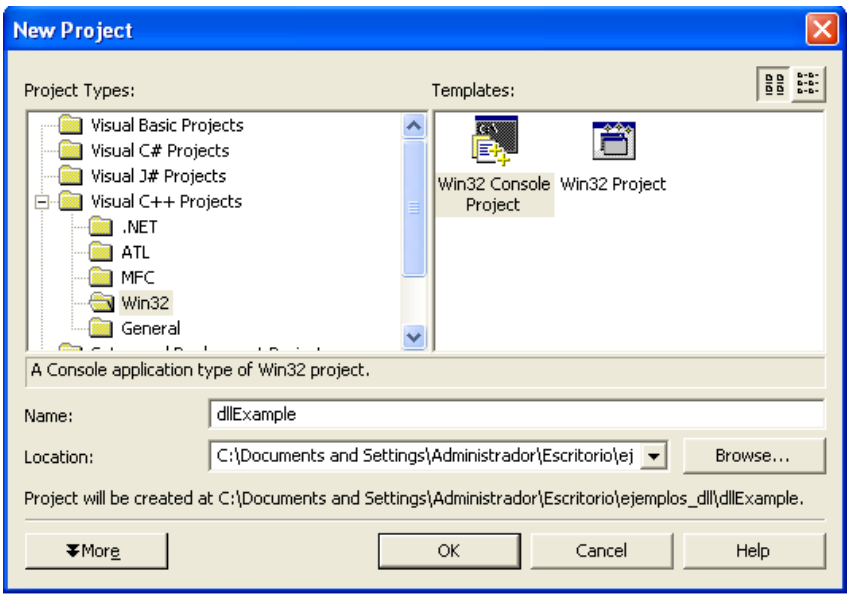


Figura A.1 Nuevo proyecto.

Se debe dar un nombre al proyecto, en este caso se llama dllExample, se selecciona OK, con lo cual nos mostrara la siguiente ventana.

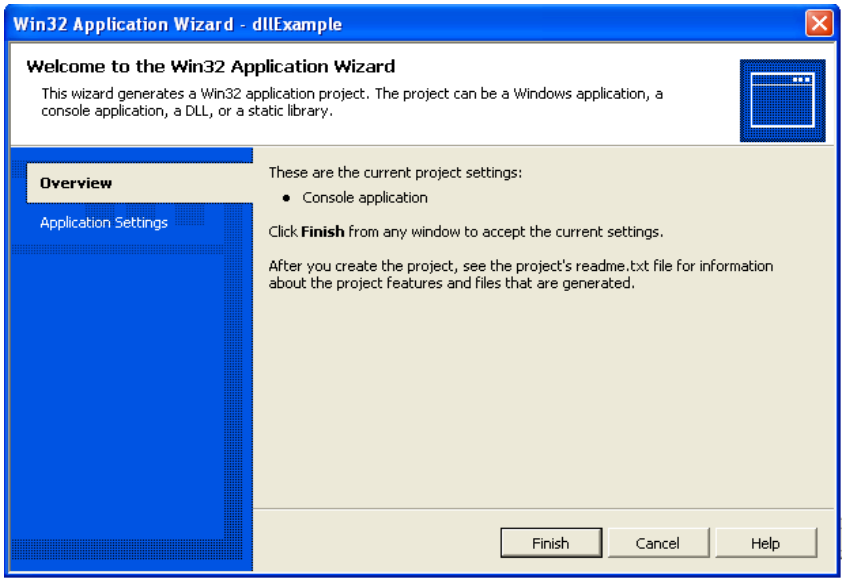


Figura A.2. Descripción del proyecto.

En esta ventana se selecciona la opción Application Settings, la que mostrará las opciones para el tipo de proyecto que se creará, así que se selecciona el tipo de aplicación DLL, como se muestra a continuación y finalmente se selecciona Finish.

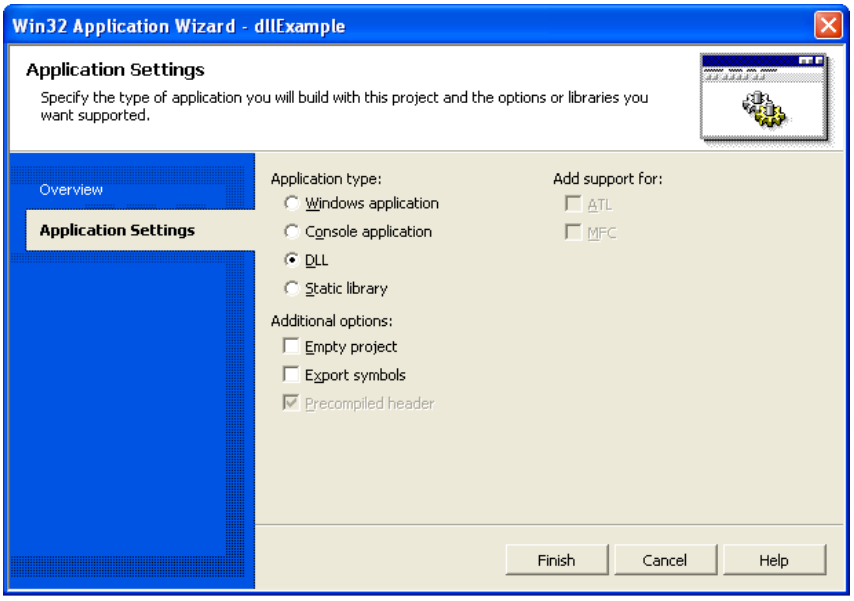


Figura A.3 Propiedades del proyecto.

Una vez hechos todos estos pasos se creará un proyecto de tipo DLL, a partir de una plantilla y se crearan los siguientes archivo.

stdafx.h
dllExample.cpp
stdafx.cpp

Donde el archivo dllExample.cpp es el de interés ya que es donde se coloca el código fuente de las funciones, también es necesario crear otro archivo con el nombre dllExample.h donde se colocan las definiciones de las funciones.

A.1.2 Agregar código fuente

Una vez que se crearon los archivos del proyecto es necesario agregar el código fuente encargado de realizar las operaciones.

A continuación se muestra el código correspondiente a cada archivo.

Código correspondiente al archivo dllExample.h

```
#ifndef    _DLL_EXAMPLE_H_
#define   _ DLL_EXAMPLE_H_

float evaluate_function (float x, float y);
float get_Square(float d);

#endif
```

Código correspondiente a dllExample.cpp

```
#include "stdafx.h"
#include <stdio.h>

//Esta función recibe un entero lo multiplica
//e imprime el resultado
float  evaluate_function(float x, float y){
        float aux;
        printf("El valor de x= %f",x);
        printf("El valor de y= %f",y);
        aux = get_Square(x*y)+2*y-x/3+2.0f;
printf("El valor de función es = %f",aux);
        return aux;
}

float get_Square(float d){
        return d*d;
}
```

A.1.3 Crear el archivo de funciones de exportación

Una vez creados los archivos donde se encuentre el código fuente de las funciones, es necesario agregar al proyecto un archivo make.def donde se indicara que funciones serán exportadas por el DLL como se muestra a continuación:

Código correspondiente al archivo make.def

```
LIBRARY dllExample
DESCRIPTION 'Ejemplo de archivo make.def'

EXPORTS

evaluate_Function     @       1
```

Las dos primeras líneas se usan para describir el archivo, la siguiente línea "EXPORTS" indica las funciones que se pueden exportar de la librería. La última línea indica la función que se puede exportar y esta dividida en tres campos, el primero es el nombre de la función "evaluate_Function", el siguiente campo "@" se usa como separador, por ultimo el tercero indica el identificador de la función en este caso 1.

A.1.4 Crear la librería

Por ultimo lo único que resta hacer para crear la librería es compilar el proyecto, si el proyecto no tiene ningún error de sintaxis o de referencias se creara una carpeta dentro del directorio donde se creo el proyecto con el nombre de Debug en el cual se creara la librería dinámica con el nombre dllExample.dll.

Si se presenta algún error, estos deberán corregirse para poder compilar el proyecto y generar la librería.

A.2 Crear una librería dinámica para GNU/Linux

Como se menciono anteriormente en los Sistemas GNU/Linux las librerías dinámicas tienen los siguientes nombre dllnombre.so, la extensión es de esta librería es diferente al que se maneja en sistemas tipo win32 debido a que el formato dll es propietario de Microsoft, por lo que en los sistemas GNU/Linux las librerías dinámicas tienen extensión so (shared object).

Para crear una librería dinámica en un sistema GNU/Linux en lenguaje c o c++ se utiliza el compilador gcc (GNU Compiler Collection).

gcc se utiliza cuando el código fuente de nuestro programa solo utiliza c.
g++ se utiliza cuando el código fuente de nuestro programa utiliza c++.

Para compilar un programa así como una librería dinámica usando el compilador gcc son necesarias cuatro etapas:

Preprocesado. En esta etapa se interpretan las directivas al preprocesador. Entre otras cosas, las variables inicializadas con #define son sustituidas en el código por su valor en todos los lugares donde aparece su nombre.

