8

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DESARROLLO DEL SOFTWARE DE UN SISTEMA PARA LA OPERACIÓN DE UN ROBOT MÓVIL EN UN AMBIENTE DE REALIDAD VIRTUAL 3D.

T E S I S P R O F E S I O N A L

PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

P R E S E N T A N:

ARTEAGA PÉREZ ALMA LETICIA

SILVA URBANO JOSÉ MANUEL

DIRECTOR :

ING. ROMÁN OSORIO COMPARÁN

CIUDAD DE MÉXICO, D.F. **2002**

PAGINACIÓN DISCONTINUA

AGRADECIMIENTOS

A la **Universidad Nacional Autónoma de México** por permitir la diversidad de ideologías y alentarnos a crear nuestros propios pensamientos, a la **Facultad de Ingeniería** y muy especialmente a nuestros maestros por brindarnos los conocimientos de Ingeniería necesarios para poder desarrollar esta Tesis.

Al **Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas** por las facilidades otorgadas tanto en equipo como en material didáctico para el desarrollo de esta Tesis.

Al **Ing. Román Osorio Comparán** por haber confiado en nosotros para el desarrollo de este proyecto, por el gran apoyo brindado como amigo y director de tesis.

Al mis padres **Alma y Bruno** por respetar mis decisiones al iniciar este gran camino, **GRACIAS**

A la **Familia Pérez Laines** por darme ese apoyo y confianza cuando más lo he necesitado, GRACIAS.

A **Alejandro Abaunza,** GRACIAS POR TODO.

A todo mi familia, amigos, compañeros y a todos aquellos que de alguna forma me han apoyado DIOS LOS BENDIGA.

A **José Manuel** por ser el gran amigo, el gran compañero y sobre todo por esa gran paciencia que ha tenido en los momentos buenos y malos.

Alma Leticia Arteaga Pérez

Gracias.

A Dios, Aurora y Francisco porque con su amor me dieron la vida.

A mi mamá, mi papá, paco, chayo, Victor, Samantha, Ángeles y Beni porque a pesar de todo seguimos juntos, porque son mi primer familia.

A Gaby, Mica, Armando, Guillermo y Juan porque se atrevieron a hacer (o quieren hacer) su familia de la mía.

A Sara, Ángel, al enano, la chaparra, Vere, Andrea, Mariana (y los que vengan) por darnos alegría y prosperidad.

No hubiese sido posible lograr esto sin ustedes.

A ti Alma, porque me aguantaste hasta el final. Esto no hubiese sido igual sin ti.

DESARROLLO DEL SOFTWARE DE UN SISTEMA PARA LA OPERACIÓN DE UN ROBOT MÓVIL EN UN AMBIENTE DE REALIDAD VIRTUAL 3D

ÍNDICE

DESARROLLO DEL SOFTWARE DE UN SISTEMA PARA LA OPERACIÓN DE UN ROBOT MÓVIL EN UN AMBIENTE DE REALIDAD VIRTUAL 3D

Problema: Realizar el estudio de las diferentes herramientas de Diseño Asistido por Computadora (CAD) y animación gráfica para lograr el desarrollo en software de un ambiente gráfico tridimensional que sea capaz de generar ambientes modificables, basándose en una plataforma Windows y programado en lenguaje Visual Basic.

Objetivo: Desarrollar por medio de las herramientas adecuadas más actuales la generación de ambientes gráficos modificables utilizando la plataforma Windows y basados en la programación con Visual Basic, las cuales permitan generar la graficación del escenario y la animación del objeto (robot móvil) en tres dimensiones con el objetivo de hacerlo un objeto tele-operado en su ambiente real y en esta forma lograr que se produzca Realidad Virtual.

INTRODUCCIÓN

El uso de los robots en la industria de la fabricación es hoy día una realidad extendida en este país. Sectores tales como los del automóvil, transformados metálicos, equipos y aparatos eléctricos, productos químicos y plásticos, etc., han incorporado a sus procesos el uso de robots industriales en diversas aplicaciones: soldadura por puntos, soldadura al arco, manipulación de materiales, montajes mecánicos, carga y descarga de máquinas, etc. Existen varios sectores en la industria que aún no han dado entrada a instalaciones robotizadas por ello podemos decir que aún existen sectores marginales en el mercado de los robots.

Desafortunadamente, el desarrollo de robots para aplicaciones especiales no representa un mercado de interés para los fabricantes de robots industriales. Esto hace que la mayor parte de los robots actualmente operativos en centrales nucleares hayan sido desarrollados por los propios fabricantes de reactores, sean adaptación de aplicaciones diseñadas con otro objetivo o provengan de trabajos de investigación desarrollados por las propias empresas nucleares en colaboración con expertos en robótica.

La constatación de la dificultad de programar un robot para operaciones complejas y con capacidad de adaptación a situaciones cambiantes ha hecho resurgir la idea de la tele-operación. Recordemos que la tele-operación consiste en el manejo de un robot a distancia el cual obedece a las instrucciones que proporciona el usuario ya sea a través de otro robot o un sistema de cómputo que las genere; además la tele-operación permite el manejo del robot remoto con una mayor facilidad si este proporciona algun dato sobre su estado ya sea a través de una imagen o coordenadas que lo ubiquen en un entorno. Además, el costo para estas aplicaciones es muy alto, ya que en la actualidad existen robots tele-operados, pero esta tele-operación se da por medio de una plataforma especializada, lo que requiere que en la máquina en donde se desee hacer la tele-operación tenga este sistema operativo.

Es por esto, que el objetivo de la tesis tiene como finalidad generar un software que permita tener el manejo del robot, hacer ambientes tridimensionales cambiantes, bajo una plataforma muy usual como lo es Windows. Además de que el robot virtual pueda interactuar con un ambiente prediseñado y al mismo tiempo pueda ser tele-operado.

Para poder redactar de forma entendible el desarrollo de la tesis ésta se ha dividido en 5 capítulos, los cuales van presentando la forma en la que se fue desarrollando la tesis hasta llegar a tener el software que se pretende, el capitulo1 nos habla de las diferentes herramientas y lenguajes que existen para desarrollo de videojuegos y ambientes gráficos, esto con la finalidad de conocer las diferentes herramientas que existen en la actualidad y poder escoger la que mejor satisfaga nuestro problema. El capitulo 2 habla de algunos paquetes de software de graficación que se pueden utilizar para la creación de objetos y ambientes tridimensionales, también se habla del componente ActiveX que ayudará con el desarrollo del software, sobre todo a entender como se desarrollan los vínculos entre el código y el componente ActiveX. A continuación se estudian las comunicaciones a través del puerto serie lo cual permitirá comunicar el robot con la computadora. La teoría de transformaciones

geométricas sirve para dar una idea del tipo de algoritmos que intervienen en la animación de un objeto. Para terminar este capítulo es necesario conocer los conceptos de graficación computacionales que son una parte fundamental para el desarrollo de la aplicación. En el capítulo 3 se desarrolla la teoría mínima de realidad virtual, los diferentes conceptos que existen y la interacción que tiene con el tiempo real, también se habla de las diferentes teórias existentes en el desarrollo de algoritmos de detección de colisiones la muestra de los diferentes algoritmos ayudará a entender cuál es el que se puede aplicar para el desarrollo del software. El capítulo 4 trata de explicar la aplicación que se obtuvo analizando únicamente aquellas funciones que tienen verdadera importancia dejando de lado aquellas que son inherentes al desarrollo de cualquier aplicación cuando se utiliza Visual Basic como lenguaje de programación

El capítulo 5 presenta los resultados de las pruebas de ajuste con el movimiento de la llanta del robot ya que esos datos son necesarios para lograr una buena simulación. También se reporta un breve esquema y explicación del programa que se encarga de atender las peticiones de movimiento en el robot.

Para finalizar con el desarrollo de la tesis se presentan las conclusiones que se obtuvieron en la investigación, desarrollo y pruebas de este software.

Como apoyo al buen uso de este software se agrega una serie de apéndices que permiten seguir paso a paso las condiciones mínimas para generar un archivo .3ds desde 3dStudioMax(sin embargo se puede generar desde cualquiera otra aplicación que lo soporte). El archivo .3ds es requisito para poder pasar a un archivo .X que es el archivo que se puede cargar para generar un ambiente.

También se agrega un manual de usuario que describe las tareas que se pueden desarrollar con el software, así como los requisitos del sistema para la instalación del mismo.

1.0 HERRAMIENTAS DE DESARROLLO PARA VIDEOJUEGOS

Para el desarrollo de esta tesis es importante conocer las herramientas existentes que se están utilizando en la creación de videojuegos, ya que en este entendimiento se puede dar una introducción a los ambientes tridimensionales que dará el panorama general para lograr la elección adecuada de las herramientas de desarrollo. De esta forma a continuación se presentan en primer lugar las herramientas más conocidas para el desarrollo de videojuegos:

1.0.1 DIV Game Studio

DIV es un lenguaje de programación especialmente diseñado para videojuegos, aunque se pueden programar todo tipo de aplicaciones. DIV fue creado por Daniel Navarro como proyecto de fin de carrera. Las características de DIV son:

- Lenguaje no secuencial, con procesos ejecutándose en paralelo, lo que facilita enormemente la programación de videojuegos.
- Permite crear cualquier tipo de juegos en 2D(solamente en 2D).
- Permite crear regiones de pantalla.
- Soporta gráficos JPGE, PCX y BMP, además de vídeo en formato FLI/FLC.
- Reproduce WAV, PCM y MOD'S(mod, xm, etc).
- Cambia el tamaño, ángulo, transparencia, flags, etc.
- Funciones para manejo de texto.
- Estructuras, matrices multidimensionales, punteros y variables integer, word, byte y string.
- Función de encriptación de datos.
- Posibilidad de ampliar el lenguaje con DLL's.
- Esta basado en MS-DOS.
- NO se pueden hacer juegos en 3D(para ello es necesario utilizar DarkBasic).

1.0.2 Fénix

Es un lenguaje pseudo-interpretado diseñado para programar juegos 2D. Para ello incluye una extensa libreria dedicada a los juegos que permite programar únicamente la lógica de movimientos de los gráficos y objetos del juegos, mientras cosas como la visualización en pantalla de gráficos y sprites, o la reproducción del sonido, corren a cargo del interprete incluido.

El lenguaje es un pascal reducido con algunas diferencias notables e influencias de otros lenguajes. Además es GNU (Frase recursiva que representa al proyecto encargado de la protección y creación del software de libre distribución), y el código fuente se puede modificar para adaptarse a las necesidades de cada programador(hay versiones de Fénix no oficiales, como la Bar Edition de Beorn). Sirve para programar juegos en Windows y Linux de 16 bits de color utilizando la sintaxis de DIV. Las caracteristicas de Fénix son las siguientes:

- Tanto Fénix como los juegos creados en él funcionan bajo Windows y Linux.
- 8 bits de color(256) ó 16 bits.
- Soporte para archivos PNG.
- Reproducción de MPEG(en el Bar Edition).
- Posibilidad de #includes.
- Tiene las funciones principales que son:
  - Detección de colisiones
  - Reproducción de WAV, PCM y MOD's
  - Reproducción de FLC/FLI.
  - Encriptación/Desencriptación de datos
  - Manejo de ficheros.

1.0.3 BlitzBasic

BlitzBasic es un compilador de juegos 2D que utiliza DirectX 7. Está basado en el lenguaje Basic(aquí sólo se manejan gráficos 2D). Sus características principales son:

- Los juegos creados son para Windows 9X/Me/2000 con DirectX 7 o superior.
- Diferentes funciones de colisión, según la precisión que necesitemos.
- Manejo de ficheros.
- Funciones multijugador (LAN y TCP/IP).
- Soporte para WAV, MIDI, MP3 y MOD's.
- Soporte para BMP, PNG Y JPEG.
- 16, 24, 32 BITS DE COLORES.
- Juegos en pantalla completa o modo ventana.
- Manejo de buffers.

1.0.4 DarkBasic

Es un lenguaje de alto nivel que como su nombre lo indica está basado en el lenguaje Basic, sólo que esta versión está basada en DirectX y orientada a la creación de videojuegos en 3D.

La plataforma para que compile DarkBasic es Windows, aunque en realidad no compile binarios, sino, como Div o Fénix, crea un código intermedio que después es interpretado en tiempo de ejecución, siendo esto algo más lento que un programa compilado a código máquina. Las principales características son:

- Aprovecha las capacidades de las aceleradoras 3d mediante Direct3d.
- Sonido tridimensional
- Soporta archivos de imagen como BMP, JPG, PNG, ETC.
- Importa de manera directa archivos .3ds o .X
- Reproducción de video en formato AVI

Recordemos que el objetivo es tener ambientes gráficos modificables para correr en plataforma Windows y desarrollar el software en Visual Basic (VB). Por lo que las anteriores herramientas para el desarrollo de videojuegos no son muy apropiadas, ya que las dos últimas corren en Windows pero no cumplen con la limitante de programarse bajo VB.

1.1 LENGUAJES DE PROGRAMACIÓN MÁS UTILIZADOS EN LA PROGRAMACIÓN DE AMBIENTES GRÁFICOS

Como se acaba de analizar en la sección anterior, existen ya diseñadas máquinas gráficas que permiten el desarrollo de video juegos de la manera más simple posible pero en nuestro caso necesitaremos bajar el nivel de programación por necesitar la programación en un lenguaje determinado. De esta forma necesitaremos desarrollar las funciones gráficas partiendo directamente de la Interfaz Gráfica que sea compatible con VB y desarrollar las funciones desde ese nivel. Por lo anterior, la siguiente sección se encargará de enunciar brevemente algunas de las características que cumplen los lenguajes de programación más utilizados.

En general podríamos decir que existen principalmente cuatro lenguajes que se utilizan para desarrollar ambientes gráficos en el ámbito profesional. El lenguaje Ensamblador, C, C++ y Java; esto es porque principalmente lo que se busca en el manejo de gráficos es la velocidad con la que se ejecuta el código y éstos son los lenguajes que se caracterizan por ello:

1.1.1 Ensamblador

Es el lenguaje más poderoso que hay y con el que se puede obtener el máximo rendimiento del microprocesador de una computadora, sin embargo, tiene el defecto de ser muy difícil, tedioso y lento para trabajar con él. No es un buen lenguaje para programar todo un ambiente gráfico aunque sirve para optimizar su desempeño.

1.1.2 C

Es un lenguaje de programación estructurado, sin duda el más rápido después del lenguaje ensamblador y es hasta ahora el lenguaje que mas se utiliza en las comunidades de desarrolladores de software.

1.1.3 C++

Se desarrolló como una extensión de C, tiene algunas cuestiones mejoradas como un control más estricto en el manejo de tipos de datos y otras características que ayudan a la programación libre de errores, además de esto está diseñado para trabajar orientado a objetos y/o en forma estructurada. En general puede llegar a ser tan rápido como C, sin embargo si se maneja herencia múltiple, funciones virtuales y polimorfismos inadecuadamente puede llegar a ser un poco más lento.

1.1.4 Java

Es un lenguaje 100% orientado a objetos. Una de sus principales características es que es independiente de la plataforma debido a que al compilar su código fuente se transforma en un código binario independiente de la plataforma, y al momento de querer ejecutarlo este código binario es leído por un intérprete que lo procesará y le dará las instrucciones adecuadas al procesador. De este modo un mismo código puede correr en cualquier computadora siempre que exista un intérprete Java en dicha máquina.

La desventaja de esto es que debido a que el código binario primero tiene que ser interpretado antes de ser ejecutado en el procesador, éste se vuelve mas lento y si hay algo importante en la programación de ambientes gráficos es la velocidad. La alternativa que ofrece Java para evitar el uso del intérprete es la utilización de compiladores Just in Time que transforman el código fuente al código nativo de la máquina sin embargo se elimina la independencia de la plataforma.

Por otra parte, Java ya cuenta con su propia API gráfica incluida (el AWT), y de este modo el API queda automáticamente elegida. En general, Java se usa más bien para juegos en línea más que para desarrollo de ambientes gráficos para PC.

1.1.5 Visual Basic(VB)

Hablando del rendimiento que se puede obtener utilizando VB en comparación con C y C++ se puede asegurar que los programas se ejecutan casi a la misma velocidad, alrededor del 10% de pérdida en rendimiento, siempre dependiendo de la configuración final que se le asigne al ejecutable a la hora de compilarlo.

No existe una respuesta definida para determinar cuál es el mejor lenguaje para programar ambientes gráficos ni qué API gráfica es la mejor. Cada lenguaje y API tiene sus propias características con un enfoque propio para resolver problemas; decidir si es bueno o malo es relativo a lo que se busca o el tipo de problema con el que se cuente.

Después de haber analizado los diferentes lenguajes de programación que existen debemos recordar que por especificaciones del problema el lenguaje de programación que se utilizara es Visual Basic, ya que por motivos de compatibilidad con otros programas ya desarrollados es necesaria la utilización de este lenguaje. Con esto ya se tiene resuelto el problema del lenguaje y nos da pie para el análisis del API a utilizar.

1.2 INTERFACES PARA PROGRAMACIÓN DE APLICACIONES (API'S) MÁS UTILIZADAS EN AMBIENTES GRÁFICOS

Una **API** (Aplication Programers Interface) desde el punto de vista técnico es la forma en la que se les presentan al programador las rutinas para llevar acabo cierto trabajo. Por ejemplo, en Java para escribir el texto *"Voy a ser un gran programador de video juegos"* se utiliza la instrucción:

```
System.out.println("Voy a ser un gran programador de video juegos");
```

En cambio en C:

```
printf("Voy a ser un gran programador de video juegos\n");
```

Y en C++ :

```
cout << "Voy a ser un gran programador de video juegos" << endl;
```

Como se puede ver Java, C y C++ le ofrecen al programador realizar el mismo trabajo (en este ejemplo imprimir en pantalla) escribiendo las instrucciones de diferente forma en cada uno, esto es... tienen diferente API.

Sin embargo comúnmente la gente que se dedica a programar ambientes gráficos se refiere a una API como un conjunto de *rutinas preconstruidas* para realizar un trabajo específico, como dibujar en pantalla, manipular sonidos, etc., el término correcto para este conjunto de rutinas es ***Librería*** y también es muy usado.

Principalmente esta confusión se debe a que cuando estamos hablando de diferentes librerías que se dedican a hacer el mismo trabajo (por ejemplo manipulación de gráficos) en la mayoría de los casos, como es de pensarse, cada una de ellas tiene su propia API, entonces para hacer referencia a alguna de estas resultaría lo mismo llamarle API o Librería indistintamente.

Una librería es un conjunto de rutinas (una rutina es un conjunto de instrucciones), previamente construidas, que se enfocan en realizar algún trabajo en especial, para que así el programador ya no tenga que preocuparse por los detalles de cómo hacer ese trabajo, simplemente manda llamar dentro de su propio código las instrucciones adecuadas que hay en la librería para hacer lo que él busca y ya, las instrucciones harán lo que deban hacer sin que nadie tenga que preocuparse por cómo lo hacen de esta forma se le facilita el trabajo al programador, dejando que se pueda enfocar más en lo que es su programa en sí.

Como una librería no es parte del código del programa, para poder utilizarla es necesario, al momento de compilar el programa, *ligarlo* con la librería, es decir, hacer saber al programa que las rutinas están en alguna otra parte, y por supuesto, en qué parte están.

Existen dos formas en las que el código y la librería pueden quedar ligados, en forma **dinámica** y en forma **estática**, y cada una tiene sus propias ventajas y desventajas.

Cuando se liga en forma estática se incluyen en el ejecutable las rutinas que se utilizan de la librería, de modo que todo queda guardado en un mismo archivo; de este modo cuando el programa se ejecute se cargará en memoria al mismo tiempo que las rutinas de la librería, y siempre se tendrán a la mano no importa cuándo, cómo, dónde ni porque, el ejecutable será "autosuficiente" siempre, el problema cuando estas ligando en forma estática es que el tamaño del ejecutable puede ocupar más espacio.

Al ligar en forma dinámica lo que se hace es indicarle al programa que cuando se esté ejecutando busque en algún directorio de la máquina local la librería que necesita; así el ejecutable ya no ocupará tanto espacio, y cuando necesite utilizar alguna rutina la cargará directamente de la librería que se encuentra en la máquina donde está corriendo, el problema con esto es que puede o no existir dicha librería en la máquina donde está corriendo, obviamente, causará que el programa pueda o no correr.

Algunos ejemplos de las APIS más utilizadas son:

1.2.1 Graphic Library (OpenGL)

OpenGL fue creado por Silicon Graphics en 1992. Es un conjunto de librerías que son utilizadas a través de lenguajes de programación para conseguir una interfaz software entre las aplicaciones y el hardware gráfico. Esta diseñado para poder usarse en una gran variedad de sistemas operativos, es decir, que el mismo código que se usa para desarrollo en Windows puede ser utilizado para desarrollo en Macintosh, Linux, etc., sin tener que modificarlo o modificándolo muy poco, esto puede significar una gran ventaja para cualquier compañía en especial en estos días donde se busca abarcar el mayor mercado posible.

OpenGL es una API de gráficos de bajo nivel implementada en dos DLL's, existen dos versiones de esas DLL's, la de Microsoft y la de SGI. Las DLL's de Microsoft son: opengl32.dll y glu32.dll; las DLL's de SGI son: opengl.dll y glu.dll. Si se utiliza NT, debe tenerse la versión Microsoft de las DLL's. Las DLL's de SGI ya no tienen soporte.

Para usar OpenGL desde Visual Basic, la única cosa que se necesita es una librería de tipos; una vez instalada, se pueden comenzar a llamar las funciones de OpenGL directamente desde VB.