Compilación. La compilación transforma el código C en el lenguaje ensamblador propio del procesador de la máquina.

Ensamblado. El ensamblado transforma el programa escrito en lenguaje ensamblador a código objeto, un archivo binario en lenguaje de máquina ejecutable por el procesador.

Enlazado. Las funciones de C/C++ incluidas en el código, tal como printf() en el ejemplo, se encuentran ya compiladas y ensambladas en bibliotecas existentes en el sistema. Es preciso incorporar de algún modo el código binario de estas funciones al ejecutable. Se reúnen uno o más módulos en código objeto con el código existente en las bibliotecas.

Para poder poner el código en una librería se organizará de la siguiente manera.

Uno o mas ficheros fuente *.c o *.cpp con el código fuente de las funciones.

Uno o mas archivos de cabecera *.h con los tipos (typedefs, structs y enums) y prototipos de las funciones que se utilizan.

Este proceso es el que se sigue cuando se quiere obtener un programa ejecutable u obtener una librería dinámica, se sigue el mismo procedimiento con excepción de que en el último paso no se obtiene un programa ejecutable si no una librería dinámica

Tomando el código de los archivos utilizados para crear una librería dinámica bajo la plataforma win32.

```
#ifndef   _DLL_EXAMPLE_H_
#define   _DLL_EXAMPLE_H_

float evaluate_function (float x, float y);
float get_Square(float d);

#endif
```

El código fuente del archivo dllExample.c es.

```
#include "stdafx.h"
#include <stdio.h>

//Esta función recibe un entero lo multiplica
//e imprime el resultado
float  evaluate_function(float x, float y){
```

```
        float aux;
        printf("El valor de x= %f",x);
        printf("El valor de y= %f",y);
        aux = get_Square(x*y)+2*y-x/3+2.0f;
printf("El valor de función es = %f",aux);
        return aux;
}

float get_Square(float d){
        return d*d;
}
```

Es recomendable separar el código fuente en varias carpetas dentro de una sola, que contendrá el proyecto, por ejemplo en la carpeta principal habrá dos carpetas una con el nombre "src" donde se encontrarán todos los archivos con extensión *.c o *.cpp y otra carpeta con el nombre "include", donde se colocarán todas cabeceras que generemos durante el desarrollo de nuestro proyecto.

Una vez que se tienen los archivos necesarios *.h y *.c o *.cpp según sea el caso para crear una librería dinámica realizaremos los siguientes pasos.

Se crean todos los archivos objeto necesarios, en este caso sólo se debe crear un archivo, si la librería cuanta con mas archivos *.c se crearán sus correspondientes archivos objeto.

Para crear los ficheros objeto (*.o) de todas las fuentes (*.c), se utilizará el comando gcc de la siguiente manera:

```
gcc -c archivo_fuente.c -o archivo_objeto.o
```

Donde la opción "-c" le dice al compilador que no cree un ejecutable sino sólo un fichero objeto. El archivo "archivo_fuente.c" como su nombre lo indica es donde se encuentra el código fuente, la opción "-o" indica que el siguiente nombre es el nombre del archivo objeto de salida.

Finalmente para crear la librería se utilizará el comando ld de la siguiente manera:

```
ld -o libnombre_de_la_libreria.so archivo_objeto1.o archivo_objeto2.o ... -shared.
```

Donde la opción "-o" indica que la siguiente palabra será el nombre de la librería, los siguientes archivos "archivo_objeto1.o archivo_objeto2.o ..." son los archivos objetos necesarios para crear la librería, por último la opción "-shared" le indica al compilador que debe crear una librería dinámica.

A continuación se muestra la secuencia de comandos necesarios para crear la librería dinámica bajo sistema GNU/Linux.

```
gcc -c dllExample.c -o dllExample.o
ld -o libExample.so dllExample.o -shared
```

Con lo que se tendrá como resultado una librería dinámica con el nombre libExample.so.

Como se pudo observar en el ejemplo solo se manejan dos archivos, pero en proyectos serios, el numero de archivos que se manejan pueden llegar a ser demasiados, por lo que existe la herramienta make que simplifica todos estos pasos, haciendo uso de un archivo Makefile se colocan todas las reglas para generar los archivos objeto y generar el archivo ejecutable, o la librería dinámica según sea el caso.

A.3 Mini manual de Makefile

Los archivos Makefile están compuestos por medio de una serie de reglas las cuales tienen la siguiente estructura.

objetivo: dependencia1 dependencia2 ...
<tab>comando1
<tab>comando2
<tab>...

objetivo: es lo que se quiere construir, puede ser el nombre de un ejecutable o el nombre de una librería.

dependencia<i>: es el nombre de otro objetivo que debe crearse antes que el nuestro o bien ficheros de los que depende nuestro objetivo.

<tab>: es un tabulador es importante que coloquemos un tabulador, ya que de lo contrario el fichero no se leerá correctamente.

comando<i>: son las acciones que se tiene que realizar para construir el objetivo, estos comandos se ejecutan de forma secuencial y pueden ser cualquier comando de shell válido. (cc, rm, cp, ls, o incluso un script).

Considerando el siguiente ejemplo se tiene un programa compuesto por los siguientes archivos main.c, imprime.c, imprime.h, stack.c, stack.h, auxFuncion.c y auxFuncion.h

Por lo que el archivo Makefile sería el siguiente.

```
programa.exe : main.o  imprime.o stack.o auxFuncion.o
        gcc –o main.o  imprime.o stack.o auxFuncion.o

main.o : main.c auxFuncion.h
        gcc –c main.c auxFuncion.h

imprime.o: imprime.c auxFuncion.h snack.h
        gcc –c imprime.c auxFuncion.h snack.h
snack.o: snack.c auxFuncion.h
        gcc –c snack.c auxFuncion.h
```

```
auxFuncion.o: auxFuncion.c
        gcc –c auxFuncion.c
```

Existen macros automáticos que se pueden utilizar sin la necesidad de definir su valor y cuyo valor depende de la regla que estemos ejecutando.

$@	Objetivo de la regla que se esta ejecutando
$*	Objetivo con el sufijo eliminado
$<	Primer fichero de dependencia que permite que la regla se ejecute
$?	Lista de ficheros de dependencias mas recientes que el objetivo
$^	Lista de todas las dependencias

A.4 Uso de variables

Las variables dentro del archivo Makefile se definen de la siguiente manera:

nombre_variable = lista de palabras a las que sustituye.

Para hacer referencia a una variable dentro del archivo Makefile se hace de la siguiente forma:

$(nombre_variable)

Modificando el ejemplo anterior para usar variables se tiene lo siguiente:

```
CC = gcc
OBJETIVOS = main.o  imprime.o stack.o auxFuncion.o

programa.exe : $(OBJETIVOS)
        $(CC) –o $@ $?

main.o : main.c auxFuncion.h
        $(CC) –o $@ $?

imprime.o: imprime.c auxFuncion.h snack.h
        $(CC) –o $@ $?

snack.o: snack.c auxFuncion.h
        $(CC) –o $@ $?

auxFuncion.o: auxFuncion.c
        $(CC) –o $@ $?
```

Como se puede observar resulta fácil cambiar el compilador o agregar un nuevo nombre de archivo objeto, ya que se utilizan variables para referirnos a ellos.

Apéndice B

Manual de Usuario

B.1 Requerimientos mínimos

Procesador compatible con Intel a 400MHz o superior
64 MB de memoria RAM
20 MB de disco duro
2 MB de memoria de video
Resolución de 800x600 píxeles
Color de 16 bits

Para sistemas tipo win32 es necesario instalar el GTK+ 2 Runtime Environment 2.10.13 o superior. Para sistemas tipo Unix es necesario instalar las librerías de desarrollo de GTK+, ya que se les proporciona el código fuente que deberán de compilar para generar el binario para su plataforma.

B.2 Instalación

Instalación en Windows

Se proporcionará un archivo comprimido con el nombre Roc2008.zip en el que se encuentra todos lo archivos necesarios para el correcto funcionamiento del software.

No requiere instalación, únicamente hay que descomprimir el archivo, una vez descomprimido el archivo hay que localizar el archivo ejecutable Roc2008.exe, y hacer doble clic sobre este.

Recuerde que es necesario instalar previamente el GTK+ 2 Runtime Environment 2.10.13 o superior para el correcto funcionamiento del software, el cual se puede descargar de la siguiente pagina:

http://sourceforge.net/project/showfiles.php?group_id=71914

Instalación en GNU/Linux

Se proporcionará un archivo comprimido con el siguiente nombre Roc208_GNULinux.tar.gz.

Para descomprimirlo desde la línea de comandos debe teclear el siguiente comando.

$ tar –xvzf Roc2008_GNULinux.tar.gz

Para compilar el proyecto y generar el binario se utiliza la utilería make, de la siguiente forma:

$ cd Roc2008
$ make

Si no obtiene ningún error al momento de la compilación se habrá generado el archivo ejecutable dentro de la carpeta Roc2008.

Finalmente para ejecutar Roc2008 desde la línea de comandos, se debe de posicionar dentro de la carpeta que contiene el archivo binario y ejecutarlo, como se muestra a continuación:

$ cd Roc2008
$ ./Roc2008

NOTA:

Recuerde que para compilar el proyecto es necesario tener instaladas las siguientes librerías de desarrollo GTK+, GtkGLExt, SolarSockets y Newton Game Dynamic.

GTK+:

Las librerías de desarrollo de GTK+ se pueden descargar de la siguiente página:

www.gtk.org.

En donde también se proporciona los documentos necesarios para realizar la instalación de GTK+ para la versión de su sistema operativo.

GtkGLExt:

Las librerías de desarrollo de GtkGLExt se pueden descargar de la pagina:

http://www.k-3d.org/gtkglext/Main_Page

En donde también se proporcionan los documentos necesarios para realizar la instalación de GtkGLExt para la versión de su sistema operativo.

SolarSockets:

Las librerías de desarrollo de SolarSockets se pueden descargar de la página:

http://www.solarsockets.solar-opensource.com/index.php/Portada

En donde también se proporciona la documentación necesaria para realizar la instalación de SolarSockets para la versión de su sistema operativo.

Newton Game Dynamic:

El archivo que descarga contiene una versión de Newton Game Dynamic, por lo que no es necesario descargarlo, pero si no es compatible para su sistema operativo deberá descargar la librería de la página:

http://www.newtondynamics.com/.

En donde también se proporcionan los documentos necesarios para realizar la instalación de Newton Game Dynamic para la versión de su sistema operativo.

El sistema Roc2008 puede ser descargado de la página de Biorobotica de la UNAM:

http://biorobotics.fi-p.unam.mx/

B.3 Entorno del sistema

El entorno del sistema Roc2008 se muestra en la figura B.1

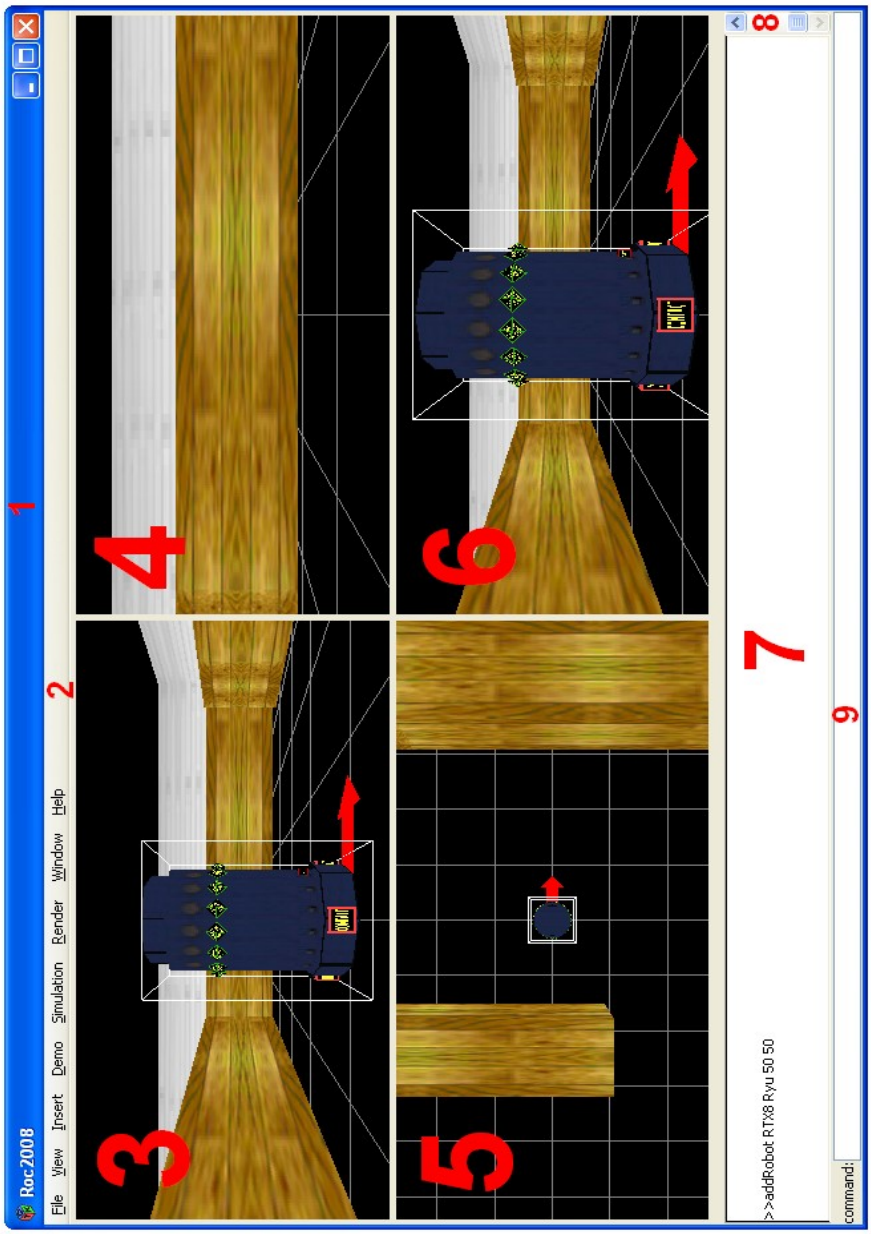
El entorno del sistema se encuentra divido en tres secciones principales, barra de titulo y barra de menú, área de visualización de resultados y sección de comandos.

La barra de titulo y barra de menú contienen el nombre del sistema y el menú principal respectivamente.

Área de visualización de resultados esta compuesta por cuatro ventanas o puertos de vista en los cuales el usuario puede observar cuatro distintos puntos de vista del robot, con los que se puede seguir la simulación de los robots.

La sección de comandos esta dividida en dos, una sección en la cual se pueden observar los comandos tecleados así como mensajes o resultados que pudieran arrojar dichos comandos, y una línea de comandos en la cual el usuario podrá teclear los comandos.

A continuación se muestra una imagen de la interfaz principal del sistema:



B.1 Entorno de trabajo del Roc2008

1. Barra de titulo, contiene el nombre de la aplicación así como los botones para minimizar, maximizar y cerrar la ventana.

2. Barra de menú, se encuentran los menús desplegables con los que se puede acceder a funciones del sistema como agregar objetos, ejecutar los demos de ejemplo, etc.

3. Puerto de vista de la cámara libre del sistema, mediante esta ventana el usuario puede desplazarse por el entorno virtual con ayuda del mouse.

4. Puerto de vista de la vista del robot, en este ventana se muestra lo que el robot esta observando en el entorno virtual.

5. Puerto de vista de la vista superior, en esta ventana se muestra la vista de una cámara posicionada a una cierta distancia sobre el robot.

6. Puerto de vista de la vista lateral, en esta ventana se muestra la vista de una cámara posicionada al costado derecho del robot y a una cierta distancia.

7. Área de visualización de texto, se muestran los comandos que se teclean en la línea de comandos así como los resultados de estos.

8. Scroll vertical, se utiliza para desplazar el texto hacia arriba y abajo del área de visualización de texto.

9. Línea de comandos, es donde el usuario escribe los comandos para interactuar con el sistema o el robot.

B.4 Interacción con el sistema

La interacción del usuario con el robot se puede realizar de forma local o remota, como se describe a continuación:

El usuario puede interactuar con el sistema de forma local a través del teclado, por medio del uso de comandos de Robel, que deben ser introducidos en la línea de comandos.

La interacción del sistema a través del uso del mouse, consiste en mover el mouse sobre un puerto de vista, mientras se mantiene presionado alguno de los botones con lo que el usuario puede cambiar la posición y orientación de la cámara libre.

La siguiente figura muestra la orientación de los ejes coordenados utilizados en el sistema.

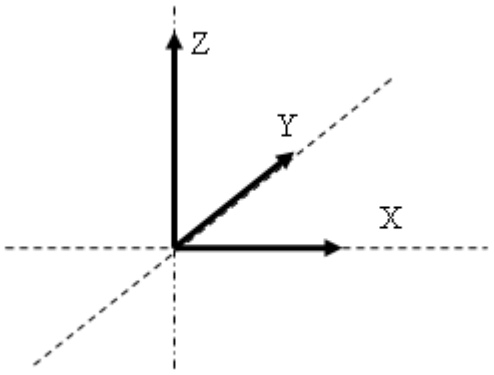


Figura B.2 Orientación de los ejes coordenados.

A continuación se muestra una tabla de los posibles movimientos del mouse.