La mayoría de los ejemplos públicos de OpenGL se han hecho con la librería GLUT de Kilgard, pero la versión Win32 de GLUT 3.2 no es completamente compatible con Visual Basic. En su lugar se puede obtener glxCtl, con un control ActiveX genérico para OpenGL escrito en Visual Basic 5. El control maneja la habilitación de la ventana OpenGL y dispara varios eventos. La glxDtl está estructurada, entonces esos ejemplos GLUT pueden fácilmente ser portados a Visual Basic. Incluye versiones Visual Basic de las rutinas de contorno de GLUT (Solidxxx, Wirexxx) y un puerto de la librería de extrusión GLE. También provee hasta el mínimo soporte para paletas, bitmaps y archivos de imágenes RGB.

La librería de tipos incluye declaraciones para GLUT 3.6. Así se puede mandar llamar a cualquier función de GLUT en VB. GLUT 3.5 da soporte a Visual Basic, pero se han encontrado problemas en gluProject, gluUnProject, gluPickMstrix y gluLoadSamplingMatrices ya que no compilan apropiadamente.

1.2.2 Simple Direct Media Layer (SDL)

Simple Direct Media Layer(SDL) es una librería API multimedia la cual provee acceso de bajo nivel a los sistemas de video frame buffer, salida de sonido y dispositivos de entrada incluyendo teclado, ratón y joystick. Es usado en emuladores y visores MPEG para manejar

los gráficos y sonidos de esos programas, pero SDL ha llegado a concebirse mayormente como un kit de herramientas esencial para el desarrollo de video juegos y aplicaciones con gráficos en Linux.

SDL es similar a la API DirectX de Microsoft. La gran diferencia es que SDL soporta múltiples sistemas operativos. Además de Linux, está disponible para BeOS, MacOS, Solaris, FreeBSD, IRIX y las versiones Windows de 32 bits. Adicionalmente, la API tiene vínculos a otros lenguajes, algunos de los cuales son Perl, PHP y Python. Escrito en C (y también trabajando nativamente con C++), SDL está disponible bajo la licencia pública de GNU.

SDL es una API mucho más fácil de usar que también está disponible para un rango más amplio de plataformas. Con SDL no se puede hacer todo lo que se puede hacer con DirectX, pero la API es lo suficientemente completa para que la usen varias docenas de títulos comerciales además de un innumerable número de aplicaciones no relacionadas directamente con el desarrollo de video juegos.

Después de desarrollar una búsqueda para utilizar Visual Basic con SDL, no se encontró ninguna aplicación de este tipo por lo que se da por hecho que SDL no es soportado o compatible con Visual Basic y por ello podemos decir que en el desarrollo de la solución del problema no se puede tomar en cuenta.

1.2.3 Allegro

Allegro es una librería portable principalmente enfocada a video juegos y programación multimedia, originalmente escrita por Shawn Hargreaves para el compilador DJGPP en una mezcla de C y ensamblador. Sus siglas significan «Allegro Low LEvel Game ROutines» [rutinas de bajo nivel para videojuegos].

Una de las características que tiene Allegro es que se trata de un proyecto "Open Source", es decir el código fuente está disponible para prácticamente cualquier persona que quiera echarle un vistazo, y puede arreglarse a gusto propio, por lo tanto actualmente existe una gran cantidad de personas que continuamente, están aportando algo nuevo a Allegro en sus ratos libres, ya sea mejorando su desempeño o añadiéndole nuevas funciones.

Una de las principales ventajas que tiene es que fue específicamente diseñado para la programación de video juegos, por lo tanto todas las rutinas que utiliza están hechas para ser fáciles de manejar y sobretodo eficiente desde el punto de vista de un juego.

Con Allegro se puede manipular prácticamente toda la multimedia del juego, tal como la Entrada/Salida, gráficos, midis (música electrónica), efectos de sonido y de tiempo, además de que tiene ya preconstruidas ciertas funciones para realizar algunos efectos que normalmente se tendrían que programar a mano.

Otra característica que tiene es que fue diseñada para ser portable a diferentes plataformas como Windows, Linux, BeOS, etc., sin tener que cambiar ni siquiera una sola línea de código; claro que muchas veces la portabilidad de un código depende de lo que escribe el programador, pero si se hace eficientemente no se tendrán problemas.

Este API no es compatible con Visual Basic porque se desarrolló para el compilador DJGPP que es un compilador de 32 bits para el procesador i386 (o más recientes) basado sobre plataforma DOS. Con esto queda descartado como posible API ya que no resuelve el problema de ser compatible con Visual Basic.

1.2.4 DirectX

DirectX fue diseñada por Microsoft en 1995 con el fin de permitir a los programadores escribir programas multimedia para sistemas Windows de manera fácil y sin tener que preocuparse sobre qué hardware está corriendo y cómo funciona, como es el caso del tipo de tarjeta de video, etc.

Una de las principales ventajas que tiene es que no está hecho solamente para la parte gráfica, sino que con DirectX se puede manejar toda la multimedia de la computadora como sonidos, música, dispositivos de Entrada / Salida como teclado, joystick, y varias otras cosas más.

Sin embargo, DirectX tiene una gran desventaja: no es portable, es decir, una vez que se ha terminado de programar un ambiente gráfico estará condenado a trabajar solamente en Windows, y cuando halla que convertirlo para que pueda correr en otras computadoras que usen Mac o Linux se necesitarán muchos cambios al código, lo cual no es nada deseable a menos que se sepa de antemano que el único mercado al que va dirigida la aplicación son personas con una computadora con Windows.

De esta manera, podemos ver que las opciones son varias pero que todas ellas tienen soporte y se orientan hacia los lenguajes de programación bajo C y C++. Por otra parte, en los últimos tiempos se ha visto un interés por parte de los programadores en desarrollar aplicaciones multimedia con el uso de DirectX a través de Visual Basic. Esto debido a que Visual Basic se distingue por ser un lenguaje de programación muy natural y fácil respecto de C y C++. Aunque existen muchos comentarios de la complejidad que presenta el trabajar con DirectX, también se acepta que se trata de la API más potente disponible para Windows a pesar de no ser multiplataforma.

Por lo anterior se elige, para el desarrollo de nuestro software, DirectX como API a utilizar para programar bajo Windows por lo que es necesario que se haga una presentación más amplia de lo que es DirectX.

Justificación

La liberación actual de DirectX reitera el firme soporte para Windows como la plataforma para aplicaciones de juegos.

Microsoft DirectX es un grupo de tecnologías diseñadas por Microsoft para hacer de las computadoras basadas en Windows una plataforma ideal para correr y mostrar aplicaciones como juegos que son ricos en gráficos de tiempo real, gran colorido, tres dimensiones, video, música interactiva y sonido envolvente. Integrado directamente en la

familia de sistemas operativos Windows, DirectX es una parte integral de Microsoft Windows 98, Microsoft Windows Millenium (Me).

DirectX les proporciona a los desarrolladores de software un grupo consistente de API's (Interfazs para Programación de Aplicaciones) que les proveen un acceso mejorado a las características avanzadas del hardware de alto desempeño como los aceleradores de gráficos 3D y tarjetas de sonido. Esas API's controlan lo que se llaman "funciones de bajo nivel", incluyendo manejo de memoria gráfica y rendering, soporte para dispositivos de entrada como joystick, teclado y ratones y control de combinación de sonidos y salida de audio. Las funciones de bajo nivel están agrupadas en componentes que integran DirectX: Microsoft Direct3D, Microsoft DirectDraw, Microsoft DirectImput, Microsoft DirectMusic, Microsoft DirectPlay, Microsoft DirectSound y Microsoft DirectShow.

DirectX provee a los desarrolladores herramientas que los ayudan a obtener el mejor desempeño posible de las máquinas que utilizan, provee mecanismos explícitos para que las aplicaciones determinen las capacidades actuales del hardware del sistema para que puedan habilitarlo para proporcionar un desempeño óptimo.

Antes de DirectX, los desarrolladores que creaban aplicaciones multimedia para PC's tenían que especializar sus productos para que pudieran trabajar en una amplia gama de dispositivos y configuraciones disponibles en las máquinas Windows. DirectX provee una "Capa de Abstracción del hardware" (HAL) que utiliza drivers de software para comunicarse entre el software del juego y el hardware de la computadora.

Como resultado, los desarrolladores pueden usar un único paradigma además de consistente de DirectX para implantar sus productos a lo largo de un amplio rango de dispositivos hardware y configuraciones.

La tecnología de gráficos en PC está avanzando más rápidamente que cualquier otra plataforma. Microsoft Direc3D permite a las computadoras que trabajan bajo Windows, mostrar las más avanzadas e interactivas gráficas tridimensionales disponibles en cualquier plataforma. Habilita a las aplicaciones para acceder a las más recientes tecnologías de aceleración 3D diseñadas para rendering, así como gráficos.

Presentación

Desde la aparición en el mercado de las librerías DirectX de Microsoft la programación de video juegos bajo Windows ha cambiado radicalmente. Pocos programas se dirigen ya hacia el paleolítico MS-DOS, puesto que la tecnología avanza a pasos agigantados y este sufrido sistema operativo tiene cada vez menos soporte para el nuevo hardware que se va implementando.

Como ejemplo, hay que destacar las nuevas aceleradoras 3D. Cualquier juego bajo MS-DOS que quiera utilizar las posibilidades de estas extraordinarias tarjetas gráficas, necesita ser programado exclusivamente para cada una de ellas, es decir, que la programación efectuada en relación con una tarjeta no funcionara con otra de distinto fabricante. Este hecho no ocurre bajo Windows 98, puesto que cada tarjeta implementa soporte para las librerías

DirectX y el programador sólo tiene que centrarse en esta API, sin importarle el tipo o clase de tarjeta gráfica que tenga instalada el PC.

De este modo, se evita el caos gráfico que existía en MS-DOS cuando aparecieron las tarjetas SVGA y su compatibilidad con el estándar VESA. Windows se presenta realmente como la plataforma ideal para la creación de video juegos, puesto que todo el hardware que tenga instalado nuestro PC será registrado en el sistema operativo, y si funciona con él, también funcionará al utilizar DirectX.

A continuación veremos una tabla resumida de los componentes incluidos en DirectX:

- API DirectPlay. Proporciona unos medios de protocolo independientes para videojuegos de varios jugadores en Internet y otros servicios Online. La interfaz DirectPlayLobby permite la creación de puntos de encuentro Online, donde los usuarios pueden reunirse y jugar. DirectPlay proporciona conectividad transparente a los jugadores aunque utilicen diferentes proveedores de servicio.
- API DirectInput. Permite recoger información en tiempo real del ratón, teclado y joystick.
- API DirectSound. Ofrece drivers de sonido con soporte Dolby y un mezclador en modo kernel, lo que permite consumir menos tiempo de CPU. El mezclador también posibilita un rendimiento óptimo de sonido posicional en 3D, lo que permite a los desarrolladores de juegos situar eventos de sonido en cualquier punto del espacio perceptual del usuario.
- API DirectDraw. Proporciona capacidades de gráficos en 2D y sirve como base de procesos de Rendering para otros servicios de video.
- API Direct3D. Es un motor de Rendering para gráficos 3D en tiempo real que integra un API de bajo nivel para el render de polígonos y vértices, y uno de alto nivel para la manipulación de escenas complejas en 3D. Direct3D incorpora un Rasterizer de MMX para la nueva generación de Intel de CPU's multimedia, así como un nuevo algoritmo RAMP de color que mejora la calidad visual.

Microsoft creó DirectX para posibilitar la creación de aplicaciones de alto rendimiento basadas en Windows, además de acceso en tiempo real al hardware instalado en los PC's de hoy en día y los que se fabriquen en el futuro. DirectX posee una interfaz consistente entre el hardware y las aplicaciones, por lo que reduce la complejidad de las instalaciones y configuraciones, además de utilizar los recursos hardware en su mayor rendimiento. Con el uso de esta interfaz, los desarrolladores aprovechan las características del hardware sin conocer los detalles internos del mismo.

Un juego de alto rendimiento creado con DirectX aprovechará las siguientes características:

-

- Tarjetas aceleradoras diseñadas específicamente para aprovechar el rendimiento gráfico en 2D y 3D.
- Plug and Play de dispositivos hardware.
- Servicios de comunicaciones construidos bajo Windows, incluyendo la API DirectPlay.
- Diversos recursos instalados en el sistema que ejecute la aplicación bajo el entorno Windows.

2.0 SOFTWARE DE GRAFICACIÓN

En el desarrollo de ambientes gráficos tridimensionales normalmente se utilizan aplicaciones de apoyo que permiten el diseño externo de objetos complejos que formarán parte de la aplicación ya sea ésta un video juego o alguna aplicación multimedia, para lo cuál, en última instancia, deberán importarse desde la aplicación que las utilizará. Así, se debe elegir algún software de graficación que nos asista en la creación de los objetos tridimensionales.

Existen numerosos y conocidos programas en el mercado para crear imágenes (Fractal Design Painter, LigthWave 3D, 3D Studio MAX, Photoshop, Corel Draw, AutoCAD,...), por mencionar algunos. Habiendo mucho software gráfico con el cual podemos interactuar haremos una selección de acuerdo a los requerimientos de nuestro problema que sería el trabajo con Windows y compatibilidad con Visual Basic comunicándose a través de DirectX. Es por ello que sólo mencionaremos a aquellos que parecen ser los más utilizados en el área.

2.0.1 LigthWave 3D

NewTek, Inc., Líder manufacturador de la industria de la animación 3D y productos de video, anuncio que LigthWave 3D, ganador del premio NewTek al mejor producto de animación y gráficos 3D, soportará DirectX 8.0, la API Multimedia de Microsoft incrustada en el sistema operativo Windows.

LigthWave 3D se desenvuelve muy bien con DirectX 8.0 y también se integra muy bien con el grupo de las nuevas herramientas de DirectX. NewTek está y continuará trabajando cercanamente con Microsoft en el mejoramiento del pipeline para la creación de contenidos de DirectX.

El exportador DirectX 8.0 para LigthWave 3D permitirá a los desarrolladores de juegos tomar los objetos de LigthWave 3D y las animaciones directamente en el ambiente de juego. Toma gran ventaja de los Mapas de Vértices y sistemas verticales para deformaciones de piel natural en un ambiente de tiempo real. El DirectX de Microsoft está siendo utilizado ampliamente por muchos programadores de juegos por proveer un estándar de desarrollo bajo la plataforma de PC's basadas en Windows permitiendo a los desarrolladores de software acceder a características especializadas del hardware sin tener que escribir código específico para el hardware. Esto es especialmente útil para los desarrolladores en tanto que toman ventaja de los numerosos avances en la tecnología de gráficas computacionales.

El exportador de DirectX 8.0 permitirá a los desarrolladores de juegos utilizar LigthWave 3D en sus pipelines para exportar la alta calidad y foto realismo generado con este software directamente en las máquinas de juegos. Actualmente LigthWave está disponible para Windows y Macintosh.

2.0.2 SoftImage 3D

Es una herramienta que trata de conectar la parte artística con la técnica para generar un buen juego de video. Es un software con el que podemos desarrollar la animación en 3D como lo son los efectos especiales, comerciales, videojuegos y film. SoftImage provee de herramientas y exportadores para plataformas específicas incluyendo PC / DirectX. Exprime al máximo las ventajas que ofrece cada plataforma en cuanto a render y herramientas de visualización.

SoftImage 3D provee de herramientas especializadas para la producción de video juegos y ambientes gráficos. SoftImage incluye la capacidad de Exportar, Importar y Visualizar sus objetos a través de Direct3D desde DirectX 5. Cuenta con herramientas para permitir moverse entre SoftImage y los formatos estándar de archivos de la industria tanto para imágenes como para objetos, contiene un convertidor para Exportar e Importar desde o a 3D Studio. Acepta una gran variedad de formatos de imagen y render.

2.0.3 3D Studio Max

Es soportado por la más grande comunidad de desarrolladores que cualquier aplicación 3D, con la más amplia interacción con terceras aplicaciones integradas. Contiene un ambiente de desarrollo adecuado para la siguiente generación de video juegos con soporte Direct3D, multitexturizado por caras, mapeo de opacidad y transparencia real. Las gráficas para puertos interactivos soportan la aceleración de hardware a través de Direct3D y OpenGL o simulación por software para cualquier pantalla Windows.

En realidad existe una gran cantidad de software que puede asistirnos en la creación de los objetos tridimensionales, pero cabe señalar que el único requisito que deben cumplir los objetos 3D que deseen ser insertados en el ambiente gráfico deberán ser archivos con formato de DirectX (que son los archivos *.X). Para obtener este tipo de archivos es necesario obtener de alguno de los paquetes gráficos el archivo con formato *.3ds para que este archivo pueda ser transformado a través de algún convertidor (como son conv3ds.exe y XskinExp.dle específico para 3D Studio Max) al formato *.X . Así pues, no nos limitaremos en el desarrollo de la aplicación hacia el uso de objetos creados con un software gráfico específico; sin embargo, utilizaremos 3DS Max 4 para el diseño de los objetos simplemente por ser el software al cual se tuvo acceso y por ser una de las herramientas más ampliamente utilizadas en el área de desarrollo de video juegos y aplicaciones gráficas.

Realmente no se utilizarán todas las capacidades que ofrece este software, porque DirectX solamente acepta el objeto con las características que se mencionan en el apéndice 1.

2.0.2 SoftImage 3D

Es una herramienta que trata de conectar la parte artística con la técnica para generar un buen juego de video. Es un software con el que podemos desarrollar la animación en 3D como lo son los efectos especiales, comerciales, videojuegos y film. SoftImage provee de herramientas y exportadores para plataformas específicas incluyendo PC / DirectX. Exprime al máximo las ventajas que ofrece cada plataforma en cuanto a render y herramientas de visualización.

SoftImage 3D provee de herramientas especializadas para la producción de video juegos y ambientes gráficos. SoftImage incluye la capacidad de Exportar, Importar y Visualizar sus objetos a través de Direct3D desde DirectX 5. Cuenta con herramientas para permitir moverse entre SoftImage y los formatos estándar de archivos de la industria tanto para imágenes como para objetos, contiene un convertidor para Exportar e Importar desde o a 3D Studio. Acepta una gran variedad de formatos de imagen y render.

2.0.3 3D Studio Max

Es soportado por la más grande comunidad de desarrolladores que cualquier aplicación 3D, con la más amplia interacción con terceras aplicaciones integradas. Contiene un ambiente de desarrollo adecuado para la siguiente generación de video juegos con soporte Direct3D, multitexturizado por caras, mapeo de opacidad y transparencia real. Las gráficas para puertos interactivos soportan la aceleración de hardware a través de Direct3D y OpenGL o simulación por software para cualquier pantalla Windows.

En realidad existe una gran cantidad de software que puede asistirnos en la creación de los objetos tridimensionales, pero cabe señalar que el único requisito que deben cumplir los objetos 3D que deseen ser insertados en el ambiente gráfico deberán ser archivos con formato de DirectX (que son los archivos *.X). Para obtener este tipo de archivos es necesario obtener de alguno de los paquetes gráficos el archivo con formato *.3ds para que este archivo pueda ser transformado a través de algún convertidor (como son conv3ds.exe y XskinExp.dle específico para 3D Studio Max) al formato *.X . Así pues, no nos limitaremos en el desarrollo de la aplicación hacia el uso de objetos creados con un software gráfico específico; sin embargo, utilizaremos 3DS Max 4 para el diseño de los objetos simplemente por ser el software al cual se tuvo acceso y por ser una de las herramientas más ampliamente utilizadas en el área de desarrollo de video juegos y aplicaciones gráficas.

Realmente no se utilizarán todas las capacidades que ofrece este software, porque DirectX solamente acepta el objeto con las características que se mencionan en el apéndice 1.

2.1 ACTIVEX

ActiveX es el nombre que Microsoft ha dado a un grupo de tecnologías y herramientas "estratégicas" orientadas a objetos. Su principal tecnología es el Modelo de Objeto Componente (*Component Object Model*, COM). Al usarlo en una red con un directorio y apoyo adicional, el COM se convierte en el Modelo Distribuido de Objetos Componentes (*Distributed Component Object Model*, DCOM). El principal objeto que uno crea al escribir un programa ejecutable en el entorno ActiveX es un componente, un programa autosuficiente que puede ejecutarse en cualquier sitio en la red ActiveX (que es actualmente una red que consta de sistemas tanto Windows como Macintosh). Este componente se conoce como un Control ActiveX. ActiveX es la respuesta de Microsoft a la tecnología Java de Sun Microsystems. Un control ActiveX es aproximadamente el equivalente a un applet Java.

Si tiene un sistema operativo Windows en su ordenador, puede observar una cantidad de archivos de Windows con la extensión "OCX". OCX significa "Control de enlace e incrustación de objetos" (*Object Linking and Embedding control*). El Enlace e Incrustación de Objetos (*Object Linking and Embedding*, OLE) fue la tecnología de programación de Microsoft para soportar documentos compuestos como lo es el escritorio de Windows. El Modelo de Objeto Componente ahora incluye OLE como parte de un concepto más amplio. Ahora, Microsoft usa el término "control de ActiveX" en lugar de "OCX" para el objeto componente.

Una de las principales ventajas de un componente es que puede ser reutilizado por muchas aplicaciones (a las que se conoce como contenedores de componentes). Un objeto componente COM (control de ActiveX) puede crearse utilizando cualquiera de varios lenguajes o herramientas de desarrollo incluidos C++ y Visual Basic, o PowerBuilder, o con herramientas de creación de scripts como VBScript.