Evento del mouse	Efecto sobre el viewport
Ningún botón es presionado y el cursor está centrado en la ventana.	Ninguno
El botón izquierdo es presionado y el cursor se localiza arriba del centro de la ventana.	El punto de vista se mueve hacia adelante.
El botón izquierdo es presionado y el cursor se localiza abajo del centro de la ventana.	El punto de vista se mueve hacia atrás.
El botón izquierdo es presionado y el cursor se localiza a la izquierda del centro de la ventana.	El punto de vista rota hacia la izquierda sobre el eje Z.
El botón izquierdo es presionado y el cursor se localiza a la derecha del centro de la ventana.	El punto de vista rota hacia la derecha sobre el eje Z.
El botón derecho es presionado y el cursor se localiza arriba del centro de la ventana.	El punto de vista se mueve hacia arriba.
El botón derecho es presionado y el cursor se localiza abajo del centro de la ventana.	El punto de vista se mueve hacia abajo.
El botón derecho es presionado y el cursor se localiza a la izquierda del centro de la ventana.	El punto de vista se desplaza hacia la izquierda.
El botón derecho es presionado y el cursor se localiza a la derecha del centro de la ventana.	El punto de vista se desplaza hacia la derecha.
Los botones izquierdo y derecho son presionados y el cursor se localiza arriba del	El punto de vista rota hacia arriba sobre el eje X.
Los botones izquierdo y derecho son	El punto de vista rota hacia

presionados y el cursor se localiza abajo del centro de la ventana.	abajo sobre el eje X.
Los botones izquierdo y derecho son presionados y el cursor se localiza a la izquierda del centro de la ventana.	El punto de vista rota hacia la izquierda sobre el eje Y.
Los botones izquierdo y derecho son presionados y el cursor se localiza a la derecha del centro de la ventana.	El punto de vista rota hacia la derecha sobre el eje Y.

Tabla B.1 Efecto de los eventos del mouse sobre la cámara libre.

A continuación se muestra una lista de los efectos del teclado sobre cualquier puerto de vista del sistema.

Tecla	Efecto sobre el robot seleccionado
Flecha arriba	Avanza una unidad
Flecha abajo	Retrocede una unidad
Flecha izquierda	Rota a la izquierda
Flecha derecha	Rota a la derecha
Barra espaciadora	Sujeta un objeto que se encuentre dentro del área de selección, si el robot ya se encuentra sujetando algún objeto lo suelta.

Tabla B.2 Efecto del teclado sobre el sistema.

Interacción de forma remota. Es posible interactuar con el sistema de forma remota pero limitada al uso de instrucciones de Robel, la comunicación se realiza mediante sockets que estan a la espera de recibir un comando que se mandara al respectivo robot, el cual los interpretara.

Menús del sistema

La barra de menús permite interactuar con el sistema, mediante menús desplegables.

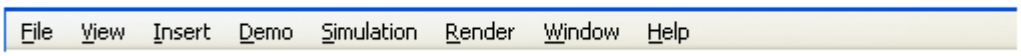


Figura B.3 Barra de menús del Roc2008.

A continuación se muestran y describen los diferentes menús con los que se pueden interactuar.

Menú File.

Mediante el menú file el usuario puede cargar en el mundo virtual dos diferentes tipos de archivos. Con la opción "Open World" se puede cargar mapas definidos por el usuario, cuando se selecciona esta opción se muestra una ventana de selección en la cual se le pide al usuario que seleccione la ubicación del archivo del mapa que desea cargar en el mundo (para obtener información de cómo definir los mapas ir al apartado obstáculos tipo mapa).

La opción "Conf File" permite cargar en el mundo virtual escenas previamente definidas mediante un archivo de configuración que se puede crear de forma gráfica como se muestra en el menú "Insert"

Con la opción "Quit" se cierra la ventana y se finaliza la ejecución del sistema.

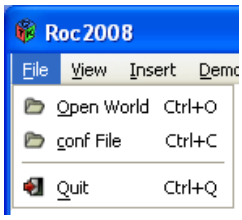


Figura B.4 Menú File.

Menú View

En el menú view muestra que puertos de vista se encuentran habilitados (cuales se encuentran visibles) en este momento, para ocultar alguno de los cuatro puertos de vista hay que deshabilitarlo en el menú view.

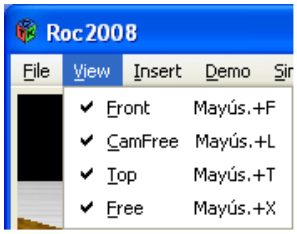


Figura B.5 Menú View.

Menú Insert

Mediante el menú Insert el usuario puede agregar al mundo virtual: objetos del tipo esferas, cilindros, cubos, conos, cajas o emisores de luz infrarroja, y geometrías complejas como modelos en formato 3DS o robots.

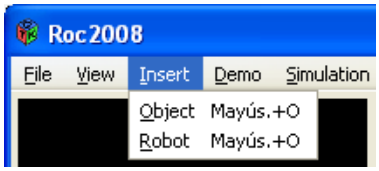


Figura B.6 Menú Insert.

Al seleccionar una de las dos opciones Object o Robot, se muestra una ventana de inserción de objetos como se muestra en la figura B.7.

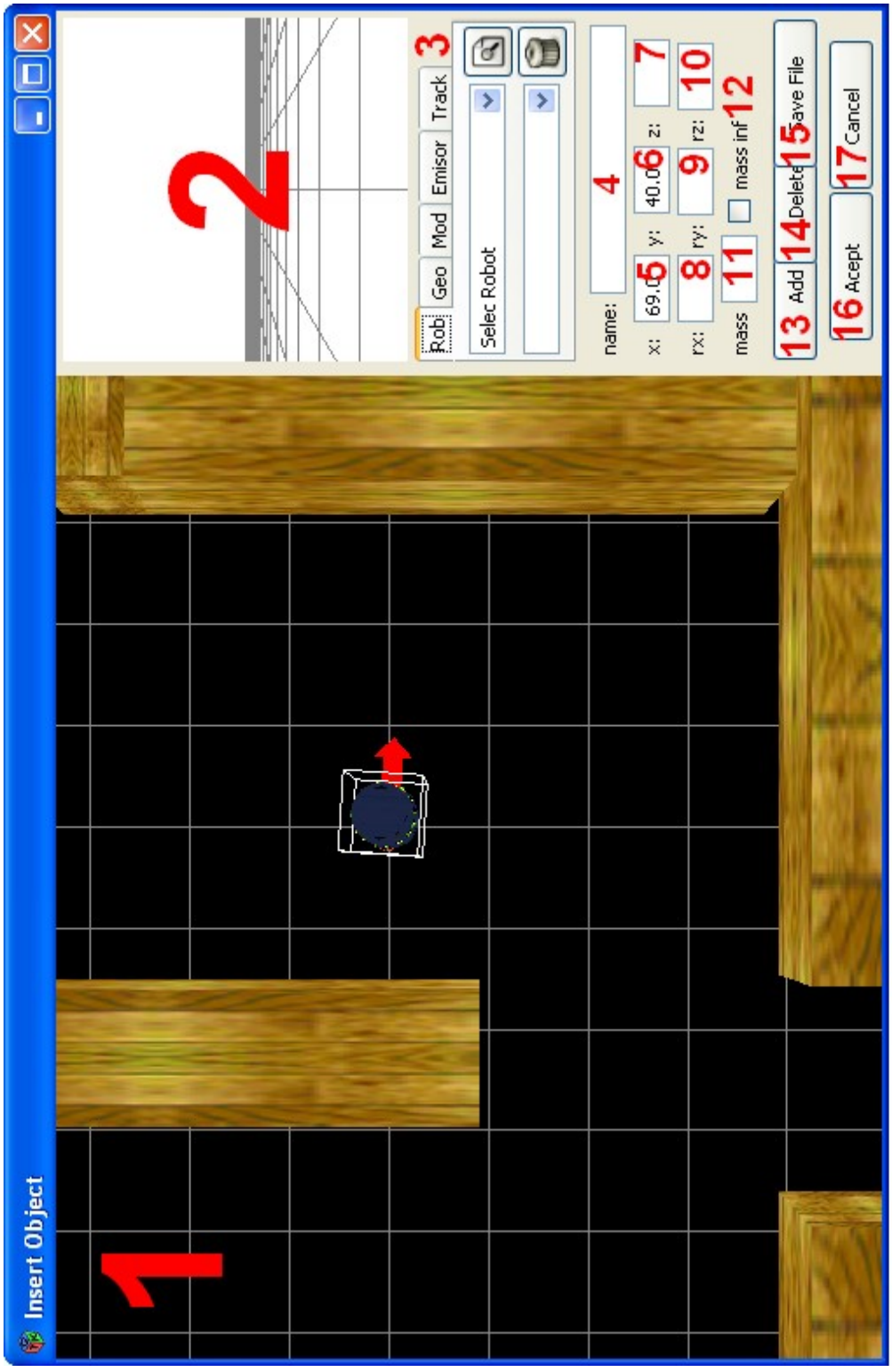


Figura B.7 Ventana de inserción de objetos.

A continuación se describe cada elemento que conforman a esta ventana.

1. Puerto de vista del mundo, se muestra la vista de una cámara localizada a una distancia sobre el mundo virtual, se encuentra orientada en dirección de –z, el usuario puede cambiar la posición haciendo uso del mouse, mas adelante se describe como realizar estos cambios.

2. Puerto de vista de los modelos, se muestra una vista previa de los objetos que se desea cargar en el mundo, una vez que se ha seleccionado de alguna de las pestañas.

3. Pestañas de selección, se pueden seleccionar los diferentes objetos, modelos o robots que se desean agregar al mundo virtual, la descripción de estas pestañas se realizará con mayor detalle en secciones posteriores.

4. Nombre del Objeto, se muestra el nombre que tendrá ya sea la geometría, el modelo tridimensional o el robot dentro del mundo virtual, y que puede ser cambiado por el usuario.

Los objetos 5,6 y 7 muestran respectivamente las coordenadas (x,y,z) del objeto dentro del mundo virtual, y pueden ser modificados por el usuario.

Los objetos 8, 9 y 10 muestran los ángulos de rotación del objeto con respecto a los ejes coordenados x, y, z respectivamente y pueden ser modificados por el usuario.

11. Masa del objeto, se muestra la masa que el objeto tendrá dentro del mundo virtual y puede ser modificada por el usuario mientras no este seleccionada la casilla de masa infinita.

12. Selección de masa, esta opción es utilizada para indicar si el objeto será un objeto estático o dinámico, cuando se encuentra seleccionada esta casilla indica que el objeto tendrá una masa infinita por lo que el objeto permanece estático, si esta casilla se encuentra deshabilitada indicará que el objeto será dinámico y la masa que tendrá este objeto se le pude asignar mediante la casilla de masa, como se muestra el punto anterior

13. Botón Add, es una de las formas de agregar objetos dentro del mundo, una vez que se tiene todos los datos necesarios para agregar el objeto al mundo, como posición, orientación y masa del objeto, se puede hacer clic sobre la posición donde se quiere colocar el objeto, o se puede agregar mediante las coordenadas de forma explicita y posteriormente hacer clic sobre este botón para colocar el objeto en esa posición.

14. Botón delete, si por algún motivo no se desea hacer uso del objeto seleccionado se puede eliminar de la escena al hacer clic sobre este botón, previamente el objeto deberá de estar seleccionado para poder ser eliminado

15. Botón save File, ya que se han seleccionaron los objetos que se desean agregar se permite guardar la configuración de la escena descrita, al hacer clic sobre este botón se despliega una ventana en la cual se pide el nombre del archivo en el cual se guardará la escena descrita, así como la ubicación de este, posteriormente este archivo puede ser agregado al mundo virtual mediante el menú “File”.

16. Botón Acept, ya que se tienen todos los objetos que se desean agregar basta con hacer clic sobre este botón para agregarlos al mundo virtual.

17. Botón Cancel. Si por alguna razón no se desean realizar los cambios antes hechos en la escena basta con hacer clic en este botón para borrar los objetos antes creados y cerrar la ventana de inserción de objetos y regresar a la ventana principal.

Pestañas de selección

Como se mencionó en el punto (3) existen cinco diferentes pestañas en las cuales se pueden seleccionar diferentes objetos y robots para agregar al mundo virtual, las cuales son:

Pestaña Rob: se utiliza para seleccionar el tipo de robot que se desea agregar a la escena.

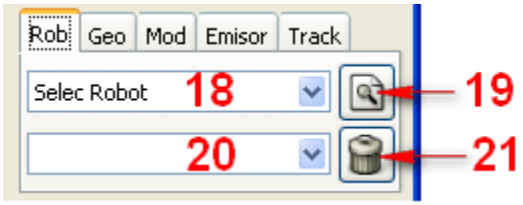


Figura B.8 Pestaña Rob.

16. comboBox de selección de Robot, permite seleccionar el tipo de robot que se desea agregar a la escena, por el momento solo se pueden cargar robots del tipo RTX8.

Figura B.9 Robot RTX8.

17. Botón de selección de Script, con este botón se pude seleccionar una serie de scripts que se cargaran en el robot para su posterior uso.

18. comoboBox de selección de Script, en este comboBox se muestran todos los Scripts que se cargaran en el robot.

19. Botón de eliminación de scripts, se pueden eliminar los scripts del comboBox de selección de Scripts, para eliminar un script solo hay que seleccionar uno del comboBox de Scripts y se habilitara este botón, así que para eliminar ese script solo hay que hacer clic sobre él.

Pestaña Geo: se utiliza para seleccionar el tipo de geometría básica que se desea agregar a la escena, se puede seleccionar geometrías básicas, esfera, cubo, cilindro, cono y rectángulo.

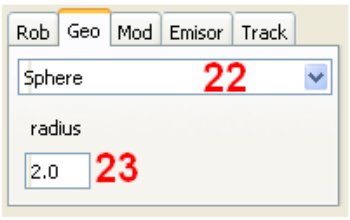


Figura B10 Selección esfera

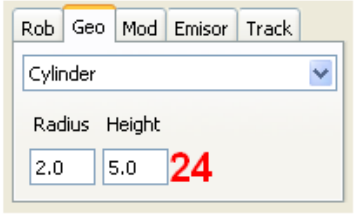


Figura B.11 Selección cilindro

Figura B.12 Selección rectángulo

Dependiendo de la geometría que se seleccione aparece cualquiera de los objetos 23, 24 o 25 en el que se podrán modificar las respectivas dimensiones de la geometría seleccionada.

Pestaña Mod: se utiliza para seleccionar el tipo de modelos tridimensional en formato 3DS que se desea agregar a la escena.

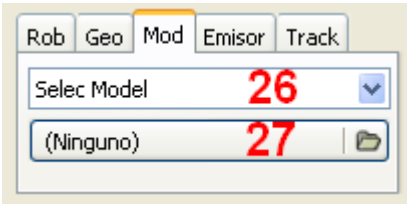


Figura B.13 Pestaña Mod.

26. comboBox de selección de modelo, se muestra una lista de modelos que se encuentran cargados en el sistema y que podrán ser agregados al mundo virtual.

27. Búsqueda de modelo, si dentro de la lista de modelos de selección no se encuentra el modelo que se desea agregar se podrá hacer uso de este botón, con el cual se podrá buscar el archivo que contiene al modelo en formato 3DS y así agregarlo a la lista de modelos de selección.

Las restricciones que se tienen para la carga de modelos 3DS son las siguientes.

- Los archivos en formato 3DS no deben tener más de un modelo en un archivo.
- El modelo debe de estar formado con una sola malla.
- No debe tener un sistema de luces.
- No debe tener un sistema de cámaras.

Pestaña Emisor: Se utiliza para seleccionar un emisor de luz infrarroja que se desea agregar a la escena.

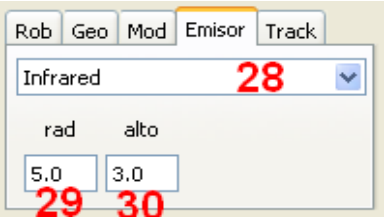


Figura B.14 Pestaña Emisor.

28. Selección del emisor, se utiliza para seleccionar el tipo de emisor que se desea agregar al mundo virtual, por el momento solo se cuenta con emisores de luz infrarroja.

Las casillas 29 y 30 modifican los parámetros por default, radio y altura del cono de emisión que se utiliza para representar el haz de luz de emisión de luz infrarroja, estos parámetros pueden ser modificados por el usuario.

Pestaña Track: se utiliza para colocar un plano rectangular en la posición (x,y,z) de dimensiones n, m con una cierta textura, el cual representara una pista en el mundo virtual.

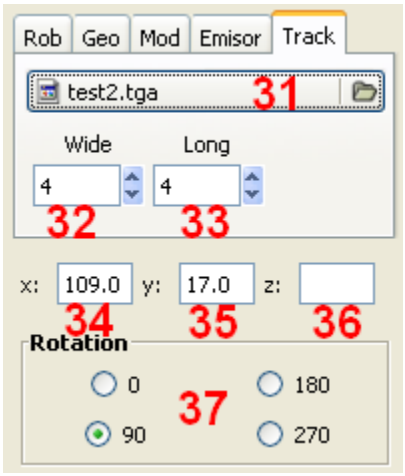


Figura B.15 Pestaña Track.

31. Botón de selección de textura, se selecciona la textura que se desea agregar a la pista, los formatos de imágenes que soporta son jpeg, png, gift, tiff y bmp, las dimensiones de la imagen deben de ser de $2^n x 2^m$, donde n y m son dos números pares positivos.

Las casillas 32 y 33 representan el ancho y largo de la pista respectivamente y pueden ser modificados por el usuario, se recomienda que sean números pares.

Las casillas 34, 35 y 36 representan la posición del centro del plano en el mundo virtual y pueden ser modificados por el usuario.

37. Ángulo de rotación de la textura, representa el ángulo que se rotara la imagen antes de ser colocada en la pista los ángulos pueden ser 0, 90, 180 y 270 grados.

B.5 Interacción con la ventana de inserción de objetos

El usuario puede moverse en el mundo y cambiar la posición del objeto seleccionado haciendo uso del mouse, también puede indicarse de forma explicita la posición del objeto a través de las casillas 5,6 y 7 que son la posición del objeto en x,y,z, respectivamente.

La orientación de los ejes para el puerto de vista del mundo es el siguiente.