Los documentos compuestos son una de las aplicaciones importantes de los objetos distribuidos, donde cada aplicación produce un tipo de documentos dado en formato propio de esa aplicación (hoja de cálculo, gráfico, texto, etc.), un documento compuesto es producido en común por varias aplicaciones, encargada cada una de una parte del documento. En un documento compuesto se mezclan hojas de cálculo, texto, gráficos, páginas Web, y otros componentes independientes que comparten la superficie de la página e intercambian eventualmente datos entre sí. Los documentos OLE definen las interfaces necesarias para compartir la página y los canales de comunicación entre componentes. En el caso de OLE, estas interfaces se basan directamente en COM y OLE Automation.

2.1.1 El componente ActiveX

Las grandes categorías de eventos que Microsoft recomienda tener en consideración para definir los eventos característicos de los componentes ActiveX son las siguientes:

- Las peticiones
- Los eventos preoperativos (before events)

* Los eventos postoperativos (after events)
* Los eventos operativos (do events)

Las peticiones son eventos enviados por el componente ActiveX al documento para autorizarlo a proseguir o emprender una operación. Por ejemplo, un componente ActiveX que hubiera recibido un clic sobre el botón Cerrar, puede enviar una petición al documento para autorizar su cierre.

Los eventos preoperativos son enviados por el componente al documento al que pertenece antes de emprender una operación. Contrariamente a las peticiones, el contenedor no tiene, en este caso, el control sobre la autorización para ejecutar la operación. Los eventos postoperativos permiten al componente informar al documento al que pertenece sobre el fin de la ejecución de una operación en particular.

Los eventos operativos son el núcleo de compartir tareas entre el contenedor y el componente. La gestión de estos eventos puede ser implementada por el componente, por el contenedor o por ambos. El evento lo envía el componente al contenedor, el cual decide si responder o no. En el primer caso, el contenedor que responde envía el valor booleano falso al componente por el propio evento. En el segundo caso, señalado por el valor booleano de retorno verdadero, el componente es quien responde al evento. Este mecanismo permite pues compartir la carga de la gestión de eventos entre el componente y su contenedor. Obsérvese que esta comunicación puede complicarse si entran en juego varios niveles de contenedores/componentes. Los eventos de esta categoría son los eventos estándar definidos por Microsoft, como los click y los movimientos del mouse o las pulsaciones de teclas del teclado.

2.1.2 El objeto COM

COM es esencialmente un estándar que describe el formato binario de los archivos ejecutables. El problema que COM resuelve es el de la interacción entre programas ejecutables escritos en lenguajes de programación diferentes.

OLE utiliza la palabra interfase para designar la descripción de los objetos, y los programas clientes y los objetos sólo interactúan por medio de estas interfaces que definen en cierto modo un contrato entre proveedor y consumidor de servicios.

Para el programador, las interfaces se definen como clases virtuales en C++, en las que debe escribir los programas de todos los métodos. El programa cliente recibe, por su parte, un puntero hacia estas interfaces, que utiliza luego para llamar a los métodos del objeto, en general también en C++, como si el objeto perteneciera al propio programa cliente. El intercambio de mensajes entre el programa cliente y los objetos a través de estas interfaces lo realiza una biblioteca proporcionada por Microsoft que implementa el modelo COM para la plataforma Windows, o bien lo desarrolla el propio programador.

2.1.3 Interfaz COM en DirectX

Las interfaces de DirectX han sido creadas a un nivel muy básico de la jerarquía empleada en la programación COM. Cada interfaz de la API corresponde a un objeto que representa un dispositivo. La creación de estos objetos básicos es manipulada por funciones especializadas de la correspondiente DLL.

Típicamente, el modelo de objeto empleado por DirectX permite un objeto principal por cada dispositivo. El soporte para otros objetos es derivado del principal.

Además de la habilidad de crear objetos subordinados, el objeto principal determina las posibilidades del dispositivo hardware que representa, como puede ser la resolución de pantalla y el número de colores, o sí la tarjeta de sonido permite síntesis de tabla de ondas.

Para desarrollar ambientes gráficos lo primero que se necesita manejar es un lenguaje de programación y después nos podemos enfocar en el manejo de multimedia por medio de algún API; con esto tendremos lo suficiente para empezar nuestro desarrollo.

2.1.4 OLE Automation

Ole Automation es el procedimiento que permite a un programa cliente invocar dinámicamente los métodos de los objetos OLE. Construido encima de COM, OLE Automation permite a una aplicación controlar a otra manipulando los objetos COM que contiene. Diseñado inicialmente como medio de realizar un lenguaje de marcos común para todas sus aplicaciones, Microsoft ha enriquecido y publicado sus interfaces para permitir a terceras empresas desarrollar por sí mismas sus propios lenguajes de scripts.

El uso típico de OLE Automation se encuentra en Visual Basic de Microsoft, y su versión para la línea de productos Office, llamada Visual Basic for Aplications (VBA). VBA reemplaza en realidad todos los lenguajes de marcos propios de cada una de las herramientas que componen la suite Office. Común a todas, permite a cada una de estas aplicaciones intercambiar datos con otras y controlarlas a través de OLE Automation.

Una de las otras ventajas de OLE Automation es proporcionar las funciones elementales de una aplicación bajo una forma utilizable por otra aplicación, o por otro desarrollador, independientemente de su implementación. En OLE Automation es donde se encontrará también el soporte de los diferentes idiomas (National Language Suport o NLS) necesario para que, por ejemplo, la versión estadounidense de Visual Basic pueda interactuar con una versión española de Word. Estos mecanismo se extienden también al formato de fechas y a la representación de los signos decimales y la moneda.

En OLE se definen objetos autómatas (Automation Objects) o servidores autómatas que son controlados por un lenguaje de scripts externo o por invocaciones dinámicas. Las aplicaciones clientes se llaman también controladores de autómatas (Automation Controllers).

Actualmente, los controles de ActiveX corren en Windows 95 y NT, y en Macintosh. Microsoft planea soportar controles de ActiveX para UNIX.

2.2 PUERTO SERIE

2.2.1 Introducción a las comunicaciones serie

Las comunicaciones serie se utilizan para enviar datos a través de largas distancias, ya que las comunicaciones en paralelo exigen demasiado cableado para ser operativas. Los datos serie recibidos desde otros dispositivos son convertidos a paralelo gracias a lo cual pueden ser manejados por el bus del PC.

Los equipos de comunicaciones serie se pueden dividir entre simples, half duplex y full duplex. Una comunicación serie simple envía información en una sola dirección; half duplex significa que los datos pueden ser enviados en ambas direcciones entre dos sistemas, pero en una sola dirección al mismo tiempo. Una transmisión full duplex significa que cada sistema puede enviar y recibir datos al mismo tiempo.

Hay dos tipos de comunicaciones: Síncronas y Asíncronas. En una transmisión síncrona los datos son enviados en bloques, el transmisor y receptor son sincronizados por uno o más caracteres especiales llamados caracteres Sync.

El puerto serie del PC es un dispositivo asíncrono. En una transmisión asíncrona, un bit identifica su bit de comienzo y uno o dos bits identifican su final, no es necesario ningún carácter de sincronismo. Los bits de datos son enviados al receptor después del bit de inicio (start). El bit de menos peso es transmitido primero. Un carácter de datos suele consistir en siete u ocho bits. Dependiendo de la configuración de la transmisión un bit de paridad es enviado después de cada bit de datos. Se utiliza para corregir errores en los caracteres de datos. Finalmente uno o dos bits de parada (stop) son enviados.

Desde que el puerto serie de comunicaciones hizo su aparición, la evolución de éste no ha sido tan espectacular como la de otros dispositivos del ordenador. Las mejoras del puerto serie siempre han llegado en forma de velocidad aumentando desde los apenas 150 bits/seg hasta los 115 200 bits/seg que pueden tener hoy en dia.

Quien indica el comienzo de una nueva palabra de información es el bit de inicio o start. El bit de inicio o start es de vital importancia para sincronizar la transmisión y la recepción, y siempre será un cero lógico. El bit de paridad puede ser añadido a los bits de datos para permitir la detección de errores en la serie de bits de información. La paridad impar con lógica positiva, se activa cuando el número de unos es impar. La paridad par se activa cuando hay un número par de ceros o unos según la lógica.

El bit de parada o stop no indica el final de una palabra, aunque su nombre refleje lo contrario, su función es separar dos palabras consecutivas poniendo la linea de transmisión en

un estado inactivo el mínimo tiempo posible. Por lo que el bit de parada será siempre un uno lógico para que el bit de inicio siguiente tenga una probabilidad muy alta de ser detectado.

2.2.2 Fundamentos de la comunicación serie

Todos los equipos se suministran con uno o más puerto serie, que se denominan sucesivamente COM1, COM2, etc. En un equipo estándar, normalmente el *mouse* (ratón) estará conectado al puerto COM1. En el puerto COM2 puede haber conectado un modem, en COM3 un escáner, etc. Los puerto serie proporcionan un canal para la transmisión de datos desde estos dispositivos serie externos.

La función esencial del puerto serie es actuar como intérprete entre la CPU y el dispositivo serie. Al enviar datos desde la CPU a través del puerto serie, los valores de tipo **byte** se convierten en series de bits. Cuando se reciben datos, las series de bits se convierten en valores de tipo **byte**.

Para completar la transmisión de los datos es necesario otro nivel de interpretación. En el lado del sistema operativo, Windows utiliza un controlador de comunicaciones, Comm.drv, para enviar y recibir datos mediante las funciones estándar de la API de Windows. El fabricante del dispositivo serie proporciona un controlador que conecta este hardware con Windows. Cuando utiliza el control **Communications**, está ejecutando funciones de la API que interpreta el controlador Comm.drv y que se transfieren al controlador del dispositivo.

2.2.3 Códigos de protocolo de transmisión

En los códigos de protocolo de transmisión siempre se omite el bit de inicio porque en todos los casos es uno. Los que se usan más comúnmente son:

- 8n1(ocho bits de información, sin paridad y un bit de parada)
- 7e1(siete bits de información, paridad par y un bit de parada)
- 7e2 (siete bits de información, paridad par y dos bits de parada)

Los que se usan comúnmente son los dos primeros.

2.2.4 Configuración RS-232C

Para que los datos circulen de un ordenador a otro o de un dispositivo al PC, se crea una línea de transmisión de lazo cerrado con una corriente estática de 20mA y una corriente de caída de 0mA. Estos datos son muy importantes para saber si hay una correcta conexión entre ambos dispositivos, ya que si la conexión es defectuosa, los dispositivos detectarán que la corriente estática no es de 20mA y emitirá un error. Los voltajes que manejan las señales son de –3 a –15 voltios para el uno lógico mientras que para el cero lógico las tensiones son positivas. La impedancia de salida del puerto serie comúnmente es de 2KΩ(para cinco mA y 10V), mientras que la de entrada es de 4.3KΩ, consiguiendo un fan out de 5(se pueden conectar 5 entradas a una salida).

Para poder conectar cualquier dispositivo a este puerto hay que contar con terminales apropiadas, éstas son las terminales de 9 pines ó 25 pines.

Las líneas más importantes son las que hacen referencia a los datos, tanto la de transmisión como la de recepción y la de tierra. Las otras líneas son utilizadas por otros dispositivos, y sirve para indicar estados internos de estos dispositivos. Los cables que conectarán cada pin son únicos, a continuación se muestra una descripción de cada uno de ellos

Nombre V.24	25 pines	9 pines	Dirección	Nombre completo	Comentario
TxD	2	3	Salida	Datos transmitidos	Datos
RxD	3	2	Entrada	Datos recibidos	Datos
RTS	4	7	Salida	Permiso para transmitir	Comunicación
CTS	5	8	Entrada	Listo para transmitir	Comunicación
DTR	20	4	Salida	Terminal preparado	Estado
DSR	6	6	Entrada	Datos preparados	Estado
RI	22	9	Entrada	Detector de llamada	Estado
DCD	8	1	Entrada	Detector de portadora	Estado
GND	7	5	----------	Masa	Nivel de referencia
----------	1	----------	----------	Masa de protección	No es masa !!

TxD, RxD	Estas señales serán las encargadas de llevar a los datos de un extremo a otro, uno para transmitir y el otro para recibir.
RTS, CTS	Son usados por los dispositivos para seguir o cortar las transmisiones de datos.
DTR, DSR	Mediante estas dos señales, los dos dispositivos que estén conectados quedarán de acuerdo sobre la velocidad a usar, los bits de paridad, etc.
RI	Indican que hay una llamada entrante, muy importante en los módems.
DCD	También otra señal muy importante en los módems, ya que indica si hay portadora, y por tanto si se puede transmitir o no.
GND	Es la señal de referencia, tras consultarla se sabrá si cualquier señal recibida es de nivel alto o bajo.

2.2.5 Conectando dispositivos

Para unir cualquier equipo conectado de datos o DCE, como puede ser un módem por ejemplo, aun ordenador, se debe seguir el siguiente esquema de señales:

GND1	----------	GND2
RxD1	----------	RXd2
TxD1	----------	TxD2
DTR1	----------	DTR2
DSR1	----------	DRS2

RTS1	----------	RTS2
CTS1	----------	CTS2
RI1	----------	RI2
DCD1	----------	DCD2

Si lo que queremos conectar son dos equipos DTE, como pueden ser dos ordenadores, debemos hacer una configuración denominada null-modem:

GND1	----------	GND2
RxD1	----------	TxD2
TxD1	----------	RxD2
DTR1	----------	DSR2
DSR1	----------	DTR2
RTS1	----------	CTS2
CTS1	----------	RTS2

Si el software lo requiere se deberá hacer la configuración siguiente: conectar DCD1 a CTS1 y DCD2 a CTS". Como si de dos punteros se tratase.

Si el hardware, no lo necesita, se pueden obviar las líneas de estado. Y quedaría una configuración como esta:

GND1	----------	GND2
RxD1	----------	TxD2
TxD1	----------	RxD2

Si el software lo necesita podemos conectar adicionalmente:

RTS1	----------	CTS1 & CD1
RTS2	----------	CTS2 & DCD2
DTR1	----------	DSR1
DTR2	----------	DSR2

2.2.6 Asignación de memoria de buffer

En las propiedades **InBufferSize** y **OutBufferSize**, especifique la cantidad de memoria asignada a los búferes de recepción y transmisión. Los valores establecidos de manera predeterminada son los mostrados en la figura 2.2. Cuanto mayor sea el valor especificado, menos memoria habrá disponible para la aplicación. Sin embargo, si el buffer es demasiado pequeño, corre el riesgo de desbordarlo, a menos que utilice un protocolo.

Nota: Dada la cantidad de memoria disponible en la mayoría de los equipos en la actualidad, la ubicación de la memoria de búfer tiene menor importancia porque tiene disponibles más recursos. En otras palabras, puede establecer los valores de búfer más elevados sin que afecte el rendimiento de la aplicación.

2.2.6 Administrar los búferes de recepción y transmisión

Como se ha indicado anteriormente, los búferes de recepción y transmisión se crean siempre que se abre un puerto. Estos búferes se utilizan para almacenar los datos de entrada y para transmitir los datos de salida. El control **Communications** permite administrarlos a través de diversas propiedades con las que puede colocar y recuperar datos, obtener el tamaño de cada búfer y tratar datos de texto y binarios. La correcta administración de estos búferes es una parte importante del uso del control **Communications**.

2.3 TRANSFORMACIONES GEOMÉTRICAS

2.3.1 Traslación

Se aplica una traslación en un objeto para cambiar su posición a lo largo de la trayectoria de una línea recta de una dirección de coordenadas a otra. Convertimos un punto bidimensional al agregar las distancias de traslación, t_x y t_y a la posición de coordenadas original (x, y) para mover el punto a una nueva posición (x',y')

$$\left.\begin{aligned} x' &= x + t_x \\ y' &= y + t_y \end{aligned}\right\} \qquad (1)$$

El par de distancia de traslación (t_x t_y) se llama vector de traslación o vector de cambio.

Podemos expresar las ecuaciones de traslación (1) como una sola ecuación matricial al utilizar vectores de columna para representar las posiciones de coordenadas y el vector de traslación:

$$P = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad P' = \begin{bmatrix} x'_2 \\ x'_1 \end{bmatrix} \qquad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \qquad (2)$$

Esto nos permite expresar la dos ecuaciones de traslación bidimensional en la forma de matriz:

$$P' = P + T \qquad (3)$$

La traslación es una transformación de cuerpo rígido que mueve objetos sin deformarlos. Es decir, se traslada cada punto del objeto la misma distancia. Se traslada un segmento de línea recta al aplicar la ecuación de transformación (3) en cada uno de los extremos de la línea y se vuelve a trazar la línea entre las nuevas posiciones de los extremos.

Se utilizan métodos similares para trasladar objetos curvos. Para cambiar la posición de una circunferencia o de una elipse, trasladamos las coordenadas del centro y volvemos a trazar la figura en la nueva posición.

2.3.2 Rotación

Se aplica una rotación bidimensional en un objeto al cambiar su posición a lo largo de la trayectoria de una circunferencia en el plano xy. Para generar una rotación, especificamos un ángulo de rotación θ y la posición (x_r, y_r) del punto de rotación (o punto pivote) en torno al cual se gira el objeto. Los valores positivos para el ángulo de rotación definen rotaciones en sentido opuesto a las manecillas del reloj. También es posible describir esta transformación como una rotación sobre el eje de rotación que es perpendicular al plano de xy y pasa a través del punto pivote.

Primero determinamos las ecuaciones de transformación para la rotación de la posición de un punto P cuando el punto pivote esta en el origen de las coordenadas. Las relaciones angulares y de coordenadas de las posiciones de puntos originales y transformadas, r es la distancia constante del punto desde el origen, el ángulo ϕ es la posición angular original del punto desde el plano horizontal y θ es el ángulo de rotación. Al utilizar identidades trigonométricas estándar, podemos expresar las coordenadas transformadas en términos de los ángulos θ y ϕ como:

$$\left.\begin{aligned} x' &= r\cos(\theta + \phi) = r\cos\phi\cos\theta - r\,\mathrm{sen}\phi\,\mathrm{sen}\theta \\ y' &= r\,\mathrm{sen}(\theta + \phi) = r\cos\phi\,\mathrm{sen}\theta + r\,\mathrm{sen}\phi\cos\theta \end{aligned}\right\} \quad (4)$$

Las coordenadas originales del punto en las coordenadas polares son

$$x = r\cos\phi \quad , \quad y = r\,\mathrm{sen}\theta \quad (5)$$

Al sustituir las ecuaciones (5) en (4), obtenemos las ecuaciones de transformación para girar un punto en la posición(x,y) a través de un ángulo θ alrededor del origen:

$$\left.\begin{aligned} x' &= x\cos\theta - y\,\mathrm{sen}\theta \\ y' &= x\,\mathrm{se}\theta + y\cos\theta \end{aligned}\right\} \quad (6)$$

con las representaciones del vector de columna 2 para las posiciones de coordenadas, podemos expresar las ecuaciones de rotación en la forma de matriz:

$$P' = R \bullet P \quad (7)$$

Donde la matriz de rotación es:

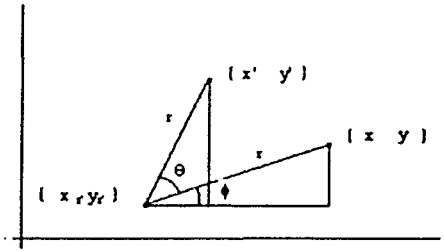
$$R = \begin{bmatrix} \cos\theta & -\text{sen}\,\theta \\ \text{sen}\,\theta & \cos\theta \end{bmatrix} \tag{8}$$

Cuando las posiciones de coordenadas se representa como vectores de renglón en vez de vectores de columna , el producto de la matriz en la ecuación de rotación (7) se transpone, de modo que el vector de coordenadas de renglón transformado [x' y'] se calcula como:

$$\left.\begin{aligned} P'^{T} &= (R \bullet P)^{T} \\ P'^{T} &= P^{T} \bullet R^{T} \end{aligned}\right\} \tag{9}$$

donde P^T = [xy] y se obtiene la transposición R^T de la matriz R con sólo cambiar el signo de los términos del seno.

En la figura (1) se ilustra la rotación de un punto alrededor de una posición pivote arbitraria.



***Figura(1)**: Rotación de un punto desde la posición (x,y) a la posición (x',y') a través de un ángulo θ con respecto del punto de rotación (x_r, y_r).*

Al utilizar las relaciones trigonométricas en esta figura, podemos generalizar las ecuaciones (6) para obtener las ecuaciones de transformación para la rotación de un punto con respecto de cualquier posición de rotación especifica (x_r y_r):

$$\left.\begin{aligned} x' &= x_r + (x - x_r)\cos\theta - (y - y_r)\,\text{sen}\,\theta \\ y' &= y_r + (x - x_r)\cos\theta - (y - y_r)\cos\theta \end{aligned}\right\} \tag{10}$$

Estas ecuaciones generales de rotación difieren de las ecuaciones de rotación (6) por la inclusión de los términos aditivos, así como los factores de multiplicación en los valores de las coordenadas.

Al igual que las traslaciones, las rotaciones son transformaciones de cuerpos rígidos que mueven los objetos sin deformarlos. Se giran a través del mismo ángulo todos los puntos de un objeto.

2.3.3 Escalación

Una transformación de escalación altera el tamaño de un objeto. Se puede realizar esta operación para polígonos al multiplicar los valores de coordenadas (x, y) de cada vértice por los factores de escalación s_x y s_y para producir las coordenadas transformadas (x', y'):

$$\left.\begin{aligned} x' &= x \bullet s_x \\ y' &= y \bullet s_y \end{aligned}\right\} \qquad (11)$$

El factor de escalación s_x escala objetos en la dirección de x, mientras que el factor de escalación s_y lo hace en la dirección de y. También se pueden expresar las ecuaciones de transformación (11) en la forma matricial:

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} X \\ y \end{bmatrix} \qquad (12)$$

ó

$$P' = S \bullet P \qquad (13)$$

Donde S es la matriz de escalación de 2 por 2 en la ecuación (12).