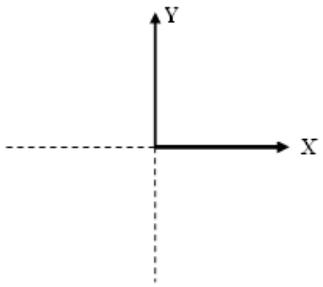


Figura B.16 Orientación de los ejes para puerto de vista del mundo.

El usuario puede mover la posición del la cámara al mover el cursor dentro del puerto de vista del mundo, como se muestra a continuación:

Movimientos del mouse	Efecto sobre la posición de la cámara
El cursor se mueve hacia arriba de la ventana.	El punto de vista se mueve hacia y
El cursor se mueve hacia abajo de la ventana.	El punto de vista se mueve hacia -y
El cursos se mueve hacia la izquierdo de la ventana	El punto de vista se mueve hacia -x
El cursos se mueve hacia la derecha de la ventana	El punto de vista se mueve hacia x

Tabla B.3 Efectos del movimiento del mouse sobre la posición de la cámara del puerto de vista del mundo.

El usuario puede colocar objetos en el mundo por medio de mouse, moviendo el mouse a la posición donde desea colocar el objeto y haciendo clic con el botón derecho del mouse para colocar el objeto. También puede indicar de manera explicita las coordenadas (x,y,z) del objeto donde quiere colocarlo a través de las casillas 5,6 y 7 de la ventana insertar objeto y haciendo clic en botón Add (13) para colocarlo en la posición indicada por las casillas 5,6 y 7.

Menú Demo

Mediante este menú se hace uso de los diferentes demo con los que cuenta el sistema, con los cuales se muestra el funcionamiento de Roc2008.

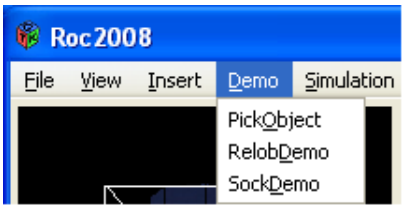


Figura B.17 Menú Demo.

Menú Simulation

El menú simulación cuenta con dos opciones, "Clear World" elimina todos los objetos que se encuentren dentro del mundo virtual y "Stop" que detiene la simulación que se este realizando en ese momento, ya sea que se este ejecutando un script o una función del usuario.

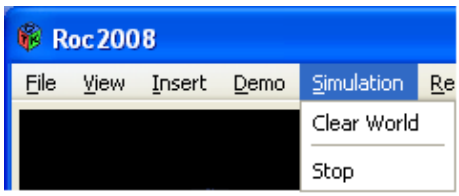


Figura B.18 Menú Simulation.

Menú Render

Mediante el menú Render se puede cambiar la forma de visualizar la escena gráfica, ya sea en forma de malla de alambre o de modelos texturizados.

Figura B.19 Menú Render.

A continuación se muestra un ejemplo.

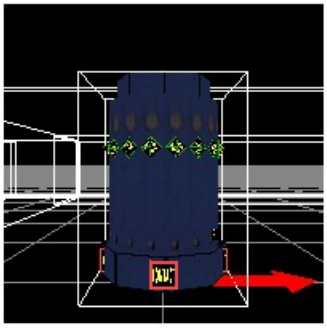


Figura B.20 Visualización en malla de alambre.

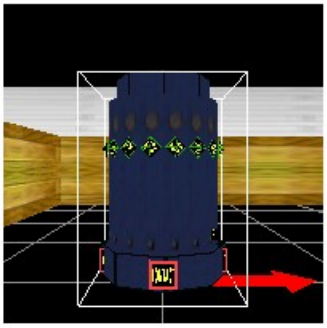


Figura B.21 Visualización en modelos texturizados.

Menú Windows

Este menú sólo cuenta con una opción “Properties” la cual muestra una segunda ventana donde se puede configurar el comportamiento de Roc2008, como se muestra a continuación.

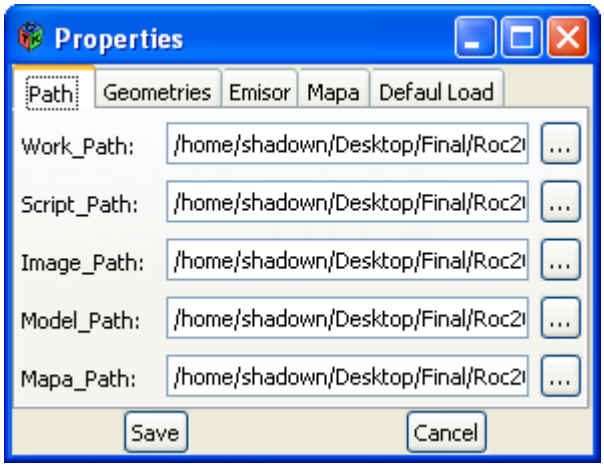


Figura B.22 Ventana de Propiedades de Roc2008.

La ventana de propiedades consta de tres secciones, la barra de titulo que contiene el nombre de la ventana y los botones para minimizar, maximizar y cerrar la ventana. La sección de pestañas en la cual se encuentran cinco pestañas de configuración del sistema, y finalmente la secciones de botones, en la cual se encuentra el botón “Save” utilizado para guardar todos los cambios que se han realizado, y el botón “Cancel” usado para cerrar la ventana y cancelar los cambios realizados en la configuración del sistema.

A continuación se describe el funcionamiento de cada pestaña.

Pestaña Path

Cuenta con cinco campos de configuración:

Work_Path: Es la carpeta de trabajo del sistema, por default se toma la carpeta donde se encuentra el archivo ejecutable del Roc2008.

Dentro de esta carpeta debe existir una carpeta con el nombre de userLib, donde se colocan las librerías dinámicas UserConectRoc y Roc2User, en donde Roc2User es la librería en la que se encuentran las funciones definidas por el usuario y la librería UserConectRoc es donde se encuentran la funciones encargadas de realizar la comunicación entre las funciones del usuario y el sistema.

Script_Path: Es la carpeta donde se encuentran los scripts que se cargaran al robot, y se deben colocar todos los scripts que el usuario desea cargar al robot.

Image_Path: En esta carpeta se guardan las imágenes capturadas por el robot, así como las imágenes que se utilizaran en las pistas.

Model_Path: En esta carpeta se colocan los archivos que contienen los modelos 3D en formato 3DS que se desean cargar en el sistema

Mapa_Path: En esta carpeta se guardan los archivos de definición de mapas.

Estas carpetas se pueden cambiar ya sea escribiendo la ruta completa de la carpeta o haciendo uso del botón que se encuentra a su derecha, el cual muestra una venta con la que se puede seleccionar la ruta de la carpeta que se desea usar.

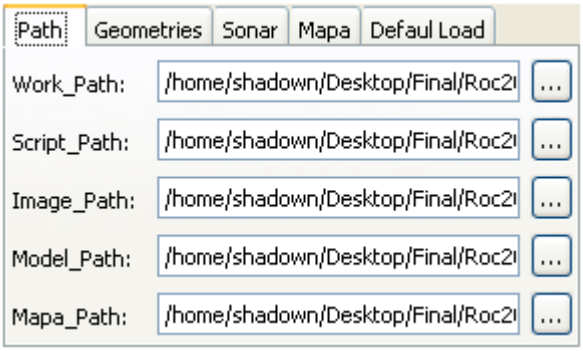


Figura B.23 Pestaña Path.

Pestaña Geometries

Esta pestaña esta dividida en cinco frames (secciones) con el correspondiente nombre de la geometría a la que corresponde, en las casillas se configuran los parámetros por default que se toman para crear respectivamente cada geometría, estos parámetros pueden ser cambiados, simplemente modificando los valores de las casillas.

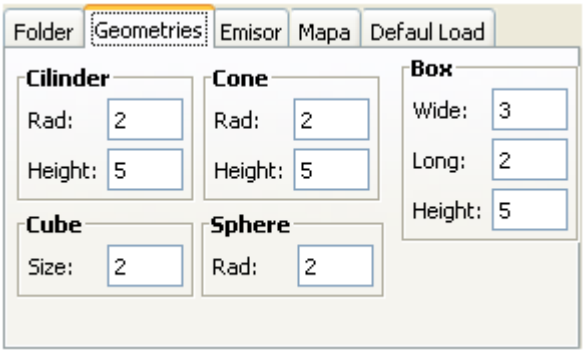


Figura B.24 Pestaña Geometries.

Pestaña Emisor

En esta pestaña se muestran los valores por default que toma el cono que representa el haz de emisión de luz infrarroja.

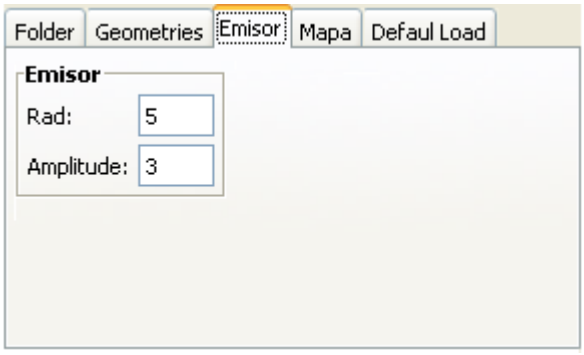


Figura B.25 Pestaña Sonar.

Pestaña Mapa

En esta sección se muestran los valores por default de configuración para los mapas, donde el frame con el nombre Wall contiene la altura que tomaran las paredes de los mapas, el frame con el nombre Object contiene la altura que tomaran los objetos que se definan dentro de un mapa, y por ultimo el frame con el nombre Track contiene las dimensiones que tomaran las pistas que se creen en el sistema.

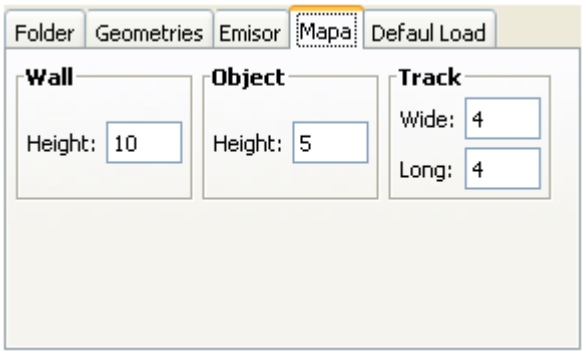


Figura B.26 Pestaña Mapa.

Pestaña Default Load

En esta pestaña se muestran dos campos un campo de mapa, que muestra la ubicación del mapa que se cargará al sistema cuando este inicie y a su derecha se muestra un botón para seleccionar la ubicación de este mapa. El siguiente campo es el comando que se ejecutará al inicio del sistema en este caso al inicio se ejecutará el comando addRobot RTX8 Ryu 50 50 el cual carga un robot del tipo RTX8 con el nombre Ryu en la posición (50,50,0).

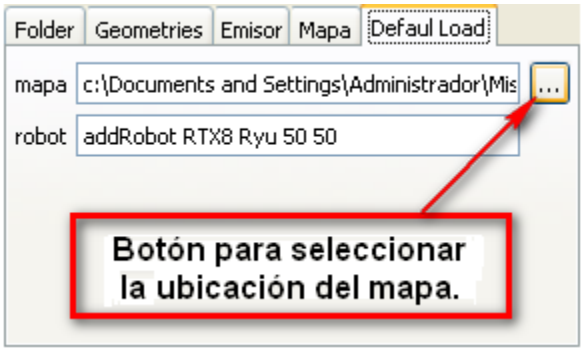


Figura B.27 Pestaña Default Load.

B.6 Interacción con el sistema a través de las librerías de usuario

Como se menciono anteriormente otra forma de interactuar con el sistema es a través de las funciones de usuario definidas dentro de bibliotecas dinámicas.

Estas funciones de usuarios como su nombre lo dice están definidas por el usuario y se encuentran empaquetadas en la librería Roc2User, y a través de esta librería se cargan las funciones al sistema.

Para que las funciones de usuario se puedan comunicar con el sistema, es necesario hacer uso de la librería UserConectRoc, la cual cuenta con todas las funciones necesarias para realizar la comunicación entre las funciones definidas en la librería Roc2User y Roc2008.

Figura B.28 Diagrama de comunicación entre las funciones de usuario y roc2008.

Para realizar la comunicación entre las funciones de usuario con el sistema Roc2008 se proporciona una librería de comunicación "UserConectRoc" contiene los elementos necesarios para comunicar las funciones del usuario con el sistema, las funciones de usuario se encuentran en la librería Roc2User. Estas librerías se encuentran en carpetas separadas dentro "userLib" con el nombre de cada librería que contienen.

Dentro de la carpeta de trabajo Roc2008 existe una carpeta con el nombre userLib en donde se proporcionan dos carpetas con los nombres UserConectRoc y Roc2User donde se encuentran los proyectos para generar las respectivas librerías. Para el caso la plataforma win32 dentro de cada carpeta se proporciona un proyecto en visual Studio 2003 .Net con todos los archivos necesario para generar las respectivas librerías, mientras que para

plataformas tipo Unix dentro de cada carpeta se encuentra un archivo Makefile con todas las instrucciones necesarias para crear las librerías a través de la utilería make.

B.7 Robel

Robel es un lenguaje de programación orientado al control de un robot móvil. Desde un principio se planeó como una herramienta para llevar a cabo el control de forma rápida y sencilla haciendo uso de instrucciones. Sus características principales son:

• Instrucciones para el control de movimiento
• Instrucciones para la lectura del estado lógico del robot
• Instrucciones para comunicación vía Internet
• Control de flujo del programa haciendo uso de ciclos while, if y saltos a etiquetas
• Soporta dos tipos de variables: enteros y flotantes
• Independencia de la plataforma
• Posibilidad de tener recursividad
• Llamadas a funciones creadas por el usuario (otros scripts)

Para que un programa escrito en Robel se pueda ejecutar sobre distintas plataformas, es necesario hacer uso de un programa que traduzca cada instrucción de Robel al lenguaje propio de cada sistema; es decir, se debe contar con la Máquina Virtual de Robel (MVR).

El acumulador forma parte de la MVR, su propósito es guardar datos para el manejo interno de la ejecución de instrucciones, así como almacenar el resultado de la ejecución de algunas instrucciones. Es decir, el acumulador es una estructura de datos utilizada para compartir información por medio de cada una de las etapas de ejecución de las instrucciones. La estructura del acumulador se muestra en el siguiente cuadro.

Registro	Tipo	Nombre	Mnemónico en Robel
F1	float	Registro flotante 1	RegF1
F2	float	Registro flotante 2	RegF2
F3	float	Registro flotante 3	RegF3
I1	Integer	Registro entero 1	RegI1
I2	Integer	Registro entero 2	RegI2
S	String	Registro string	RegS
Snack	Pila de flotantes	pila	RegSp

Tabla B.4 Registros de la maquina virtual

Sólo puede hacerse uso de los registros de la máquina virtual dentro un script y no es posible modificar su contenido. Para hacer referencia a un registro sólo hay que usar el mnemónico asociado.

Las instrucciones de Robel pueden ser ejecutadas por el sistema mediante dos formas distintas: un script o introduciéndolas en línea comandos. Si desea utilizar un script, para

ambos casos las instrucciones deben tener el formato indicado en el cuadro. Los parámetros entre corchetes son opcionales.

Formato para instrucciones
[<nombre>] instrucción [parámetros] @n

Tabla B.5 Formato de instrucción para Robel.

Tabla B.6 Tabla de instrucciones de Robel

El formato de instrucción de Robel se divide en cuatro partes, las cuales se describen a continuación:

<nombre>: este parámetro es utilizado para indicar el nombre del robot que ejecutará la instrucción. Algunas instrucciones son independientes de la existencia de un robot y en esos casos el parámetro no es necesario.

instrucción: instrucción a ser ejecutada.

parámetros: parámetros asociados a la instrucción.

@n: código de control. Para llevar un control de las instrucciones asignadas, ya sea en scripts o vía Internet, es necesario añadir un código de control denotado por @ y seguido por un identificador numérico (n) preferiblemente progresivo. Esto tiene como finalidad conocer qué instrucción se está ejecutando, y en el caso de ser un comando remoto, enviar el código de la instrucción al host (anfitrión) como mensaje de fin de ejecución. Si el comando es introducido a través de la ventana de comandos, no es necesario añadir el código de control ya que éste es asignado automáticamente.

La siguiente tabla muestra el conjunto de instrucciones que son soportadas por el Roc2008

Instrucción	Categoría	Sintaxis	Ejemplo	Descripción
Let.	movimiento	Let.	Let.	Movimiento de rotación a la izquierda
Light	movimiento	Light	Light	Movimiento de rotación a la derecha
forward	movimiento	forward	forward	Movimiento de traslación de una unidad hacia delante.
backwrd	movimiento	backward	Backward	Movimiento de traslación de una unidad hacia atrás.
mv	movimiento	mv distancia ángulo [tiempo]	mv 0 3.4	Movimiento de rotación dado por un ángulo, seguido de una traslación dada por una distancia.
mvto	movimiento	mvto posX posY [tiempo]	mvto -5.3 2.7 10	Traslación hacia la posición dada por (posX, posY)
loads	script	loads nombre_script	load test.txt	Carga el script nombre_script en la maquina virtual
script	script	script nombre_script	script test.txt	Ejecuta el script nombre_script que debe de estar cargado en la maquina virtual
ascript	script	ascript	ascript	Aborta la ejecución de un script
opsk	comunicación	opsk r/w hostname puerto	opsk w localhost 2000	Establece comunicación de entrada (r) o salida (w)
ksk	comunicación	ksk r/w	ksk w	Cierra comunicación de entrada (r) o salida (w)
ic	estado	ic posX posY ángulo	ic -5 5 3.1	Establece la posición actual del robot (cambia el origen de coordenadas relativo al robot)
sc	estado	sc	sc	Muestra la posición y orientación del robot
shs	estado	shs nombre_sensor número	shs reflective 1	Obtiene una lectura del sensor número, cuyo tipo es nombre_sensor
tap	estado	tap	tap	El robot captura una imagen de lo que esta observando en ese instante

Tabla B.6 Tabla de instrucciones de Robel.