Se pueden asignar valores numéricos positivos cualesquiera a los factores de escalación s_x y s_y. Los valores menores que 1 reducen el tamaño de los objetos y los valores mayores que 1 producen una ampliación. Al especificar el valor de 1 tanto para s_x como para s_y no se altera el valor de los objetos. Cuando se asigna el mismo valor a s_x y s_y se genera una escalación uniforme que mantiene las proporciones relativas de los objetos. Cuando s_x y s_y tienen valores distintos, se obtiene una escalación diferencial que se emplea con frecuencia en aplicaciones de diseño, en que se crean imágenes a partir de unas cuantas formas básicas que se pueden ajustar por medio de transformaciones de escalación y colocación como se muestra en la figura (2).

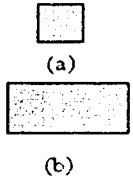


***Figura (2)**: Conversión de un cuadrado (a) en un rectángulo (b) con los factores de escalación $s_x = 2$ y $s_y = 1$.*

Los objetos que se transforman con la ecuación (12) se escalan y cambian de posición. Los factores de escalación con valores menores que 1 acercan los objetos al origen de las coordenadas, en tanto que los valores mayores que 1 alejan las posiciones de coordenadas del origen.

Podemos controlar la localización de un objeto escalado al seleccionar una posición, llamada punto fijo, que debe permanecer sin cambios después de la transformación de escalación. Se pueden seleccionar las coordenadas para el punto fijo (sf, sf) como uno de los vértices, el centroide del objeto, o cualquier otra posición figura(3).

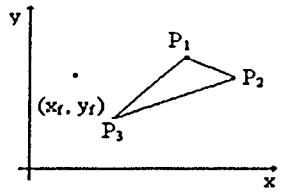


Figura (3): Escalación con respecto de un punto fijo seleccionado (x_f, y_f). Las distancias desde cada vértice del polígono al punto fijo se escalan mediante las ecuaciones de transformación (14).

Así se escala un polígono con respecto del punto fijo al escalar la distancia desde cada vértice al punto fijo. Para un vértice con coordenadas escaladas (x', y') como

$$\left.\begin{aligned} x' &= x_f + (x - x_f)\, s_x \\ y' &= y_f + (y - y_f)\, s_y \end{aligned}\right\} \qquad (14)$$

Podemos volver a expresar estas transformaciones de escalación para separar los términos de multiplicación y de adición:

$$\left.\begin{aligned} x' &= x.s_x + x_f(1 - s_x) \\ y' &= y.s_y + y_f(1 - s_y) \end{aligned}\right\} \qquad (15)$$

donde los términos aditivos $x_f(1 - s_x)$ y $y_f(1 - s_y)$ son constantes para todos los puntos en el objeto.

Se escalan polígonos al aplicar las transformaciones (15) en cada vértice y luego volver a generar el polígono utilizando los vértices transformados. Se escalan otros objetos al aplicar las ecuaciones de transformación de escalación a los parámetros que definen los objetos.

Es importante mencionar que la transformación geométrica de escalación no será implementada en el desarrollo del proyecto ya que no es necesaria para el funcionamiento de los movimientos del carro virtual. Pero como es parte de la teoría de transformaciones geométricas es bueno considerarla.

2.5 CONCEPTOS DE GRAFICACIÓN PARA DESARROLLO DE APLICACIONES

2.4.1 Arquitectura Raster System

Podemos simplificar la arquitectura de un sistema gráfico de barrido (raster system architecture) de la forma en que queda representada en la figura (4), donde podemos apreciar

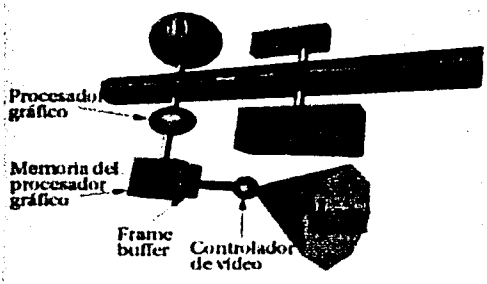


Figura (4) Sistema de barrido

que además del procesador de propósito general en el que se ejecutan todos los programas, a partir de la memoria principal del sistema, es necesario disponer de un procesador especializado en gráficos que realiza, más eficientemente, algunas funciones gráficas que el procesador principal no tendrá que realizar. De esta forma se dispone de dos procesadores, cada uno especializado en un objetivo específico y cada uno con su memoria (aunque el procesador gráfico puede tener acceso a la memoria del sistema).

El procesador gráfico va guardando la imagen que se mostrará en la pantalla en una memoria dedicada a tal efecto: el frame buffer. Al mismo tiempo el controlador de vídeo toma la información del frame buffer y la transpasa adecuadamente a la pantalla del monitor. La frecuencia con la que es posible realizar esta operación es la llamada **Frame Rate** .

Esto último es una de las características principales de estos sistemas, ya cuando deseamos tener animación las imágenes estáticas que dispuestas sucesivamente dan una sensación de movimiento siempre se sucederán como mucho tan deprisa como sea posible actualizar el frame buffer y poner la imagen en la pantalla.

2.4.2 Monitor con tubo de rayos catódicos

Además de la limitación del apartado anterior tenemos otra que podemos encontrar en los monitores de barrido como los que usan un tubo de rayos catódicos. Como podemos apreciar en la figura (5),un monitor convencional esta formado principalmente por una pantalla de vidrio recubierta interiormente formando pequeños grupos de tres compuestos

diferentes de fósforo que despedirán los colores rojo, verde o azul al ser excitados por los electrones provenientes de los cañones de electrones que se encuentran en el fondo del tubo de rayos catódicos. Pero estos cañones sólo pueden excitar a un grupo en cada instante, quedando excitado durante un tiempo hasta que de nuevo los cañones actúen sobre él. Por esta razón un haz de electrones debe barrer de forma continua la pantalla, de tal forma que todos sus grupos de flúor brillen con una intensidad uniforme. A la frecuencia con la cual es necesario realizar esta acción la llamaremos **Frecuencia de Refresco.**

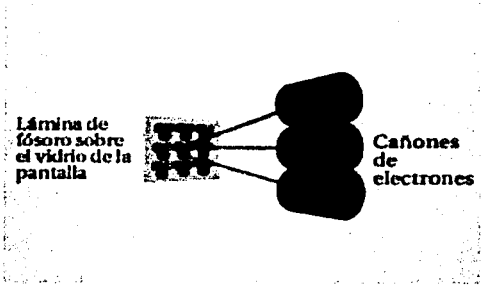


***Figura (5)**: Monitores de barrido como los que usan un tubo de rayos catódicos*

2.4.3 Pixel

Surge un nuevo concepto: ¿Qué es un píxel? No es cada uno de esos grupos de tres puntos de compuestos de flúor. El concepto de pixel depende de la resolución con la que estemos trabajando. **Resolución** denota al número de puntos de color distinguible que estemos representando en el monitor, aunque el monitor esté preparado para representar a más puntos distinguibles. Cada uno de estos puntos de color distinguible entre sí es un **Pixel.** En la figura (6) podemos apreciar que en la imagen de la cara que aparece en la pantalla del monitor, al ser ampliada con una lupa, se descubre que está formada por cuadraditos muy cercanos entre sí. Cada uno de estos cuadrados es un pixel, que mirados más de cerca parecen partidos en tres puntos de color rojo, verde y azul, de diferente intensidad que conforman al verse de lejos un color uniforme, el del pixel.

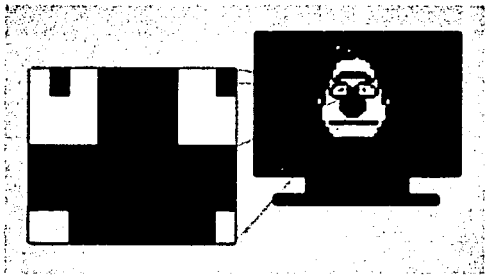


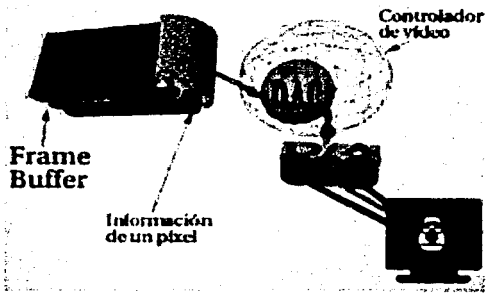
***Figura (6)**: Recordemos que nuestros pixeles dependen de la resolución que se tenga en la pantalla.*

2.4.4 El frame buffer

Una vez que nos hemos acercado al funcionamiento de un monitor, consideraremos de nuevo al frame buffer, de vital importancia a la hora de programar los gráficos.

¿Qué es el frame buffer? No es más que una memoria especial en la que se guarda el color de cada uno de los píxeles de la pantalla del monitor esto lo podemos apreciar en la figura(7).

¿Cómo se almacenan los colores? Cada color se guarda como una cadena de bits. Si cada píxel pudiera colorearse con uno de entre 256 colores, podríamos representar el color de cada píxel como tres bytes, el primer byte codificando 256 intensidades de rojo, el segundo a 256 intensidades de verde y el tercero con el mismo significado para el color azul. Si quisiéramos más colores tendríamos que utilizar más bits para cada uno de los colores básicos. En ocasiones, en el frame buffer no se guarda directamente el color del píxel, sino el número de la fila de una tabla en la que está guardado el valor para cada uno de los colores.



***Figura(7)** Representación del Frame Buffer*

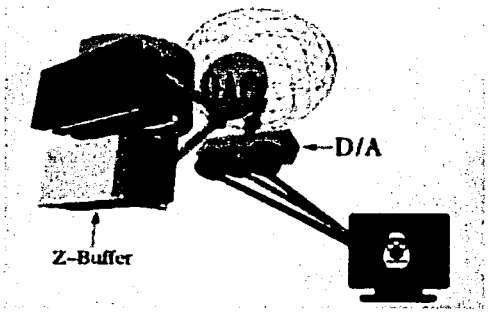
2.4.5 Doble buffer

Si el controlador de vídeo es tan rápido que es capaz de refrescar la pantalla con el contenido del frame buffer antes de que el procesador gráfico pueda actualizar del todo el frame buffer, nos encontraremos en la pantalla con imágenes a medio fabricar cuando queríamos que siempre se vieran imágenes completas. Para resolver este problema podemos dividir toda la información del frame buffer en dos, de forma que desde el punto de vista del programador disponemos de dos frame buffer, uno del que lee el controlador de video y otro en el que se va fabricando la nueva imagen. Al acabar de construir la nueva imagen se cambian los papeles.

2.4.6 Z-Buffer

Todo lo anterior es suficiente si siempre quisiéramos representar escenas bidimensionales. Pero si nuestra intención es poder visualizar una escena 3-D, necesitamos ordenar todos los objetos de forma que sepamos cuales tapan a otros. Esta ordenación puede realizarse por los algoritmos que conocemos, pero es más rápido que la realice nuestro hardware gráfico a consta de que el frame buffer use más memoria.

Ahora no sólo guardaremos el color de cada uno de los pixeles, sino que también almacenaremos la distancia a la cual se encuentra el punto del objeto representado por ese píxel, esto se aprecia en la figura(8).



Figura(8). Representación del Z-Buffer

2.4.7 Componente Alfa

Cuando manejamos situaciones 3-D, podemos no sólo querer tener objetos opacos, sino además objetos semitransparentes a través de los cuales sea posible ver a los objetos que estén detrás.

La transparencia es guardada como una componente más del color, llamada **Alfa**, que viene a engrosar la información guardada en el frame buffer, como podemos apreciar en la figura anterior.

3.0 REALIDAD VIRTUAL

A finales de los años ochenta, los gráficos por computadora entraron en una nueva época. No era sólo que las soluciones tridimensionales (3D) comenzaron a remplazar los enfoques bidimensionales y de dibujo de línea (2D), sino que también existía la necesidad de un espacio de trabajo totalmente interactivo generado a través de la tecnología.

La realidad virtual es un área que combina distintas tecnologías para visualizar, manipular e interactuar con una computadora en una forma que busca simular comportamientos y percepciones naturales para el hombre. La Realidad Virtual es la tecnología que permite sumergir a un usuario en un ambiente tridimensional simulado por el computador, de forma interactiva y autónoma en tiempo real.

En la actualidad, la realidad virtual toma múltiples matices, donde las formas de interacción y el grado de inmersión van variando. Sin embargo, aún cuando es un área que todavía es tema de importantes investigaciones, la riqueza proporcionada por su interfaz ha permitido su aplicación a múltiples problemas en distintas áreas de trabajo tales como la arquitectura, las artes, el diseño industrial, la medicina, entre otras. De esta forma, por medio de la realidad virtual se puede estar en lugares y situaciones simuladas, sin estar físicamente en ellos.

La Realidad Virtual es en tiempo real, interactiva e inmersiva. Trata de utilizar la mayor cantidad de sentidos para crear la sensación de inmersión. Los métodos inmersivos de realidad virtual con frecuencia se ligan a un ambiente tridimensional creado por computadora el cual se manipula a través de cascos, guantes u otros dispositivos que capturan la posición y rotación de diferentes partes del cuerpo humano. La realidad virtual no inmersiva utiliza medios como el que actualmente nos ofrece Internet en el cual podemos interactuar a tiempo real con diferentes personas en espacios y ambientes que en realidad no existen sin la necesidad de dispositivos adicionales a la computadora.

La realidad virtual no inmersiva ofrece un nuevo mundo a través de una ventana de escritorio. Este enfoque no inmersivo tiene varias ventajas sobre el enfoque inmersivo como: bajo costo, fácil y rápida aceptación de los usuarios. Los dispositivos inmersivos son de alto costo y generalmente el usuario prefiere manipular el ambiente virtual por medio de dispositivos familiares como son el teclado y el ratón que por medio de cascos pesados o guantes.

3.0.1 Características de Realidad Virtual

Se toman como características básicas de un sistema de realidad virtual las siguientes:

Interacción

Rasgos que permiten al usuario manipular el curso de la acción dentro de una aplicación de realidad virtual, permitiendo que el sistema responda a los estímulos de la

persona que lo utiliza; creando interdependencia entre ellos. Existen dos aspectos únicos de interacción en un mundo virtual. El primero de ellos es la navegación, que es la habilidad del usuario para moverse independientemente alrededor del mundo. Las restricciones para este aspecto las coloca el inventor del software, que permite varios grados de libertad, si se puede volar o no, caminar, nadar, etcétera.

El otro punto importante de la navegación es el posicionamiento del punto de vista del usuario. El usuario se puede mirar a sí mismo (a través de los ojos de alguien más), o puede moverse a través de cualquier aplicación observando desde varios puntos de vista.

El otro aspecto de la interacción es la dinámica del ambiente, que no es más que las reglas de cómo los componentes del mundo virtual interactúan con el usuario para intercambiar energía o información.

Inmersión

Esta palabra significa bloquear toda distracción y enfocarse selectivamente solo en la información u operación sobre la cual se trabaja. Posee dos atributos importantes, el primero de ellos es su habilidad para enfocar la atención del usuario, y el segundo es que convierte una base de datos en experiencias, estimulando de esta manera el sistema natural de aprendizaje humano (las experiencias personales).

Tridimensionalidad

Esta es una característica básica para cualquier sistema llamado de realidad virtual, tiene que ver directamente con la manipulación de los sentidos del usuario, principalmente la visión, para dar forma al espacio virtual; los componentes del mundo virtual se muestran al usuario en las tres dimensiones del mundo real, en el sentido del espacio que ocupan, y los sonidos tienen efectos estereofónicos (direccionalidad).

3.0.2 Software

El software de Realidad Virtual es el programa encargado en darle vida al mundo.... por medio de él, podemos dar movimiento a los objetos, hacer que el mundo aparezca ante nosotros desde una perspectiva diferente cuando miramos alrededor, simular situaciones tanto cotidianas como imposibles, y permitir que una persona interactúe con los elementos que le rodean.

Cada vez más cosas preprogramadas (grabadas) que saben cómo comportarse están siendo conectadas a las aplicaciones, reservando la programación detallada de instrucciones para propósitos de manipulación. Las técnicas de trabajo con objetos programados es denominada Programacion Orientada al Objeto, Software Orientado a los Objetos o Tecnología Orientada al Objeto.

Los creadores japoneses han combinado técnicas orientadas al objeto con el lenguaje de programación C, para llegar a una forma de escribir programas con una décima parte del código requerido previamente. El nuevo lenguaje es llamado C concurrente orientado al objeto

(COOC por Concurrent Object Oriented C). Este podría ser importante en aplicaciones largas y complejas como aquellas de los entornos virtuales.

Construir objetos para entornos virtuales no es todavía ni mucho menos trivial. Muchos asuntos permanecen sin resolver, incluyendo la incompatibilidad de muchos sistemas operativos y las formas en las que las partes de un programa se envían mensajes.

Cada módulo independiente debe contar con muchas reglas de comportamiento para lo que represente, y probar los contenidos de cada módulo es extremadamente difícil. Los atributos, como los dinámicos, no son fácilmente transferidos entre formatos de sistemas gráficos.

El software de la realidad virtual tendrá que ser independiente de los dispositivos, es decir, no limitado a cualquier tipo particular de computadora o sistema operativo. Sólo aquellos programas y configuraciones que puedan ser adaptados y modificados creativamente por usuarios y diseñadores son adecuados para el propósito de desarrollo de la realidad virtual.

3.0.3 Hardware

Para lograr la inmersión en un mundo de Realidad Virtual, es necesario el uso de dispositivos que simulen los estímulos sensoriales que recibimos a diario en nuestra "Realidad Real". Así, por medio de cascos o gafas estereoscópicas, podemos simular nuestra visión tridimensional, y veremos el mundo a nuestro alrededor; mediante el uso de guantes podemos coger y manipular los elementos del entorno, de una forma natural e intuitiva; y mediante el uso de otros dispositivos podemos oír y hasta sentir el entorno que nos rodea.

La Realidad Virtual busca modelar la Realidad misma; por ello, se puede aplicar en cualquier área del conocimiento como lo son:

Arquitectura	Educación	Medicina
Arte y Diseño	Electrónica	Mercadeo
Astronomía	Entrenamiento	Milicia
Ciencias Físicas y Químicas	Industria	Psicología
Deporte	Juegos	Simuladores
	Mecánica	Turismo

3.0.4 Tecnología que hace posible la realidad virtual.

La tecnología de la realidad virtual representa un avance decisivo en nuestra interacción con las computadoras y se han realizado asombrosos progresos en la creación de "mundos" cada vez más verosímiles generados por computación.

Para viajar a otra realidad basta colocarse un casco, un guante con una sensibilidad especial y estar conectado a un computador.

El casco llamado HDM (Head Mounted Display) tiene audífonos y dos monitores de cristal líquido (uno para cada ojo). El casco desarrolla la tecnología de las imágenes estereográficas y el rastreo de cabeza, que son las que permiten generar la sensación de estar inmerso en un mundo artificial de la realidad virtual.

A diferencia de los pescados, los humanos son seres binoculares: ven la misma imagen por los dos ojos (un pez ve una imagen por el ojo izquierdo y otra diferente por el derecho). Pero aunque el ojo izquierdo ve la misma imagen que el derecho la observa desde un ángulo ligeramente diferente. Cuando el cerebro compara estas dos imágenes similares las personas perciben la sensación de profundidad y distancia.

Los cascos de realidad virtual (Head Mounted Display) logran esa tridimensionalidad gracias a los dos lentes de cristal líquido. Y cada pantalla envía la misma imagen pero desde un ángulo diferente, tal como en el mundo real. Esas son las imágenes estereográficas.

El rastreo de cabeza sigue el movimiento de ésta, para que los programas de computador puedan colocar frente a los ojos del usuario las imágenes que corresponden, por ejemplo: el cielo cuando el usuario mueve la cabeza hacia arriba, el piso cuando la mueve hacia abajo, si se avanza la imagen aumenta de tamaño igual que si nos acercáramos a ella.

Cada fabricante de Head Mounted Display usa una tecnología distinta para el rastreo de cabeza: sensores inerciales, señales electromagnéticas o ultrasonido, entre otras. Estos dos efectos son complementados con sonido, ya que los cascos tienen audífonos que envían sonido digital de alta calidad a los oídos de la persona, con lo que aumenta el realismo.

Un sensor magnético en el casco transmite los cambios en la orientación de la cabeza al ordenador para que se recalculen instantáneamente desde el punto de vista que se presenta en las pantallas de dentro de las gafas. (Méndez, 1996)

Para aumentar el nivel de realismo se usan también los guantes de datos (Data Glove) que poseen sensores que rastrean el movimiento de la muñeca y de cada uno de los dedos (posición, orientación y curvatura de los ángulos de los dedos) permitiendo así, reproducir dentro del escenario virtual cada acción con la mano en el mundo real.

Este guante transmite información desde la mano del usuario al casco. En conjunto con los guantes y el Head Mounted Display, a través de los sensores, dan constantemente información de la posición del cuerpo. El guante de lycra instrumentado con fibras ópticas y sensores que miden la posición de la mano y el movimiento de los dedos, sustituye al ratón del ordenador personal.

Otros dispositivos aún más sofisticados son sillas y trajes. Las primeras sirven para complementar experiencias virtuales. Los trajes por su parte registran a través de múltiples sensores, todos los movimientos del cuerpo; de esa forma el nivel de interacción con el mundo virtual es total. (Morse, 1994)

En esta aplicación no usamos ninguna de las tecnologías anteriormente mencionadas sólo se utiliza el Joystick para el manejo del carrito y el teclado para la inserción del mismo en el ambiente que se desee manejar.