REFERENCIAS

[Ref01] Es un dispositivo telegráfico de transmisión de datos para enviar y recibir mensajes mecanografiados punto a punto a través de un canal de comunicación simple, a menudo un par de cables de telégrafo.

[Ref02] Todas las partes físicas y tangibles de un computador, como componentes eléctricos, electrónicos, electromecánicos y mecánicos.

[Ref03] Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.

[Ref04] Software especializado para el diseño asistido por computadora

[Ref05] Aparato capaz de transformar magnitudes físicas o químicas en magnitudes eléctricas

[Ref06] Un pipeline o tubería es un conjunto de elementos procesadores de datos conectados en serie, en donde la salida de un elemento es la entrada del siguiente.

[Ref07] Es un modelo de iluminación y sombreado que asigna brillo a los puntos de una superficie modelada

[Ref08] Imagen obtenida del libro: Real-Time Rendering Edit A.K. Peters Natick, Tomas Akenine-Möller, Eric Haines: Massachusset 1991.

[Ref09] Acrónimo del inglés *picture element*, "elemento de imagen" es la menor unidad homogénea en color que forma parte de una imagen digital, ya sea esta una fotografía, un fotograma de vídeo o un gráfico.

[Ref10] El aliasing es el artefacto gráfico que se presenta al mostrar en una pantalla ciertas curvas y líneas, estas no se muestran como una linea continua si no como un tipo "sierra" o "escalón".

[Ref11] Software desarrollado por pixar para el rendering de imágenes de películas o video producciones.

[Ref12] Pixar Animation Studios es una compañía de animación por ordenador especializada en 3D, ubicada en Emeryville, California (Estados Unidos)

[Ref13] Acrónimo inglés de la expresión *Non Uniform Rational B-Splines*. Modelo matemático muy utilizado en los gráficos por ordenador para generar y representar curvas y superficies

[Ref14] Imagen obtenida de http://juank.black-byte.com/xna-colisiones-2d/

[Ref15] Método por el que en un mundo virtual en 3D se representa una cámara imaginaria, por cuyo lente se mostrara el mundo virtual y son las imágenes que se muestran en la pantalla.

[Ref16] Librerías de desarrollo 3D utilizadas para programar aplicaciones que necesitan gráficos tridimensionales.

[Ref17] Término con el que se conoce al software distribuido y desarrollado libremente. Fue utilizado por primera vez en 1998 por algunos usuarios de la comunidad del software libre, tratando de usarlo como reemplazo al ambiguo nombre original en inglés del software libre (*free software*).

[Ref18] Es una curva definida a trozos mediante polinomios.

[Ref19] Proceso de analizar una secuencia de símbolos a fin de determinar su estructura gramatical con respecto a una gramática formal dada.

[Ref20] Se denomina frame en inglés, a un fotograma o cuadro a una imagen particular dentro de una sucesión de imágenes que componen una animación.

[Ref21] Es la licencia de software otorgada principalmente para los sistemas BSD (Berkeley Software Distribution). Pertenece al grupo de licencias de software Libre.

[Ref22] En inglés *End User License Agreement* (EULA), es una licencia por la cual el uso de un producto sólo está permitido para un único usuario (el comprador).

[Ref23] Es una licencia de software libre que define los términos bajo los cuales zlib y libpng pueden ser distribuidos. La licencia es también usada por otros paquetes de software libre.

[Ref24] Licencia a aquellas obras de cualquier tipo que protegen los derechos de autoría.

[Ref25] Para mayor información visitar http://www.gimp.org/

[Ref26] Para mayor información visitar http://www.gnu.org/licenses/gpl.html

[Ref27] Para mayor información visitar http://www.gnome.org/

[Ref28] Para mayor información visitar http://es.wikipedia.org/wiki/Widgets

[Ref29] En programación, *null* resulta ser un valor especial que significa que no existe valor.

[Ref30] Para mayor información sobre DirectX visitar http://es.wikipedia.org /wiki/DirectX

[Ref31] Para mayor información visitar http://www.opengl.org/

[Ref32] Para mayor información visitar http://www.newtondynamics.com/

[Ref33] Esta información fue copiada de la tesis "INTERFAZ GRÁFICA 3D PARA SIMULAR Y CONTROLAR ROBOTS MÓVILES USANDO REALIDAD VIRTUAL" escrita por Emmanuel Hernández Hernández, Gabriel Vázquez Rodríguez

[Ref34] Robot Behaivor Language (ROBEL) lenguaje que fue diseñado en el laboratorio de Interfaces Inteligentes de la facultad de ingeniería de la UNAM, para el control y manejo de robots móviles.

[Ref35] Programa mediante el cual se pueden comunicar dos aplicaciones distintas que pueden estar en la misma computadora o en computadoras distintas.

[Ref36] Es una característica que permite a una aplicación realizar varias tareas de forma concurrente.

[Ref37] Espacio de memoria en la que se guarda información

[Ref38] Área de una ventana en la que se muestran las imágenes generadas en 3D

[Ref39] Es una serie de normas léxicas y gramaticales parecidas a la mayoría de los lenguajes de programación, pero sin llegar a la rigidez de sintaxis de estos ni a la fluidez del lenguaje coloquial.

[Ref40] El formato 3DS es propietario de AutoDesk el cual fue diseña para almacenar información de modelos tridimensionales,

[Ref41] Imagen obtenida de la tesis "INTERFAZ GRÁFICA 3D PARA SIMULAR Y CONTROLAR ROBOTS MÓVILES USANDO REALIDAD VIRTUAL" escrita por Emmanuel Hernández Hernández, Gabriel Vázquez Rodríguez

[Ref42] Es una caja que en su interior contiene todos los vértices que forman un modelo 3D.

[Ref43] Imagen obtenida de http://graphics.stanford.edu/courses/cs468-01-winter/papers/h-apwsftccd- 96.pdf

[Ref44] Es una herramienta que provee un ambiente de desarrollo para la producción y ejecución de sistemas expertos. Información obtenida de http://es.wikipedia.org/wiki/CLIPS

BIBLIOGRAFÍA

[Ake91] Tomas Akenine-Möller, Eric Haines: Real-Time Rendering. Edit A.K. Peters Natick, Massachusset 1991.

[Anm08] Anonimo. Physics engine Fuente: http://en.wikipedia.org/wiki/Physics_engine

[Ang00] Edward Angel Interactive Computer Graphics a TOP-DOWN APPROACH WITH OpenGL second Edition 2000

[Bir07] Jeremy Birn : Iluminación y Render Edición 2007 Editorial: Anaya Multimedia

[Char05] Francisco Charte Ojeda. INTRODUCCION A LA PROGRAMACION EDICION 2005 Ediciones ANAYA MULTIMEDIA (Grupo ANAYA S. A.)

[Cos92] Raúl Coss Bu Simulación un enfoque Práctico. Limusa 1992

[Fol96] James D. Foley, Andries Van Dam, Steven K. Feiner, John F. Hughes, Richard L. Phillips: Introducción a la graficacion por computador. Edit. Addison-Wesley Iberoamericana, 1996

[Fra94] Harald Frater, Paulissen Dirk: El Gran Libro de Multimedia. Edit. Marcombo 1994

[Har99] Eric Harlow Desarrollo de aplicaciones Linux con GTK+ y GDK Guía Avanzada. Prentice Hall 1999

[Hea95] Donal Hearn, M. Pauline Baker: Gráficas por Computadora. Edit. Prentice Hall Hispanoamericana 1995.

[Her02] Emmanuel Hernández Hernández, Gabriel Vázquez Rodríguez Tesis:
INTERFAZ GRÁFICA 3D PARA SIMULAR Y CONTROLAR
ROBOTS MÓVILES USANDO REALIDAD VIRTUAL
UNAM. México, 2002.

[Ins97] Daniel Insa Ghisaura, Rosario Morata Sebastián Multimedia e Internet Paraninfo 1997

[Mcc93] D. McCloy, D. M. J. Harris: Robotica una Introducción. Edit. Limusa Primera Edicio 1993.

[Ram08] María del C. Ramos, José Larios, Daniel Cervantes y Renato Leriche
Creación de ambientes virtuales inmersivos con sofware libre
http://www.oei.es/noticias/spip.php?article823

[Rui08] Juan Carlos Ruiz Pacheco
http://juank.black-byte.com/xna-colisiones-2d/

[Ser08] Fernando José Serrano García Introducción a OpenGL
http://usuarios.lycos.es/andromeda_studios/paginas/tutoriales/tutgl001.htm

[Wik01] http://es.wikipedia.org/wiki/CG

[Wik02] http://es.wikipedia.org/wiki/Direct3D

[Wik03] http://es.wikipedia.org/wiki/ATK

[Wik04] http://es.wikipedia.org/wiki/Cairo_(biblioteca)

[Wik05] http://es.wikipedia.org/wiki/Pango

[New08] http://www.newtondynamics.com/