3.1 TIEMPO REAL

3.1.1 Concepto de Tiempo Real

Este concepto se puede aplicar a distintas ramas de la Informática y proviene de la ingeniería de control. En general la caracterización como 'tiempo real' se refiere a la existencia de determinadas restricciones sobre el comportamiento temporal de nuestro sistema. Dependiendo del tipo de aplicación esas restricciones serán de una clase u otra. El concepto de tiempo real, por tanto, puede aplicarse de forma más o menos estricta según lo duras que sean estas restricciones.

El **tiempo de respuesta (Tr)** se define como aquel periodo de tiempo que transcurre entre la entrada de un dato y la obtención de una salida (ver figura (9)). En un sistema de simulación visual representaría el tiempo que transcurre desde que el sujeto efectúa una acción (como mover el ratón) hasta que, después de efectuarse los cálculos correspondientes al modelo matemático del proceso y la visualización, se presenten las imágenes que tienen en cuenta su acción inicial. Una de las condiciones que podemos poner para hablar de tiempo real es limitar este parámetro a un valor máximo permitido (no queremos que haya un retraso mayor de un cierto intervalo), o incluso imponer que el tiempo de respuesta sea constante en todo momento.

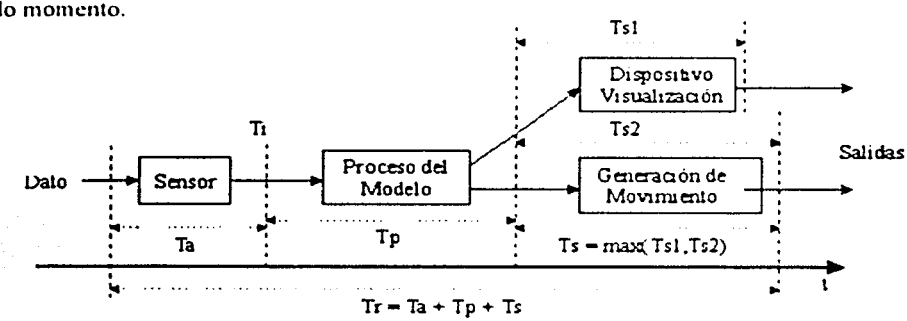


***Figura (9).**: Influencia de las diferentes partes del proceso de simulación en el tiempo de respuesta Tr.*

La restricción de este tiempo de respuesta (o 'retraso de transporte') está a la base del concepto de interactividad. En ciertas aplicaciones de simulación la sensación del sujeto de sentirse 'inmerso' en un escenario realista puede depender de este parámetro. Por ejemplo, en un sistema de realidad virtual en el que se usa un casco estereoscópico con localizador de posición, si el retraso observado entre el giro real de la cabeza del sujeto y los cambios

correspondientes en la imagen es mayor de una décima de segundo, resulta difícil mantener la ilusión de inmersión.

La **frecuencia de salida** es un parámetro diferente, aunque a veces relacionado, que depende del intervalo de respuesta entre dos salidas consecutivas del sistema (por ejemplo, si se presenta una nueva imagen cada 50 milisegundos, la frecuencia de refresco de los gráficos es de 20 Hertzios, veinte imágenes por segundo). Si el sistema no acepta una nueva entrada de datos hasta que no ha terminado de procesar la anterior, entonces la frecuencia de salida resulta ser precisamente el inverso del tiempo de respuesta. Pero también puede pasar que el sistema vaya procesando entradas de datos con una frecuencia diferente a la que se produce la salida. Por ejemplo, en un simulador de conducción puede leerse la posición del freno cada milisegundo (1000 hz.), pero producirse una salida del sistema de movimiento (plataforma móvil) 120 veces por segundo y una salida gráfica 30 veces por segundo (estos dos subsistemas tendrían diferentes tiempos de respuesta).

En el caso de la representación gráfica la frecuencia de salida constituye lo que habitualmente se conoce como **frecuencia de refresco** o ***frame rate***. En muchos simuladores es habitual especificar la restricción de que haya una frecuencia de refresco mínima, e incluso constante.

Además de las construcciones relativas al tiempo de respuesta y la frecuencia de salida, existen otras restricciones temporales independientes: la correspondencia entre el **tiempo aparente** de la simulación y el **tiempo físico** del observador (los objetos deben moverse en la simulación con la misma velocidad que lo harían en el mundo real, el tiempo interno utilizado por el modelo de la simulación debe tener la misma escala - o un factor constante y conocido - que el tiempo físico); y también es importante la **sincronización** entre diferentes salidas que corresponden al mismo fenómeno (por ejemplo, el sonido y los gráficos).

En los gráficos en tiempo real, el coste final de todas las operaciones depende fuertemente de su implementación, y fundamentalmente del hardware encargado de ejecutarlas. El hardware también puede ser importante al influir en el retraso de la transferencia de datos entre las diferentes fases del procesamiento, dependiendo de la arquitectura del sistema. Por ejemplo, por muy buena que sea una tarjeta gráfica para PC, puede estar limitada por la velocidad y el uso del bus del sistema, lo que le impedirá seguramente alcanzar frecuencias de refresco muy altas si en cada fotograma la CPU tiene que enviar todos los datos de la escena a la tarjeta.

Evidentemente, no todas las fases tardarán lo mismo, de modo que finalmente la frecuencia de salida vendrá determinada por la fase más lenta (menor frecuencia de procesamiento de datos o 'throughput'). Deberemos tener en cuenta, por consiguiente, que si las diferencias de duración entre fases son demasiado altas, la etapa más lenta condicionará completamente el funcionamiento de la demás, produciéndose el efecto de cuello de botella o 'bottleneck'.

3.1.2 Definición de un Sistema de Tiempo Real (STR)

STR: "Cualquier sistema que tiene que responder a estímulos generados externamente dentro de un plazo especificado y finito"

La correctitud de un STR depende:

* no sólo del *resultado lógico* de la computación (STC)
* sino también del *tiempo* en el que este resultado tarda en generarse.

Más que ser rápido, un STR debe ser *predecible*.

El STR forma parte de un sistema de ingeniería más amplio, bien un horno microondas, bien un sistema de guía de misiles, un automóvil o una central nuclear (**sistemas empotrados**)

1. Mediante un sensor, el computador conoce las variaciones de flujo
2. La respuesta en la válvula debe ser lo suficientemente rápida como para asegurar un flujo constante en la tubería

3.1.3 Técnicas de simulación en Tiempo Real

La principal característica de la simulación en tiempo real es que la aplicación informática debe mostrar las imágenes sintéticas a medida que se van produciendo, de forma que reflejen los cambios producidos por las acciones del usuario sobre el programa (concepto de *interactividad*). Estos cambios suelen responder a la representación de un fenómeno real que se intenta replicar en el tiempo (por ejemplo, el vuelo de un avión manejado por un piloto). El principal desafío desde el punto de vista gráfico es, por tanto, optimizar el costo de los cálculos necesarios para realizar la visualización. Como veremos porqué las técnicas utilizadas para conseguir la disminución del coste se basan normalmente en controlar el número de objetos a visualizar, el número de polígonos que los forman y el costo de rellenar estos polígonos.

3.1.4 Comprobación de visibilidad

En general un objeto puede no ser visible desde la posición del observador por dos motivos, bien porque se encuentre fuera del campo de visión, caso que es fácil comprobar sin un gran coste computacional; o bien porque, aún estando dentro del campo de visión, aparece totalmente ocluido por otro objeto. En este segundo caso la comprobación de la condición de visibilidad supone un mayor coste, pues habría que proyectar la silueta del objeto que puede ser ocluido hasta el punto de observación y ver si es completamente tapada o no por otros posibles objetos.

En cualquiera de estos dos casos podríamos llegar a determinar que un objeto no va a ser visible, dejando de enviar sus datos a la pipeline gráfica. Si el sistema de visualización recibiera los datos de estos objetos 'invisibles' no llegaría nunca a rellenar los pixeles, pero sí

necesitaría proyectar sus vértices para saber si caen fuera o dentro de la ventana (en la fase de recorte o *clipping*), y en el caso de un objeto ocluido (que sí cae dentro del campo de visión) también debería hacer la comparación de z-buffer con cada píxel cubierto para poder determinar su no-visibilidad. Por tanto, al dejar de enviar sus datos estamos evitando todas esas operaciones

3.2 ALGORITMOS PARA LA DETECCIÓN DE COLISIONES

La intención de este apartado es mostrar algunos de los algoritmos para detección de colisiones que han sido más utilizados en el desarrollo de aplicaciones gráficas tanto de entretenimiento, como los video juegos, así como de desarrollo científico y tecnológico como el CAD/CAM, robótica, arquitectura, etcétera. Primero se muestran los algoritmos más sencillos pero de menor precisión hasta llegar a un algoritmo que puede detectar con mucha precisión intersecciones entre objetos tridimensionales. Al final se describe el método elegido en nuestra aplicación.

La detección de colisiones ha sido un problema fundamental en la animación por computadora, modelado físico, modelado geométrico y robótica. En esas aplicaciones la interacción entre objetos que se mueven es modelada con restricciones dinámicas y análisis de contacto. El movimiento de los objetos está restringido por varias causas incluyendo colisiones.

Un ambiente virtual, como un lugar para caminar, crea un mundo generado computacionalmente, llenado con objetos virtuales. Ambientes como esos deben proporcionar al usuario la sensación de presencia lo cual incluye que las imágenes del usuario y de los objetos que le rodean parezcan sólidos. Así, los objetos no deben pasar unos a través de otros y las cosas se deben mover como se espera cuando se empujan o jalan. Tales acciones requieren de una detección de colisiones precisa si se quiere alcanzar un grado de realismo. No obstante, pueden existir cientos o miles de objetos en el mundo virtual y un mal algoritmo de detección de colisiones puede tomar mucho tiempo solo para probar las posibles colisiones cada vez que el usuario se mueve. Esto no es aceptable para ambientes virtuales donde el problema de la interactividad es una restricción fundamental del sistema.

El objetivo de la detección de colisiones tanto en ambientes virtuales como simulaciones es reportar contactos geométricos entre los objetos. Si sabemos la posición y orientación de los objetos en movimiento, se puede resolver la detección de colisiones como una función del tiempo. Sin embargo, este no es el caso en los ambientes virtuales o cualquier otra aplicación interactiva. De hecho, en un ambiente para caminantes, normalmente no se tiene información respecto a la velocidad máxima o aceleración, debido a que el usuario puede moverse con cambios abruptos tanto de dirección como de velocidad. Debido a esas variables no restringidas, la detección de colisiones es actualmente considerada uno de los mayores cuellos de botella en la construcción de ambientes interactivos simulados.

3.2.1 Método Bounding Box

Este es el método más rápido y menos preciso de detección, abstrayendo las formas a simples cajas. Supóngase que la geometría está encerrada por una caja. Se considerará que se ha tenido una colisión cuando su caja se traslapa con alguna otra caja definida por otra geometría. Se obtienen las coordenadas de la caja que encierra a los modelos, en caso de que las cajas estén traslapadas, la detección *Bounding Box* es verdadera. Una simple representación de este método la observamos en la figura (10)

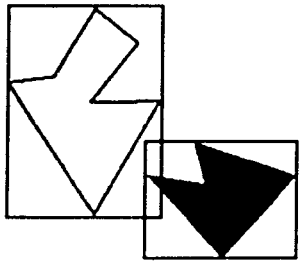


Figura (10) *El algoritmo de detección por cajas envolventes da verdadero a pesar de que las formas todavía no se tocan.*

Las condiciones para determinar la colisión son representadas en la ecuación(16):

$$\begin{aligned} X_{Amax} < X_{Bmin} \; Y_{Amax} < Y_{Bmin} \; Z_{Amax} < Z_{Bmin} \\ X_{Bmax} < X_{Amin} \; Y_{Bmax} < Y_{Amin} \; Z_{Bmax} < Z_{Amin} \end{aligned} \qquad \textbf{(16)}$$

Donde los valores corresponden a las coordenadas máximas y mínimas de las cajas envolventes. Se puede asegurar que no existe colisión si no más de una condición de las anteriores se cumple.

Una variante de este algoritmo consiste en cambiar las cajas por esferas y comprobar la colisión calculando la distancia entre sus centros y comparándola con la suma de sus radios. Esto lo podemos graficar en la siguiente figura (11):

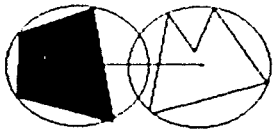


Figura (11) *El algoritmo de detección por círculos envolventes da verdadero cuando la distancia entre sus centros es menor o igual a la suma de sus radios*

3.2.2 Determinación rápida y mínima de la intersección de un triángulo y una raya

El algoritmo traslada el origen de la raya y cambia la base del vector de lo cual se obtiene un vector $(t\ u\ v)^T$, donde t es la distancia al plano en el cual se encuentra el triángulo y (u v) representan las coordenadas dentro del triángulo.

Una ventaja de este método es que no se necesita tener la ecuación del plano o calcularse en tiempo de ejecución o almacenarse, lo cual puede representar una gran ventaja en cuanto al ahorro de memoria para mallas de triángulos.

Una raya R(t) con origen O y dirección normalizada D está definida en la ecuación(17):

$$R(t) = O + tD \tag{17}$$

y un triángulo se define por tres vértices V_0, V_1 y V_2. En el problema de la intersección de una raya y un triángulo se quiere determinar si la raya intersecta al triángulo. En el algoritmo se construye una transformación y se aplica al origen de la raya. La transformación da un vector que contiene la distancia, t a la intersección y las coordenadas (u,v) de la intersección.

Un punto T(u,v) en un triángulo está dado por la ecuación(18):

$$T(u,v) = (1-u-v)V_0 + uV_1 + vV_2 \tag{18}$$

Donde (u,v) son las coordenadas baricéntricas las cuales deben cumplir que $u \geq 0$, $v \geq 0$ y $u+v \leq 1$. Calcular la intersección entre la raya R(t) y el triángulo T(u,v) es equivalente a hacer R(t) = T(u,v) lo cual da como resultado la ecuación(19):

$$O + tD = (1-u-v)V_0 + uV_1 + vV_2 \tag{19}$$

Rearreglando los términos de la ecuación(19) tenemos:

$$\begin{bmatrix} -D, V_1 - V_0, V_2 - V_0 \end{bmatrix} \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - V_0 \tag{20}$$

Esto significa que las coordenadas baricéntricas (u,v) y la distancia, t del origen de la raya al punto de intersección puede encontrarse resolviendo el sistema lineal de las ecuaciones (19) y (20).

Esto puede pensarse geométricamente como la traslación del triángulo al origen y transformarlo en un triángulo unitario en y y z con la dirección de la raya alineada con x como se ilustra en la figura(12): (donde $M = [-D, V_1-V_0, V_2-V_0]$)

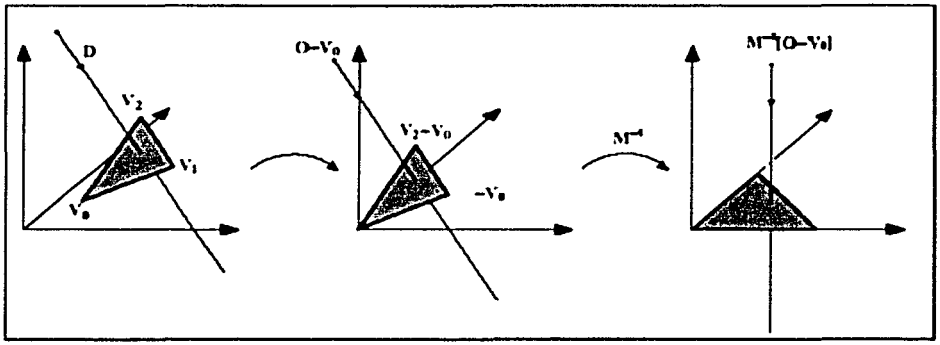


Figura (12) Traslación y cambio de base del origen de la raya.

Denotando E1 = V1-V0 , E2 = V2-V0 y T = O-V0 , la solución a la ecuación(20) se obtiene usando la regla de Cramer:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{-D, E_1, E_2} \begin{bmatrix} T, E_1, E_2 \\ -D, T, E_2 \\ -D, E_1, T \end{bmatrix} \qquad (21)$$

De álgebra lineal se sabe que | A, B , C | = -(A x C)*B = -(C x B)*A

Así la ecuación(21) se puede re escribir así:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{(D \times E_2) \cdot E_1} \begin{bmatrix} (T \times E_1) \cdot E_2 \\ (D \times E_2) \cdot T \\ (T \times E_1) \cdot D \end{bmatrix} = \frac{1}{P \cdot E_1} \begin{bmatrix} Q \cdot E_2 \\ P \cdot T \\ Q \cdot D \end{bmatrix} \qquad (22)$$

Donde P = (D x E2) y Q = T x E1

3.2.3 Detección de colisiones utilizando cajas envolventes orientadas.

Las jerarquías de volúmenes envolventes (BVH's) han sido utilizadas ampliamente en algoritmos de detección de colisiones. Los tipos de volúmenes utilizados con mayor frecuencia son las cajas envolventes alineadas con los ejes (AABB's) además de las esferas, las cuales son de representación simple, ocupan poco espacio de almacenamiento y son fáciles de implantar. Históricamente las cajas envolventes orientadas (OBB's) han sido menos utilizadas debido a que los métodos conocidos previamente para probar la intersección de las OBB's eran relativamente caros, no se conocían buenos métodos para la construcción automática de árboles de OBB's y los beneficios de la utilización de las cajas envolventes orientadas no habían sido realmente comprendidos. A continuación se mostrará un método

eficiente para construir árboles de OBB's y un nuevo método para probar la intersección de las OBB's el cual es más eficiente que los conocidos anteriormente.

Aplicaciones de la detección de colisiones

Definimos un objeto o un modelo como un subgrupo del espacio tridimensional. La forma de este subgrupo puede representarse por una colección de polígonos, por formas primitivas sólidas o por superficies curvas.

Una pregunta por colisión determina la intersección entre objetos dados y es utilizada en diseño asistido por computadora y manufacturación, sistemas de animación y modelado físico.

Una pregunta por distancia calcula la distancia entre dos objetos. Los sistemas de planeación de rutas pueden utilizar este tipo de preguntas los cuales automáticamente encuentran caminos viables para ensamblar y desensamblar objetos compuestos de múltiples partes o conducen robots a través de obstáculos en un ambiente que a veces requieren de detección de colisiones o cálculos de distancia como subrutinas de sus propios algoritmos de más alto nivel.

Para muchas de esas aplicaciones, las preguntas por colisión y por distancia son cuellos de botella computacionales. Por ejemplo las aplicaciones que utilizan dispositivos con force-feedback necesitan calcular todos los contactos con los modelos tridimensionales complejos en menos de un milisegundo. La reducción del tiempo de ejecución de esas preguntas se alcanzará un mejor desempeño y una mejor utilización de las herramientas que las utilizan.

Las preguntas por colisión y distancia son miembros de una clase más general conocida como preguntas de proximidad las cuales aportan información dejando de lado el lugar relativo de los modelos. Otros ejemplos de preguntas de proximidad son el ancho de distancia (la distancia entre los puntos más separados entre los modelos), la distancia de penetración (la traslación más pequeña que se necesita para separar dos modelos) y la distancia de Hausdorf (la mayor distancia de cualquiera de los puntos de un modelo de todos los puntos del otro modelo).

Además existen preguntas aplicables a escenarios dinámicos como la búsqueda del momento en que el siguiente contacto ocurrirá entre dos objetos que se están moviendo. Algunas aplicaciones pueden necesitar una extensión de esas preguntas las cuales trabajan con colecciones de modelos más que con sólo pares de ellos.

Existe una gran variedad de preguntas con diferentes usos para diferentes aplicaciones además de existir una gran cantidad de algoritmos para implantarlos. En este caso presentaremos la pregunta por colisión para un par de modelos poligonales.

Representaciones de modelos

Los modelos geométricos utilizados en gráficos computacionales, CAD/CAM y robótica pueden ser divididos en dos categorías: poligonales y no poligonales. Las representaciones poligonales son colecciones de polígonos las cuales pueden ser desordenadas, arregladas en matrices o formados por formas convexas. Las representaciones no poligonales incluyen superficies paramétricas, superficies implícitas y geometría constructiva de sólidos (CSG).

Una sopa de polígonos son colecciones arbitrarias de polígonos en el espacio tridimensional no necesariamente conectados. A diferencia de las representaciones paramétricas e implícitas, los polígonos no pueden representar superficies curvas exactamente. En su lugar las superficies lisas curveadas son aproximadas cuadriculando la superficie a una determinada precisión. Con esta limitante, los polígonos puedes aproximar cualquiera de las formas alcanzables por las superficies paramétricas o implícitas. Existe hardware que soporta el procesamiento de modelos poligonales en muchos de los sistemas computacionales disponibles lo cual hace que la representación poligonal sea una alternativa verdaderamente atractiva para la visualización de aplicaciones. De esta forma en nuestro caso nos enfocaremos en los modelos geométricos representados por polígonos.

Jerarquías de volúmenes envolventes

Las jerarquías de volúmenes envolventes (BVH's) son una de las más simples y ampliamente difundidas estructuras de datos para implantar la detección de colisiones en modelos complejos. Los métodos basados en BVH's no necesitan contar con ninguna propiedad topológica de los modelos y consecuentemente son aplicables a la sopa de polígonos.

Una jerarquía de volúmenes envolventes es un árbol de volúmenes envolventes como pueden ser esferas o cajas cuya colección de nodos hoja encierran espacialmente todo el modelo geométrico y en el cual cada padre encierra espacialmente toda la geometría cubierta por sus nodos hoja descendentes. Generalmente, cada volumen circundante en la jerarquía está hecho lo más pequeño posible en términos de volumen, área superficial, diámetro o alguna otra propiedad de medida de tamaño, mientras siga cubriendo su geometría inherente. Una muestra de esta la podemos observar en la siguiente figura(13):

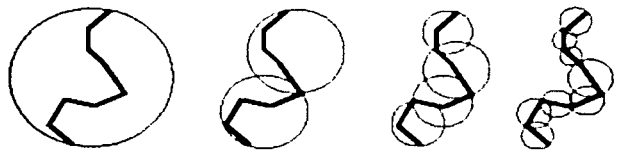


***Figura (13)** La geometría de ejemplo es una colección de segmentos de línea. La esfera de mayor nivel cubre toda la geometría. En el siguiente nivel, dos esferas cubren cada una la mitad de los segmentos. En cada etapa, las primitivas cubiertas por cada volumen envolvente también están cubiertas por la unión de los volúmenes envolventes que son hijos del volumen envolvente en cuestión. En los nodos de menor jerarquía los volúmenes envolventes cubres sólo una primitiva. Los volúmenes hijo pueden traspasar el volumen que encierra el padre y entre nodos hermanos pueden traslaparse*

Otra clase de estructuras de datos espaciales usadas para detección de colisiones son subdivisiones espaciales. Las subdivisiones espaciales son una partición recursiva del espacio cubierto, mientras que la jerarquía de volúmenes envolventes está basada en una partición recursiva de las primitivas de un objeto. Ejemplos de subdivisión espacial son octrees, árboles de partición espacial binaria (árboles BSP) y árboles k-d. En la subdivisión espacial, las regiones hermanas nunca se traslapan, nunca se extienden más allá del espacio del padre lo cual las distingue de la jerarquía de volúmenes espaciales. En nuestro caso nos enfocaremos en la división por volúmenes envolventes.

Una pregunta por colisión basada en BVH procede recursivamente probando volúmenes envolventes de los modelos por intersectarse. Por cada prueba, si se traslapan, entonces los hijos del volumen circundante se prueban. Si no se traslapan, entonces el ciclo recursivo termina. Si dos nodos hoja se intersectan entonces los polígonos que ellos encierran son probados y el resultado de esa prueba se agrega a la respuesta de la pregunta.

Existe una gran cantidad de variaciones a este algoritmo. Por ejemplo, pueden utilizarse diferentes tipos de volúmenes envolventes además de las esferas como son las cajas envolventes alineadas a los ejes (AABB's), cajas envolventes orientadas arbitrariamente (OBB's), elipsoides, cubiertas convexas, cilindros, formas esféricas. De hecho, cualquier forma puede usarse que pueda proveer dos operaciones. La primera es cómo encerrar la forma de una colección de polígonos y la segunda, cómo probar el traslape de dos formas. En general las formas más complejas permiten encerrar de mejor manera a la geometría permitiendo que la pregunta se complete utilizando menos pruebas de traslape, sin embargo, cada prueba se hace más larga. El problema se encuentra en elegir la mejor forma que proporcione el menor número de pruebas y sea más rápida de probar. Una representación de ello se muestra en la figura(14).

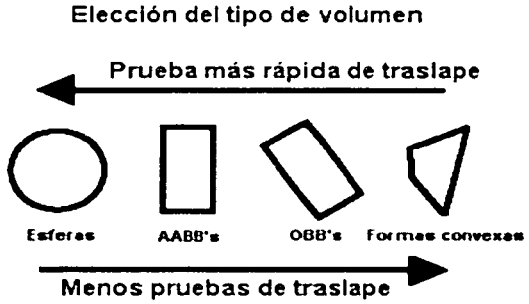


Figura (14): Figuras de envolventes

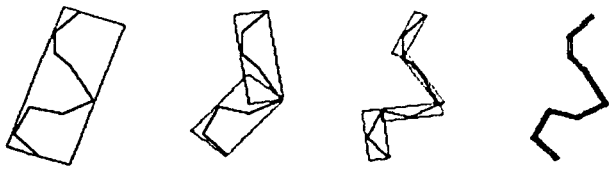
Otra elección del algoritmo es cómo determinar la secuencia de pruebas del traslape. Uno puede proceder con recursión por profundidad o puede proceder en recursión por anchura en el árbol. También se puede pedir que cuando se encuentre traslape en dos volúmenes se comparen los volúmenes hijos de los dos o uno de los padres con los hijos de acuerdo a algún

criterio como puede ser su volumen , área superficial o diámetro. Esas elecciones pueden ser consideradas reglas. Diferentes reglas pueden tener diferentes consecuencias en el desempeño.

Además otra elección del algoritmo se encuentra en cómo construir la jerarquía de volúmenes envolventes, top-down, botton-up, o inserción incremental. Un método top-down ajusta un volumen al modelo completo, parte las primitivas en dos grupos y recursivamente aplica la operación a cada grupo. Un método botton-up ajusta volúmenes envolventes a cada primitiva individualmente y entonces iterativamente combina combina grupos para formar grupos más grandes, ajustando un nuevo volumen circundante a cada nuevo grupo cuando se forma. Una técnica de inserción incremental construye un árbol insertando primitivas en la estructura del árbol una primitiva a la vez, ajustando la jerarquía de volumen circundante con cada inserción. Cada una de esas aproximaciones tiene muchas variaciones. Todavía no se sabe cuál método produce árboles que permiten la detección de colisiones más eficiente.

Árboles de cajas envolventes orientadas (OBBTrees)

Una caja circundante orientada (OBB) es un rectanguloide orientado arbitrariamente. Una caja circundante alineada a los ejes (AABB) es un rectanguloide cuyas caras están alineadas con los ejes coordenados de su sistema de coordenadas. Mientras que una AABB puede representarse con sólo puntos máximos y mínimos a lo largo de cada eje, una representación de OBB debe codificar no sólo la posición y anchos también la orientación. La ventaja que tienen las OBB's sobre las AABB's como volúmenes envolventes es que pueden encerrar con mayor precisión la geometría. Un esquema bidimensional de la jerarquía de las OBB's la podemos ver en la figura(15):



***Figura(15):** Jerarquización de un conjunto de líneas utilizando cajas envolventes orientadas.*

Costo de pregunta por colisión

Las operaciones fundamentales de una pregunta por colisión son la prueba de traslape de los volúmenes envolventes y la prueba de traslape de primitivas. El tiempo requerido para ejecutar una pregunta por colisión puede ser aproximado a:

$$T = N_v T_v + N_p T_p \qquad (23)$$

donde N^V y N^P son el número de pruebas por traslape para volúmenes envolventes y primitivas respectivamente y T^V y T^P son el tiempo promedio requerido para realizar cada una

de las pruebas. Pruebas utilizando volúmenes simples como esferas y AABB's exhiben una Nv mayor y un Tv menor, mientras que aquellos que utilizan volúmenes más complejos tienen una Nv menor y un Tv mayor. De esta forma no existe una mejor elección porque depende de la aplicación que se está tratando la elección del mejor volumen a utilizar.

Muchos factores contribuyen al costo de una pregunta por colisión. Los modelos con muchos polígonos tienden a ser más costosos para preguntar que los modelos con menos polígonos. Los modelos con formas geométricas más complejas (como puede ser un árbol) tienden a ser más caros que aquellos con una forma simple (como una pelota), incluso si la cantidad de polígonos es la misma. Las preguntas por colisión son más caras cuando los modelos están más cercanos que cuando están alejados. Además, las preguntas por colisión entre modelos bien separados puede resolverse en un número constante de pruebas de volúmenes envolventes, sin importar el número de polígonos . Finalmente, las preguntas por colisión que producen una lista de todos los polígonos que se tocan tenderán a ser más caras conforme el número de contactos se incrementa.

Así, el costo de una pregunta por colisión no sólo depende del tamaño de la entrada (el número de polígonos en los dos modelos) y el tamaño de la salida (el número encontrado de pares de primitivas que se estén tocando) sino también de la naturaleza y grado de proximidad de los modelos. Sin embargo la proximidad no es fácilmente cuantificable dado que debe capturar la complejidad de la forma de los modelos como esas formas interactúan espacialmente.

Implantación del algoritmo de detección de colisiones

De esta forma, después de haber investigado entre éstos y otros algoritmos para la detección de colisiones, los cuales no fueron especificados por tratarse de algoritmos que utilizan modelos no poligonales como son superficies paramétricas, llegamos a la conclusión que por las características de nuestros ambientes podemos utilizar una implantación jerárquica de los algoritmos de cajas envolventes no jerárquicas e intersección de rayas con triángulos recursivamente aprovechando las funciones que proporciona DirectX. No se utiliza plenamente el último algoritmo por requerir de un equipo de muy alto desempeño y procesamiento pipeline que no soportan la mayoría de las computadoras personales, así como la necesidad de un cálculo de un árbol jerárquico de volúmenes que requiere de una gran cantidad de tiempo en una máquina común; sin embargo, se tomaron ideas para combinar los dos primeros algoritmos.

Debido a que nuestro ambiente se encuentra postrado en un solo plano (el piso) y el único objeto que se está moviendo es el robot (carrito), entonces podemos implantar un algoritmo de detección de colisiones por círculos el cuál pasará al siguiente nivel de prueba de colisión si los círculos se traslapan.

El siguiente nivel de detección de colisiones aprovecha las utilidades desarrolladas por la librería D3DX8 de DirectX la cual proporciona la prueba de cajas envolventes. Esta función determina si una raya (ray) interseca a la caja envolvente de un modelo. Así se especifican las rayas de un modelo y se comparan con la caja envolvente del otro modelo a través de la función que proporciona DirectX.

Debido a que este algoritmo no permite una detección precisa de la colisión con objetos de formas complejas, nos vimos en la necesidad de implantar un método alternativo para el usuario el cual sustituye la detección por cajas y prueba la malla contra una raya. Así, nos apoyamos en otra función de DirectX que necesitas como parámetros esenciales una malla y una raya y determina si la raya intersecta a la malla. El método preciso toma cada una de las rayas que conforman la malla de un objeto y las compara con la malla del otro objeto.

Después de haber implantado los dos algoritmos pudimos observar que el primer algoritmo es realmente eficiente cuando en el ambiente no se encuentran objetos irregulares o con espacios debajo de ellos como puede ser una mesa sencilla debido a que el espacio que se encuentra debajo de la mesa es parte de la caja envolvente del objeto. El segundo algoritmo disminuye en gran medida la velocidad con que se puede mover el carrito ya que toma mucho más tiempo de procesamiento el método de detección de colisiones exacto.

Cabe mencionar que dentro de los ambientes que se pretenden manejar con esta aplicación únicamente se considera al carrito como el objeto en movimiento, es decir, se considera al ambiente estático. Por otra parte, los objetos que se encuentran en el ambiente están al piso y se espera sean de formas simples (cajas, esferas, cubos, etcétera) por lo que se considera el primer algoritmo como suficiente para satisfacer las necesidades de la aplicación. Se deja como opción el algoritmo preciso para que el usuario pueda interactuar con los espacios no accesibles que considera el primer algoritmo.

4.0 EXPLICACIÓN DEL DESARROLLO DEL SOFTWARE

Esta parte está desarrollada con la finalidad de describir las funciones más importantes de la aplicación que se diseñó con el apoyo del marco teórico ya expuesto; principalmente se mostrarán las funciones codificadas que tienen su fundamento en la explicación teórica o que fueron determinantes para el desarrollo de la aplicación.

4.0.1 Obtención, Configuración y Dibujado(Renderización) del dispositivo DirectGraphics

Para desarrollar la aplicación existen dos formas en pantalla completa o modo ventana; se eligió esta última debido a que era necesario interactuar con la aplicación a través de botones y cajas de texto además de facilitar la configuración del dispositivo adaptándose al modo actual de video del sistema que lo esté ejecutando. De esta forma al iniciar la aplicación una de las primeras cosas que se crean es el dispositivo gráfico sobre el que se va a trabajar. Las funciones que se encargan de inicializar el dispositivo de video son **Inicializacion, Obtener_Disp.** Inicialización obtiene información del estado actual del monitor y verifica la capacidad del acelerador gráfico para la graficación de la profundidad(Z- Buffer), además de inicializar las variables necesarias para la ejecución del programa. El siguiente código es de la función **Inicializacion:**

```
'El siguiente procedimiento inicializara todo el proceso, Regresa True si se logro el éxito, False en caso contrario

Public Function Inicializacion() As Boolean
On Error GoTo 0
On Error GoTo ErrHandler:
Set Dx = Nothing
Set Dx = New DirectX8
Set D3DX = Nothing
Set D3DX = New D3DX8
Set D3D = Nothing
Set D3D = Dx.Direct3DCreate() 'A través del objeto maestro se crea la interfaz Direct3D
Dim DispMode As D3DDISPLAYMODE 'Describe nuestro modo de despliegue
Dim i As Integer 'Variable auxiliar
D3D.GetAdapterDisplayMode D3DADAPTER_DEFAULT, DispMode
D3DWindow.Windowed = 1 'Tipo de pantalla
D3DWindow.SwapEffect = D3DSWAPEFFECT_DISCARD 'Tipo de sincronización con la etapa de refresco del monitor
D3DWindow.BackBufferCount = 1 'Numero de buffers para mostrar los frames
D3DWindow.BackBufferFormat = DispMode.Format

If D3D.CheckDeviceFormat(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, DispMode.Format,
D3DUSAGE_DEPTHSTENCIL, D3DRTYPE_SURFACE, D3DFMT_D16) = D3D_OK Then
'verifica que se pueda utilizar el tipo de dispositivo elegido
'Se puede utilizar Z-Buffer de 16 Bits
   D3DWindow.EnableAutoDepthStencil = 1
   D3DWindow.AutoDepthStencilFormat = D3DFMT_D16
End If

'A continuación se crea un dispositivo que utiliza capa de abstraccion hardware si es posible, procesamiento de vértices por
software y utiliza picture1 como targeta de dibujo y Obbene Un dispositivo permitido
```

```
Obtener_Disp Picture1.hWnd, D3DWindow, DispMode, Texture, D3Ddevice
'Iniciar variables del entorno
JoyStick1.Interval = 0 'Deshabilita el Joystick
Derecha = False 'Inicia las variables de teclado a falso
Izquierda = False
Arriba = False
Abajo = False
Control = False
Mayus = False
margen.Checked = True 'Inicia el tipo de algoritmo para la detección de colisiones
exacto.Checked = False
Piso.Checked = True
Cuadros.Checked = False

'Inicia el alejamiento

Alejamiento_x = 83
Alejamiento_z = 83
Levantamiento_y = 35
Text4.Text =
Alejamiento_x
Text5.Text = Levantamiento_y
With UpDown4
   .Max = 500
   .min = -500
   .Value = Alejamiento_x
   .Increment = 1
End With
With UpDown5
   .Max = 500
   .min = 0
   .Value = Levantamiento_y
   .Increment = 1
End With
inicia_plano D3DDevice

'Carga la textura para el plano

Set Texture = D3DX.CreateTextureFromFileEx(D3DDevice, App.Path & "\texture.bmp", 256, 256, D3DX_DEFAULT, 0,
DispMode.Format, D3DPOOL_MANAGED, D3DX_FILTER_LINEAR, D3DX_FILTER_LINEAR, 0, ByVal 0, ByVal 0)

'Inicialización los valores de las barras

VScroll1.Max = plano(0).z ' valores que responden a las dimensiones del plano para saber
VScroll1.min = plano(2).z ' cuanto se recorreran las barras
HScroll1.Max = plano(1).X
HScroll1.min = plano(0).X
ancho_original = Form1.Width ' guarda el ancho y el alto de la forma
alto_original = Form1.Height
Picture1.ScaleHeight = Picture1.Height ' para que nos regrese escalas funcionales del picture
Picture1.ScaleWidth = Picture1.Width

'Inicializar Ambiente

    Iniciar_Vista D3DDevice, vector_camara, vector_punto, False, 5000
```

```
    Iniciar_cuadricula 30
    Nuevo_Click
    UpDown3.Max = plano(0).z / Tan(pi / 8)
    UpDown3.min = (Objetos(0).esquina_sup_der_max.z - Objetos(0).esquina_inf_izq_min.z) / Tan(pi / 8)
    UpDown3.Value = CInt((UpDown3.Max - UpDown3.min) / 2)
Joystick.Checked = False
Command7.Enabled = False
Teclado.Checked = True

'Iniciar area de datos

With MSFlexGrid1
.Cols = 7
.Rows = 5
.Width = 0.97 * Frame3.Width
.Height = 0.8 * Frame3.Height
.ColWidth(0) = Frame3.Width / 3
.ColWidth(1) = Frame3.Width / 4
For i = 2 To .Cols - 1
   .ColWidth(i) = (.Width - .ColWidth(0) - .ColWidth(1)) / (.Cols - 1)
Next
.col = 0
.Row = 0
.Text = "Coordenadas virtuales(x,y)"
.Row = 1
.Text = "Coordenadas reales(x,y)"
.Row = 2
.Text = "Dirección virtual(x,y)"
.Row = 3
.Text = "Dirección real(x,y)"
.Row = 4
.Text = "Sensores"
For i = 1 To .Cols - 1
   .col = i
   .Text = i
Next
End With

'Iniciar reloj

Timer1.Enabled = False
Horas = 0: Minutos = 0: Segundos = 0
Hora.Text = CStr(Horas) & ":" & CStr(Minutos) & ":" & CStr(Segundos)
Inicializacion = True
Exit Function 'La inicialización se logró con éxito
ErrHandler:  'Manejo de errores
   MsgBox "Hubo un error de inicialización número :" & Err.Number
   Inicializacion = False
   End
On Error GoTo 0
End Function
```

La función **Obtener_Disp** se encarga de crear y asociar un dispositivo al picture1 además de determinar el tipo de texturizado que se utiliza, así como el tipo de luz y el modo de procesamiento de vértices.

```
Public Function Obtener_Disp(hWnd As Long, D3DWindow As D3DPRESENT_PARAMETERS, DispMode As
D3DDISPLAYMODE, ByRef Texture As Direct3DTexture8, ByRef D3DDevice As Direct3DDevice8)
Set D3DDevice = D3D.CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hWnd,
D3DCREATE_SOFTWARE_VERTEXPROCESSING, D3DWindow)
If Err.Number <> 0 Then 'Entonces maneja el error
   MsgBox " Hubo un error de inicialización de dispositivo"
   Set D3DDevice = Nothing
   On Error GoTo 0
   Unload Me 'En caso de error limpia la variable y termina
   Exit Function
End If

'Habilita la luz ambiental

D3DDevice.SetVertexShader Unlit_FVF 'Define el tipo de vértice a utilizar en el render
D3DDevice.SetRenderState D3DRS_LIGHTING, 1 'Habilita las luces
D3DDevice.SetRenderState D3DRS_AMBIENT, &HFFFFFF 'Establece una luz ambiental blanca
D3DDevice.SetRenderState D3DRS_CULLMODE, D3DCULL_NONE

'Habilita el buffer z

D3DDevice.SetRenderState D3DRS_ZENABLE, 1

'Iluminación

   With Ligth.diffuse
      .a = 1: .r = 1: .g = 1: .b = 1
   End With
   With Ligth.Ambient
      .a = 1: .r = 1: .g = 1: .b = 1
   End With
   D3DXVec3Normalize Ligth.Direction, MakeVector(0, -1, 0)
   Ligth.Type = D3DLIGHT_DIRECTIONAL
End Function
```

Después que se obtiene un dispositivo y se ha configurado, es necesario actualizar constantemente el dispositivo de salida. La función **Render** se encarga de proporcionar esas actualizaciones llenando los buffers de intercambio, además se encuentra dentro de un ciclo cerrado para mantener actualizada la pantalla. El código de esta función se muestra a continuación:

```
Public Sub Render()

'Variables requendas
   Dim i As Long, j As Long  'varaible de bucle
   Dim Str As String '

'Limpiar el dispositivo de salida antes de dibujar cualquier cosa

   D3DDevice.Clear 0, ByVal 0, D3DCLEAR_TARGET Or D3DCLEAR_ZBUFFER, &H0, 1#, 0

'Interpretar los gráficos

   D3DDevice.SetRenderState D3DRS_FILLMODE, D3DFILL_SOLID
```

```
    D3DDevice.BeginScene 'En adelante comienza a llenarse la escena

'//Dibuja el plano

        If Piso.Checked = True Then
            D3DDevice.SetMaterial material 'Establece el material para mostrar el plano
            D3DDevice.SetTexture 0, Texture 'Establece la textura activa
            D3DDevice.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, plano(0), Len(plano(0))    'Se agrega el plano
        Else
            D3DDevice.SetMaterial material_cuadricula 'Establece el material para mostrar el plano
            D3DDevice.DrawPrimitiveUP D3DPT_LINELIST, Lineas_verticales + Lineas_horizontales, Cuadricula(0),
Len(Cuadricula(0))  'Se agrega la cuadricula
        End If
        For j = Abs(CInt(cargando)) To Num_Objetos - 1 'Bucle que dibuja cada uno de los objetos insertados
            If j = 0 Then
                D3DDevice.SetRenderState D3DRS_FILLMODE, D3DFILL_WIREFRAME
            For i = 0 To Llanta_delantera_virtual.nMaterials - 1
                D3DDevice.SetMaterial Llanta_delantera_virtual.MatList(i) 'Establece el material activo
                D3DDevice.SetTexture 0, Llanta_delantera_virtual.TexList(i) 'Establece la textura activa
                Llanta_delantera_virtual.Mesh.DrawSubset i 'Dibuja la parte de la malla en el momento
            Next
            Else 'No cambiar nada
            D3DDevice.SetRenderState D3DRS_FILLMODE, D3DFILL_SOLID
            End If
            For i = 0 To Objetos(j).nMaterials - 1
                D3DDevice.SetMaterial Objetos(j).MatList(i) 'Establece el material activo
                D3DDevice.SetTexture 0, Objetos(j).TexList(i) 'Establece la textura activa
                Objetos(j).Mesh.DrawSubset i 'Dibuja la parte de la malla en el momento
            Next
        Next
        If Not cargando Then
            D3DDevice.SetRenderState D3DRS_FILLMODE, D3DFILL_SOLID
            For i = 0 To Carro_real.nMaterials - 1
                D3DDevice.SetMaterial Carro_real.MatList(i) 'Establece el material activo
                D3DDevice.SetTexture 0, Carro_real.TexList(i) 'Establece la textura activa
                Carro_real.Mesh.DrawSubset i 'Dibuja la parte de la malla en el momento
            Next
            For i = 0 To Llanta_delantera_real.nMaterials - 1
                D3DDevice.SetMaterial Llanta_delantera_real.MatList(i) 'Establece el material activo
                D3DDevice.SetTexture 0, Llanta_delantera_real.TexList(i) 'Establece la textura activa
                Llanta_delantera_real.Mesh.DrawSubset i 'Dibuja la parte de la malla en el momento
            Next
        End If
        D3DDevice.EndScene 'Termina la escena
End Sub
```

4.0.2 Inicialización del Ambiente

Una parte muy importante en el desarrollo de la aplicación además de la creación del dispositivo gráfico es la definición del sistema de coordenadas, el punto en donde se ubicará la cámara y la perspectiva que se tendrá del ambiente; estos tres aspectos se cubren al asignar la matriz correspondiente al dispositivo, con esto se define como se verá la imagen en la pantalla. Esto se puede analizar en la función de **Iniciar_Vista** la cual se muestra a continuación:

```
Public Sub Iniciar_Vista(D3DDevice As Direct3DDevice8, vector_camara As D3DVECTOR, vector_punto As D3DVECTOR,
ArribaY As Boolean, profundidad As Integer)
'Se inicializan las matrices para la presentación, Matriz de presentación
   D3DXMatrixIdentity matWorld
   D3DDevice.SetTransform D3DTS_WORLD, matWorld

'Matriz de Vista

    If ArribaY = True Then
        D3DXMatrixLookAtLH matView, vector_camara, vector_punto, MakeVector(0, 1, 0)
                        'Determina desde dónde va a ser la vista del picture en 2D
    Else
        D3DXMatrixLookAtLH matView, vector_camara, vector_punto, MakeVector(0, 0, 1)
                        'Determina desde dónde va a ser la vista del picture en 3D
    End If
    D3DDevice.SetTransform D3DTS_VIEW, matView

'Matriz de Proyección

   D3DXMatrixPerspectiveFovLH matproj, pi / 4, 1, 0 1, profundidad
   D3DDevice.SetTransform D3DTS_PROJECTION, matproj
End Sub
```

Además, aunque no esenciales, existen un par de funciones que se encargan de orientar al usuario dentro del ambiente las cuales son **inicia_plano** e **Iniciar_cuadricula. inicia_plano** carga una textura al dispositivo y la muestra como el piso de referencia. Por otra parte **Iniciar_cuadricula** tiene la misma función de proporcionar la referencia del piso pero a través de una cuadricula. Los códigos de las funciones se muestran a continuación:

```
'Inicializa los parámetros que constituirán el plano
Public Sub inicia_plano(ByRef D3DDevice As Direct3DDevice8)

plano(0) = Crear_Vertice(-1000, 0, 1000, 0, 0, 0, 0, 0) 'Primer punto
plano(1) = Crear_Vertice(1000, 0, 1000, 0, 0, 0, 0, 1) 'Segundo punto
plano(2) = Crear_Vertice(-1000, 0, -1000, 0, 0, 0, 1, 0) 'Tercer punto
plano(3) = Crear_Vertice(1000, 0, -1000, 0, 0, 0, 1, 1) 'Cuarto punto
  vN = Generar_Normales(plano(0), plano(1), plano(2)) 'Calcula las normales para el primer triangulo
       plano(0).nx = vN.X: plano(0).ny = vN.Y: plano(0).nz = vN.z 'Asigna los valores a cada vértice del triangulo
       plano(1).nx = vN.X: plano(1).ny = vN.Y: plano(1).nz = vN.z
       plano(2).nx = vN.X: plano(2).ny = vN.Y: plano(2).nz = vN.z
  vN = Generar_Normales(plano(1), plano(3), plano(2)) 'Genera las normales para el segundo triangulo
       plano(1).nx = vN.X: plano(1).ny = vN.Y: plano(1).nz = vN.z 'Asigna las normales al segundo triangulo
       plano(2).nx = vN.X: plano(2).ny = vN.Y: plano(2).nz = vN.z
       plano(3).nx = vN.X: plano(3).ny = vN.Y: plano(3).nz = vN.z
col.a = 0.1: col.r = 1: col.g = 0.9: col.b = 1 'Inicia el color del material
material.Ambient = col 'Inicia el material
material.diffuse = col

'Crea un buffer de inicialización de vértices

  Set VBuffer = D3DDevice.Crear_VerticeBuffer(Len(plano(0)) * 4, 0, Unlit_FVF, D3DPOOL_DEFAULT)
  If VBuffer Is Nothing Then
    MsgBox "No se pudo crear el plano"
    Unload Me
```

```
    End If '//Error handler

'Rellena el buffer con los datos necesarios

D3DVertexBuffer8SetData VBuffer, 0, Len(plano(0)) * 4, 0, plano(0)
End Sub


Public Sub Iniciar_cuadricula(Espacio As Single)

Dim Ancho As Single
Dim Alto As Single
Dim i As Integer, j As Integer
   Ancho = plano(1).X - plano(0).X 'Define el ancho de la superficie a cuadricular
   Alto = plano(0).z - plano(2).z 'Define el alto de la superficie a cuadricular
Lineas_verticales = Ancho \ Espacio + 1 'Define el número de lineas verticales que contendrá la cuadricula
Lineas_horizontales = Alto \ Espacio + 1 'Define el número de  lineas horizontales que contendrá la cuadricula
ReDim Cuadricula(2 * Lineas_verticales - 1) As UnlitVertex
j = 0
For i = 0 To Lineas_verticales - 1
   Cuadricula(j).X = plano(0).X + Espacio * i
   Cuadricula(j).Y = plano(0).Y
   Cuadricula(j).z = plano(0).z
   j = j + 1
   Cuadricula(j).X = plano(2).X + Espacio * i
   Cuadricula(j).Y = plano(2).Y
   Cuadricula(j).z = plano(2).z
   j = j + 1
   Next
ReDim Preserve Cuadricula(2 * Lineas_verticales + 2 * Lineas_horizontales - 1) As UnlitVertex
j = 2 * Lineas_verticales
For i = 0 To Lineas_horizontales - 1
   Cuadricula(j).X = plano(2).X
   Cuadricula(j).Y = plano(2).Y
   Cuadricula(j).z = plano(2).z + Espacio * i
   j = j + 1
   Cuadricula(j).X = plano(3).X
   Cuadricula(j).Y = plano(3).Y
   Cuadricula(j).z = plano(3).z + Espacio * i
   j = j + 1
Next

'Asignar el color a la cuadricula

With col_cuadricula
 .a = 0.5
 .r = 0.3
 .g = 0.3
 .b = 0.5
End With
material_cuadricula.Ambient = col_cuadricula
material_cuadricula.diffuse = col_cuadricula
End Sub
```

4.0.3 Carga de los archivos .X

Para importar las geometrías prediseñadas en alguna otra aplicación de diseño tridimensional y aprovecharlas en el desarrollo de nuestro ambiente, recurrimos a funciones de DirectX específicamente a las que proporciona la librería **D3DX8** las cuales permiten la carga de archivos en formato .X, obteniendo información sobre la geometría prediseñada tal como es el tipo de material y textura de cada uno de los subgrupos que integran la geometría importada. La función que se encarga de realizar esta tarea es **Crear_Nuevo_Objeto**; además de cargar archivos, esta función llena una estructura de datos que permite inicializar adecuadamente el objeto en la aplicación, esta estructura contiene información referente a su posición, rotación, dirección, centro, coordenadas máximas y mínimas de su caja envolvente, datos que permiten que la aplicación manipule al objeto. El código de la función se muestra a continuación:

```
'Función generadora de cuadros directores
Private Function Crear_Nuevo_Objeto(Filename As String, TexturePrefix As String, ByRef key As KeyFrame, D3DDevice
As Direct3DDevice8)
On Error GoTo ErrOut 'Manejador de errores

'Variables requeridas

Dim i As Long
Dim XBuffer As D3DXBuffer 'Buffer que almacena los materiales del archivo X
Dim TextureFile As String 'Nombre de las texturas del objeto
Dim hResult As Long 'Variable para determinar el éxito de la lectura de vértices

'Cargar el archivo .X en memoria

Set key.Mesh = D3DX.LoadMeshFromX(Filename, D3DXMESH_MANAGED, D3DDevice, Nothing, XBuffer, key.nMaterials)
If key.Mesh Is Nothing Then GoTo ErrOut 'No continuar si no se pudo cargar el archivo

'Generar materiales y texturas

ReDim key.MatList(key.nMaterials) As D3DMATERIAL8
ReDim key.TexList(key.nMaterials) As Direct3DTexture8
ReDim key.TexList_Nombres(key.nMaterials) As String
For i = 0 To key.nMaterials - 1

'Obtener los materiales que cargamos en el buffer e insertarlos en nuestro arreglo
  D3DX.BufferGetMaterial XBuffer, i, key.MatList(i)

'En directX La luz de ambiente y la difusa son las mismas

  key.MatList(i).Ambient = key.MatList(i).diffuse
'Obtener el nombre de la Textura utilizada para esta parte de la malla
  TextureFile = D3DX.BufferGetTextureName(XBuffer, i)
'Crear la Textura
  key.TexList_Nombres(i) = TextureFile 'Guarda el nombre de la textura
  If TextureFile <> "" Then 'No permite crear una textura cuando no la hay
    On Error GoTo ErrOut
```

```
    Set key.TexList(i) = D3DX.CreateTextureFromFileEx(D3DDevice, TexturePrefix & TextureFile, 128, 128,
D3DX_DEFAULT, 0, D3DFMT_UNKNOWN, D3DPOOL_MANAGED, D3DX_FILTER_LINEAR, D3DX_FILTER_LINEAR, 0,
ByVal 0, ByVal 0)
  End If
Next i

'Calcula la caja envolvente del objeto recién cargado

D3DX.ComputeBoundingBoxFromMesh key.Mesh, key.esquina_inf_izq_min, key.esquina_sup_der_max
'Inicializa el centro del objeto leído

  key.Centro.X = key.esquina_inf_izq_min.X + (key.esquina_sup_der_max.X - key.esquina_inf_izq_min.X) / 2
  key.Centro.Y = 0
  key.Centro.z = key.esquina_inf_izq_min.z + (key.esquina_sup_der_max.z - key.esquina_inf_izq_min.z) / 2
'Establece el radio mínimo de la circunferencia del área que ocupa

  key.radio = Generar_Radio(key.Centro, key.esquina_inf_izq_min)
'Inicializa la dirección del objeto leído en dirección z

  key.direccion.X = 0
  key.direccion.Y = 0
  key.direccion.z = 1
  key.grados = 90 'Inclinación inicial respecto del eje x
  key.Archivo = Filename
Exit Function 'Si llega a este punto la función se logró con éxito
  ErrOut:
MsgBox "Ha ocurrido un error al generar un cuadro llave del archivo " & Filename
Unload Me
End Function
```

4.0.4 Métodos de inserción

Existen dos formas de agregar los objetos al ambiente, una más precisa(teclado) que la otra(mouse). La inserción por teclado tiene la desventaja de que a pesar de ser un método exacto el usuario tiene que estar muy familiarizado con el ambiente y el tipo de objetos que desee insertar pero sobre todo con el sistema de coordenadas que se utiliza en esta aplicación.

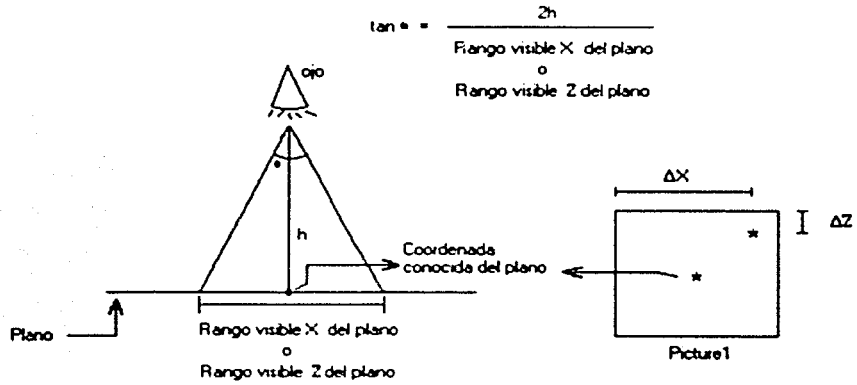
El método de inserción por teclado se programó con dos cajas de texto modificables por el usuario las cuales establecen las coordenadas(X, Z) donde se encontrará el centro del objeto a insertar. La función que se encarga de aplicar la translación del objeto a las coordenadas especificadas es:

```
Private Sub Command1_Click()' boton de insertar 'Tomar los valores de insercion y llevar la figura hasta el lugar
especificado en las celdas
On Error Resume Next ' habilita el control de errores
If cargando = False Then
    Translacion -Objetos(Num_Objetos).Centro.X + CDbl(Text1.Text), 0, -Objetos(Num_Objetos).Centro.z +
CDbl(Text2.Text), Objetos(Num_Objetos), False ' translada el obejeto de insercion a las nuevas
Else
        Translacion -Objetos(0).Centro.X + CDbl(Text1.Text), 0, -Objetos(0).Centro.z + CDbl(Text2.Text), Objetos(0), False
        ' translada el objeto de inserción a las nuevas coordenadas
End If
If Err.Number <> 0 Then 'Hubo un error al especificar los números
  Text1.Text = "0"
```

```
    Text2.Text = "0"
    On Error GoTo 0 'Limpia la variable de errores
    Exit Sub
End If
Command1.Enabled = False  ' Inhabilita el boton de inserción por teclado
Picture1_MouseUp 1, 1, 1, 1  ' manda llamar a la rutina de inserción que esta especificada en el ratón
End Sub
```

El método de inserción por mouse a diferencia del anterior permite al usuario insertar el objeto en la posición geométrica que coincida con la posición en la cual se de un click dentro de la caja de imagen(picture 1) que muestra el ambiente en una vista bidimensional. El método que se diseñó para capturar la coordenada geométrica correspondiente a la posición de inserción consiste en lo siguiente:

Se apoyó en el hecho de que se debe definir un sistema de coordenadas en el cual se orientará a los objetos en el ambiente. La matriz que define el sistema de coordenadas se conoce como **worldmatrix;** esta matriz se especificó de tal forma que se tratase de un sistema de coordenada ortogonal. Apoyada en el hecho de que se debe definir un punto desde el cual se contemplará el ambiente geométrico. Dentro de los parámetros que se deben especificar se encuentra la definición de un vector que proporciona la dirección de cómo se vera la imagen, el cual en este caso coincide con el eje Z cuando se pretende insertar un nuevos objeto y la vista es bidimensional; esto ocasiona que el eje vertical del picture1 sea paralelo con el eje Z y el eje horizontal sea paralelo con el eje X. Por último se debe definir adicionalmente una matriz de perspectiva la cual especifica dentro de sus parámetros un ángulo de visión en radianes. Todo lo anterior se puede visualizar en la siguiente figura(16):



Figura(16): *Especifica la ubicación de los objetos realesen las ventanas de visualización.*

Una vez tomado en cuenta lo anterior se sabe que al centro del picture1 le corresponde el vector punto actual, el cual proporciona un punto del plano XZ. Entonces si se conoce h se puede calcular el rango visible del plano, ya sea en su dirección X ó Y; además se puede conocer el ancho y el alto del picture1. La altura h se conoce de la distancia entre el plano y el punto conocido como vector_camara(variable del programa). Así, se pueden realizar las siguientes equivalencias:

$$\Delta X_{delplano} = \frac{X_{Picture1}(RangovisibleenX_{delplano})}{Ancho_{Picture1}} \tag{24}$$

$$\Delta Z_{delplano} = \frac{Y_{Picture1}(RangovisibleenZ_{delplano})}{Alto_{Picture1}} \tag{25}$$

Las X y Y del picture1 se obtienen del click generado en el picture1 medidos desde la esquina superior izquierda, como se presentó en la figura anterior. Así, para obtener la coordenada geométrica real donde se desea insertar el centro del objeto, se calcula con la ecuación (26) y (27):

$$X_{insercion} = X_{coordenadascomando} - \frac{RangovisibleenX_{delplano}}{2} + \Delta X_{delplano} \tag{26}$$

$$Z_{insercion} = Z_{coordenadascomando} + \frac{RangovisibleenZ_{delplano}}{2} - \Delta Z_{delplano} \tag{27}$$

Con esto se obtienen las coordenadas de inserción que se deben utilizar para transladar el objeto. El cálculo de estas coordenadas se lleva a cabo en el evento **Picture1_MouseUp** cuyo código se muestra a continuación:

```
' Toma las coordenadas originales del ratón para poder hacer la inserción
Private Sub Picture1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)

'Calcular las Coordenadas de Insercion

Dim Ancho As Double   ' variables locales que se utilizan para el calculo de las coordenadas de inserción de objeto en el
picture
Dim Alto As Double
Dim Coordenadax As Double
Dim Coordenadaz As Double
If Esperando_Pick = True Then
   If cargando = False Then
      Esperando_Pick = False
      Insertar.Enabled = True
      StatusBar1.SimpleText = "En Render"
```

```
        vector_camara = Construye_Vector(Objetos(0).Centro.X - Alejamiento_x * Objetos(0).direccion.X,
Objetos(0).Centro.Y + (Objetos(0).esquina_sup_der_max.Y - Objetos(0).esquina_inf_izq_min.Y), Objetos(0).Centro.z -
Alejamiento_z * Objetos(0).direccion.z) 'Inicializa el punto de vista'Inicializa el punto de vista

'Actualizar el vector cámara2

    vector_camara2 = Construye_Vector(Objetos(0).Centro.X, Objetos(0).Centro.Y + UpDown3.Value, Objetos(0).Centro.z)
    vector_punto = Construye_Vector(Objetos(0).Centro.X, Objetos(0).Centro.Y + Levantamiento_y, Objetos(0).Centro.z)
  End If

  If Command1.Enabled = True Then 'Inserción por mouse
      Command1.Enabled = False
'Calcula las coordenadas reales geométricamente

      Ancho = Picture1.Width
      Alto = Picture1.Height
      Coordenadax = HScroll1.Value- (plano(0).z * Tan(pi / 8)) * (1 - 2 * X / Ancho)
      Coordenadaz = -VScroll1.Value + (plano(0).z * Tan(pi / 8)) * (1 - 2 * Y / Alto)
    If cargando = False Then
      'Transladar el objeto al punto elegido para el centro fisico del objeto al punto que se eligio con el mouse
      Translacion Coordenadax - Objetos(Num_Objetos).Centro.X, 0, Coordenadaz -
Objetos(Num_Objetos).Centro.z, Objetos(Num_Objetos), False
    Else 'Insertando Posición Inicial del carro

'Transladar el objeto al punto elegido para el centro fisico del objeto al punto que se eligio con el mouse

      Translacion Coordenadax -Objetos(0).Centro.X, 0, Coordenadaz- Objetos(0).Centro.z,
      Llanta_delantera_virtual, False
      Translacion Coordenadax - Objetos(0).Centro.X, 0, Coordenadaz- Objetos(0).Centro.z, Objetos(0), True
      Translacion Coordenadax -Carro_real.Centro.X, 0, Coordenadaz - Carro_real.Centro.z, Llanta_delantera_real,
False
      Translacion Coordenadax -Carro_real.Centro.X, 0, Coordenadaz -Carro_real.Centro.z, Carro_real, False

    End If
  End If

'Agrega el Objeto al ambiente

  If cargando = False Then
      If Not Colision(Objetos(Num_Objetos), Num_Objetos) Then 'Se puede insertar
        Num_Objetos = Num_Objetos + 1
        Exit Sub 'Termina Agregando Objeto
      End If
  Else
      If Not Colision(Objetos(0), 0) Then 'Se puede insertar el carro
        cargando = False 'Termina de cargar el archivo
        Esperando_Pick = False
        Insertar.Enabled = True
        If Joystick.Checked = True Then
          Form1.JoyStick1.Interval = 10 'Habilita el dispositivo de conducción
        End If

'Actualizar el vector cámara
```

```
            vector_camara = Construye_Vector(Objetos(0).Centro.X - Alejamiento_x * Objetos(0).direccion.X,
Objetos(0).Centro.Y + (Objetos(0).esquina_sup_der_max.Y - Objetos(0).esquina_inf_izq_min.Y), Objetos(0).Centro.z -
Alejamiento_z * Objetos(0).direccion.z) 'Inicializa el punto de vista

'Actualizar el vector cámara2

            vector_camara2 = Construye_Vector(Objetos(0).Centro.X, Objetos(0).Centro.Y + UpDown3.Value,
Objetos(0).Centro.z)
            vector_punto = Construye_Vector(Objetos(0).Centro.X, Objetos(0).Centro.Y + Levantamiento_y,
Objetos(0).Centro.z)
            With MSFlexGrid1
              .col = 1
              .Row = 0 'Coordenadas virtuales
              .Text = "(" & CStr(CInt(Objetos(0).Centro.X)) & "," & CStr(CInt(Objetos(0).Centro.z)) & ")"
              .Row = 2
              .Text = "(" & CStr(CSng(Objetos(0).direccion.X)) & "," & CStr(CSng(Objetos(0).direccion.z)) & ")"
            End With
            StatusBar1.SimpleText = "En Render"
          Exit Sub 'Termina Agregando Objeto
        End If
        Command1.Enabled = True 'Rehabilita la inserción
    End If
    MsgBox "El objeto se inserta en una posición no permitida." 'Termina Sin agregar Objeto
End If
End Sub
```

4.0.5 Guardado y Apertura del ambiente

Para conservar los ambientes generados previamente se desarrollaron dos funciones encargadas de guardar y abrir los archivos, así como la definición de la organización e interpretación de los datos dentro de los archivos. Debido a la complejidad geométrica posible de un ambiente se optó por guardar en un archivo de texto las posiciones de los objetos finales dentro del ambiente además de una referencia al archivo que contiene la figura prediseñada. Por cada objeto encontrado en el ambiente se crea una copia del archivo que lo contiene y se translada a una carpeta generada previamente en la misma ruta en donde se desea guardar el ambiente. Junto con estos datos se guarda la dimensión del plano, pero no se guarda la posición final del robot, esto con la finalidad de que éste pueda ser insertado en otra parte del ambiente diseñado cuando se vuelva a cargar el ambiente. La función que se encarga de este guardado es .

Guardado:

```
Public Sub Crear_Archivo(File As String) 'Crea un nuevo archivo así como una carpeta para guardar datos

Dim fso As New FileSystemObject, txtfile As TextStream 'fso se encarga de manejar los archivos
Dim folder As String 'Nombre de la carpeta
Dim i As Single, j As Single
Dim textura As String
folder = Mid(File, 1, Len(File) - 4)
Set fso = CreateObject("Scripting.FileSystemObject")
On Error Resume Next
  fso.CreateFolder folder  'Crea el folder con el mismo nombre que el archivo
On Error GoTo ErrHand
```

```
  If fso.FileExists(File) Then 'El archivo existe previamente
    fso.DeleteFile File 'Borra el archivo existente
  End If
  Set txtfile = fso.CreateTextFile(File, True) 'Crea el archivo de texto
  txtfile.WriteLine (CStr(plano(0).X)) 'Guarda las dimensiones del plano
  txtfile.WriteLine (CStr(plano(0).Y))
  txtfile.WriteLine (CStr(plano(0).z))
  txtfile.WriteLine (CStr(plano(3).X))
  txtfile.WriteLine (CStr(plano(3).Y))
  txtfile.WriteLine (CStr(plano(3).z))
  txtfile.WriteLine (CStr(Num_Objetos)) 'Guarda la dimensión de la Matriz
  For i = 1 To Num_Objetos - 1 'Lleva todos los archivos .X requeridos al folder
    fso.CopyFile Objetos(i).Archivo, folder & "\", True  'Mueve el archivo dentro de la carpeta
    For j = 0 To Objetos(i).nMatenals - 1
     'Copiar las texturas correspondientes
        textura = Objetos(i).TexList_Nombres(j)
        If textura <> "" Then 'Se debe copiar
          fso.CopyFile Mid(Objetos(i).Archivo, 1, Len(Objetos(i).Archivo) - Len(fso.GetFileName(Objetos(i).Archivo))) &
textura, folder & "\", True
        End If
    Next
    txtfile.WriteLine fso.GetFileName(Objetos(i).Archivo) 'Escribe en el archivo amb el nombre del archivo que se copió.
    txtfile.WriteLine CStr(Objetos(i).Centro.X)  'Escribe la coordenada x del centro del objeto
    txtfile.WriteLine CStr(Objetos(i).Centro.Y)  'Escribe la coordenada x del centro del objeto
    txtfile.WriteLine CStr(Objetos(i).Centro.z)  'Escribe la coordenada x del centro del objeto
  Next
  txtfile.Close 'Se cierra el hilo para operar dentro del archivo
  Exit Sub
ErrHand:
  MsgBox "Error en la operación de Guardado"
  On Error GoTo 0
End Sub
```

Para la función de **Apertura** se lee el archivo de texto y se cargan los objetos geométricos referenciados y se buscan en la carpeta con el mismo nombre del archivo que se debe encontrar en la misma ruta. Teniendo los datos de ese archivo de texto, los objetos cargados se transladan a las coordenadas correspondientes y se cargan tanto el robot real como el robot virtual esperando que se dé el click para su posición de inserción. El código que se encarga de esto es:

```
Public Sub Abrir_Archivo(File As String)

Dim fso As New FileSystemObject, txtfile As TextStream 'fso maneja operaciones con archivos
Dim Linea As String 'Contiene el texto leído del archivo
Dim i As Single
Dim j As Single
Dim folder As String 'Nombre del folder que contiene los archivos x y las texturas
Dim Cantidad As Long 'Número de objetos que tiene el ambiente
Dim Centro As D3DVECTOR 'Centro del objeto en el ambiente
Set fso = CreateObject("Scripting.FileSystemObject")
Set txtfile = fso.OpenTextFile(File, ForReading, False, TristateFalse) 'Abrir el archivo de texto
folder = Mid(File, 1, Len(File) - 4)
On Error GoTo Salida
  For i = 0 To 5
    If Not (txtfile.AtEndOfStream) Then
```

```
      Linea = txtfile.ReadLine
    Else
      GoTo Salida: 'Rompe el loop
    End If
    Select Case i 'Iniciar el plano
      Case 0
        plano(0).X = CDbl(Linea)
        plano(2).X = CDbl(Linea)
      Case 1
        plano(0).Y = CDbl(Linea)
        plano(1).Y = CDbl(Linea)
        plano(2).Y = CDbl(Linea)
      Case 2
        plano(0).z = CDbl(Linea)
        plano(1).z = CDbl(Linea)
      Case 3
        plano(3).X = CDbl(Linea)
        plano(1).X = CDbl(Linea)
      Case 4
        plano(3).Y = CDbl(Linea)
      Case 5
        plano(3).z = CDbl(Linea)
        plano(2).z = CDbl(Linea)
    End Select
  Next
  If Not (txtfile.AtEndOfStream) Then
      Linea = txtfile.ReadLine
  Else
      GoTo Salida  'Rompe el loop
  End If
  Cantidad = CLng(Linea)
  ReDim Objetos(0) As KeyFrame 'Limpia la matriz
  Num_Objetos = 0 'Inicia la Variable de la matriz de objetos
  Agregar_ObjetoN (App.Path & "\carrovirtual.x") 'Agregar el objeto principal (Carro)
  Num_Objetos = Num_Objetos + 1
  Crear_Nuevo_Objeto App.Path & "\llantavirtual.x", App.Path & "\", Llanta_delantera_virtual, D3DDevice
  Crear_Nuevo_Objeto App.Path & "\carroreal.x", App.Path & "\", Carro_real, D3DDevice
  Crear_Nuevo_Objeto App.Path & "\llantavirtual.x", App.Path & "\", Llanta_delantera_virtual, D3DDevice
  Crear_Nuevo_Objeto App.Path & "\llantareal.x", App.Path & "\", Llanta_delantera_real, D3DDevice
  For i = 1 To Cantidad - 1
    If Not (txtfile.AtEndOfStream) Then
      Linea = txtfile.ReadLine
    Else
      GoTo Salida  'Rompe el loop
    End If

   'Leer Archivo

    Agregar_ObjetoN (folder & "\" & Linea) 'Carga objeto
    Num_Objetos = Num_Objetos + 1
    If Not (txtfile.AtEndOfStream) Then
      Linea = txtfile.ReadLine
    Else
      GoTo Salida  'Rompe el loop
    End If
        Centro.X = CDbl(Linea) '
```

```
    If Not (txtfile.AtEndOfStream) Then
      Linea = txtfile.ReadLine
    Else
      GoTo Salida: 'Rompe el loop
    End If
    Centro.Y = CDbl(Linea) '
    If Not (txtfile.AtEndOfStream) Then
      Linea = txtfile.ReadLine
    Else
      GoTo Salida: 'Rompe el loop
    End If

    Centro.z = CDbl(Linea) '

'Transladar de su centro en el archivo hasta el centro en el ambiente

Translacion Centro.X - Objetos(i).Centro.X, Centro.Y - Objetos(i).Centro.Y, Centro.z - Objetos(i).Centro.z, Objetos(i), False
  Next
    txtfile.Close 'Cierra el archivo

  'Cambia la Matriz de Vista para ver el ambiente en 2D

  vector_camara = Construye_Vector(HScroll1.Value, plano(0).z, -VScroll1.Value)
  vector_punto = Construye_Vector(HScroll1.Value, 0, -VScroll1.Value)
  vector_camara2 = Construye_Vector(HScroll1.Value, UpDown3.Value, -VScroll1.Value)
  Text1.Text = CStr(Objetos(0).Centro.X) ' coordenadas de inserción del objeto originales
  Text2.Text = CStr(Objetos(0).Centro.z)
  Command1.Enabled = True ' activa el boton de insertar
  Insertar.Enabled = False 'Desactiva la inserción
  cargando = True
  Esperando_Pick = True
  StatusBar1.SimpleText = "Esperando pick para la posición inicial del carro"
  Exit Sub
Salida:
  MsgBox "Hubo Un erro al cargar el archivo"
On Error GoTo 0
End Sub
```

4.0.6 Métodos de Conducción

Traslación

Como se mencionó anteriormente, para animar un objeto existen dos operaciones básicas: translación y rotación; en este caso, la translación del objeto se logra extrayendo sus vértices, se aplica el incremento o decremento en sus coordenadas y se actualiza el objeto geométrico. El código resultante fue:

```
Private Function Translacion(X As Single, Y As Single, z As Single, ByRef key As KeyFrame, Camara As Boolean)  '
Desarrolla la translación de los objetos
Dim i As Single  ' variable de bucle
Dim VertexList() As D3DVERTEX 'El arreglo de vertices de la geometria
ReDim VertexList(key.Mesh.GetNumVertices) As D3DVERTEX
i = D3DXMeshVertexBuffer8GetData(key.Mesh, 0, Len(VertexList(0)) * key.Mesh.GetNumVertices, 0, VertexList(0))
If Not (i = D3D_OK) Then GoTo ErrOut  'Termina si no puede obtener los datos
```

```
For i = 0 To key.Mesh.GetNumVertices - 1
   With VertexList(i) ' agrega el incremento a cada coordenada
      .X = .X + X
      .Y = .Y + Y
      .z = .z + z
   End With
Next i
   With key.Centro  ' agrega el incremento a cada coordenada pero para el centro
      .X = .X + X
      .Y = .Y + Y
      .z = .z + z
   End With
If Camara = True And Esperando_Pick = False Then
   With vector_camara ' actualiza la posicion de la camara
      .X = .X + X
      .Y = .Y + Y
      .z = .z + z
   End With
   With vector_camara2 ' actualiza la posicion de la camara
      .X = .X + X
      .Y = .Y + Y
      .z = .z + z
   End With
   With vector_punto
      .X = .X + X
      .Y = .Y + Y
      .z = .z + z
   End With
End If
i = D3DXMeshVertexBuffer8SetData(key.Mesh, 0, (Len(VertexList(0)) * key.Mesh.GetNumVertices), 0, VertexList(0))
 '  actualiza los valores en el buffer de la malla

'Actualiza las coordenadas de la Caja

D3DX.ComputeBoundingBoxFromMesh key.Mesh, key.esquina_inf_izq_min, key.esquina_sup_der_max

'Actualizar tabla de datos

With MSFlexGrid1
    .col = 1
    .Row = 0
    .Text = "(" & CStr(CInt(Objetos(0).Centro.X)) & "," & CStr(CInt(Objetos(0).Centro.z)) & ")"
End With
Avance = Avance + Incremento 'Calcula cuanto ha avanzado el carro
If Not (i = D3D_OK) Then  ' se fija que no haya ocurrido un error
ErrOut
MsgBox "No se pudo transladar" '  si ocurrio un error manda un mensaje
End
End If   ' terminación del manejo de errores
End Function
```

Rotación

De la misma manera, en esta función se tuvo que recurrir a la extracción de los vértices pero además especificar un punto fijo respecto al cual se rotará el objeto. El código que desarrolla esta tarea es:

```
'Rota un objeto cargado al rededor de su centro
Private Sub Rotar(ByRef key As KeyFrame, grados As Single, Apoyo As D3DVECTOR, Camara As Boolean,
Actualizar_grados As Boolean)
Dim i As Long 'Variable de bucle
Dim tempx As Single 'Variable temporal
Dim tempz As Single 'Variable temporal
Dim VertexList() As D3DVERTEX 'El arreglo de vértices de la geometria
ReDim VertexList(key.Mesh.GetNumVertices) As D3DVERTEX
tempx = D3DXMeshVertexBuffer8GetData(key.Mesh, 0, Len(VertexList(0)) * key.Mesh.GetNumVertices, 0, VertexList(0))
  If Not (tempx = D3D_OK) Then GoTo ErrOut 'Termina si no puede obtener los datos
  For i = 0 To key.Mesh.GetNumVertices - 1
    With VertexList(i)
      tempx = .nx - Apoyo.X 'Traslada las normales al centro
      tempz = .nz - Apoyo.z
      'Las rota y regresa a su posición original
      .nz = (tempz * Cos(grados * Rad) - tempx * Sin(grados * Rad)) + Apoyo.z
      .nx = (tempz * Sin(grados * Rad) + tempx * Cos(grados * Rad)) + Apoyo.X
      tempx = .X - Apoyo.X 'Traslada los vértices al centro
      tempz = .z - Apoyo.z
      'Los rota y regresa a su posicion original
      .z = (tempz * Cos(grados * Rad) - tempx * Sin(grados * Rad)) + Apoyo.z
      .X = (tempz * Sin(grados * Rad) + tempx * Cos(grados * Rad)) + Apoyo.X
    End With
  Next i
  With key.direccion
    tempx = .X 'Guarda la inclinación original
    tempz = .z
      'Rota
    .z = (tempz * Cos(grados * Rad) - tempx * Sin(grados * Rad))
    .X = (tempz * Sin(grados * Rad) + tempx * Cos(grados * Rad))
  End With
  'Actualiza el punto de vista de la camara
  If Camara = True And Esperando_Pick = False Then
    With vector_camara
      tempx = .X - Apoyo.X 'Lleva el punto de dirección
      tempz = .z - Apoyo.z
      'Rota y lo regresa a su posición original
      .z = (tempz * Cos(grados * Rad) - tempx * Sin(grados * Rad)) + Apoyo.z
      .X = (tempz * Sin(grados * Rad) + tempx * Cos(grados * Rad)) + Apoyo.X
    End With
  End If

'Actualiza los vértices en el buffer de vértices de la malla

tempx = D3DXMeshVertexBuffer8SetData(key.Mesh, 0, (Len(VertexList(0)) * key.Mesh.GetNumVertices), _
                    0, VertexList(0))

'Actualiza la Caja
```

```
  D3DX.ComputeBoundingBoxFromMesh key.Mesh, key.esquina_inf_izq_min, key.esquina_sup_der_max
  key.Centro.X = key.esquina_inf_izq_min.X + (key.esquina_sup_der_max.X - key.esquina_inf_izq_min.X) / 2
  key.Centro.Y = 0
  key.Centro.z = key.esquina_inf_izq_min.z + (key.esquina_sup_der_max.z - key.esquina_inf_izq_min.z) / 2
  If Actualizar_grados Then
     key.grados = key.grados + grados
     If key.grados > 360 Then
        key.grados = key.grados - 360
     ElseIf key.grados < 0 Then
        key.grados = key.grados + 360
     End If
  End If

'Actualiza los datos de la tabla

  With MSFlexGrid1
     .col = 1
     .Row = 2 'Dirección virtual
     .Text = "(" & CStr(CSng(Objetos(0).direccion.X)) & "," & CStr(CSng(Objetos(0).direccion.z)) & ")"
  End With
If Not (tempx = D3D_OK) Then
ErrOut:
MsgBox "No se pudo rotar" 'En caso de error termina el programa
End
End If
End Sub
```

Las funciones que se encargan de llamar a las anteriores son aquellas que se refieren al dispositivo que se utiliza para conducir al carrito las cuales son: atención del Teclado y Atención por Joystick.

Atención del teclado y Joystick

Tanto en el teclado como en el Joystick se desarrolló el mismo algoritmo para el manejo del robot adaptando la respuesta de acuerdo a las teclas presionadas o estado de la palanca y botones para el caso del joystick. Al contar el robot con una sola rueda motorizada la cual tiene la capacidad de rotar sobre su propio eje además de rotar hacia delante y hacia atrás se determinó que con las teclas de cursor y Ctrl se rota la llanta sobre su eje, por ejemplo Ctrl. + ← (tecla cursor izquierda) rota la llanta delantera en sentido antihorario. Con lo que respecta al joystick para poder girar la llanta es necesario presionar el boton 1 y con la palanca cargada ala izquierda. Por otra parte con la tecla Shift y las teclas cursor arriba o abajo el movimiento del robot es adelante o atrás respectivamente. Fueron desarrollados estos dos métodos para que el usuario pudiera seleccionar el que más le agrade. La dinámica que se sigue es considerar el estado de las teclas o del joystick para poder determina una translación o rotación y, respecto al estado de las teclas, se envia la instrucción correspodiente al robot real si es que existe una conexión establecida. Como los dos códigos son muy parecidos entonces sólo se muestra el código de **JoyStick1_DataTick**:

```
Private Sub JoyStick1_DataTick(xPos As Long, yPos As Long, zPos As Long, rPos As Long, uPos As Long, vPos As Long,
Pov As Long, CurButton As Long)
  If Comm1.Checked = False And Comm2.Checked = False Or ((Comm1.Checked = True Or Comm2.Checked = True) And
(Listo = True)) Then    'No se tiene comunicación entonces se puede atender
    If CurButton = 1 Then 'El botón 1 está presionado
      If yPos <= Por_minY * JoyStick1.yMax Then
        'Zona superior
        If xPos >= Por_minX * JoyStick1.xMax And xPos <= Por_maxX * JoyStick1.xMax Then
          'Palanca arriba (Adelante y Shifth)
          If Llanta_delantera_virtual.grados >= (90 + Grados_Rotacion) And Incremento <> 0 Then
            'Transladar  hacia adelante y rotar en sentido antihorario
              Rotar Objetos(0), grados, Objetos(0).Centro, True, True
              Translacion Incremento * Objetos(0).direccion.X, 0, Incremento * Objetos(0).direccion.z, Objetos(0), True
              Translacion Incremento * Objetos(0).direccion.X, 0, Incremento * Objetos(0).direccion.z,
Llanta_delantera_virtual, False
              Rotar Llanta_delantera_virtual, grados, Objetos(0).Centro, False, False
              Caracter = "P" 'Transladar Hacia Delante
              Enviar_datos
              If Colision(Objetos(0), 0) Or Colision_real = True Then
                'Revierte la translación y la rotación
                  Translacion -Incremento * Objetos(0).direccion.X, 0, -Incremento * Objetos(0).direccion.z, Objetos(0),
True
                  Translacion -Incremento * Objetos(0).direccion.X, 0, -Incremento * Objetos(0).direccion.z,
Llanta_delantera_virtual, False
                  Caracter = "Q" 'Transladar hacia atrás
                  Enviar_datos
                  Rotar Objetos(0), -grados, Objetos(0).Centro, True, True
                  Rotar Llanta_delantera_virtual, -grados, Objetos(0).Centro, False, False
              End If
              Text8.Text = CDbl(Objetos(0).grados)
          ElseIf Llanta_delantera_virtual.grados <= (90 - Grados_Rotacion) And Incremento <> 0 Then
            'Transladar hacia adelante y rotar en sentido horario
              Rotar Objetos(0), -grados, Objetos(0).Centro, True, True
              Translacion Incremento * Objetos(0).direccion.X, 0, Incremento * Objetos(0).direccion.z, Objetos(0), True
              Translacion Incremento * Objetos(0).direccion.X, 0, Incremento * Objetos(0).direccion.z,
Llanta_delantera_virtual, False
              Rotar Llanta_delantera_virtual, -grados, Objetos(0).Centro, False, False
              Caracter = "P" 'Transladar hacia adelante
              Enviar_datos
              If Colision(Objetos(0), 0) Or Colision_real = True Then
                'Revierte la translación y la rotación
                  Translacion -Incremento * Objetos(0).direccion.X, 0, -Incremento * Objetos(0).direccion.z, Objetos(0),
True
                  Translacion -Incremento * Objetos(0).direccion.X, 0, -Incremento * Objetos(0).direccion.z,
Llanta_delantera_virtual, False
                  Caracter = "Q"
                  Enviar_datos
                  Rotar Objetos(0), grados, Objetos(0).Centro, True, True
                  Rotar Llanta_delantera_virtual, grados, Objetos(0).Centro, False, False
              End If
              Text8.Text = CDbl(Objetos(0).grados)
          ElseIf Incremento <> 0 Then
            'Transladar hacia adelante
              Translacion Incremento * Objetos(0).direccion.X, 0, Incremento * Objetos(0).direccion.z, Objetos(0), True
```

```
                Translacion Incremento * Objetos(0).direccion.X, 0, Incremento * Objetos(0).direccion.z,
Llanta_delantera_virtual, False
                Caracter = "P" 'Translada el carro real hacia adelante
                Enviar_datos
                If Colision(Objetos(0), 0) Or Colision_real = True Then
                  'Revierte la translación. Se supone el ambiente estático
                    Translacion -Incremento * Objetos(0).direccion.X, 0, -Incremento * Objetos(0).direccion.z, Objetos(0),
True
                    Translacion -Incremento * Objetos(0).direccion.X, 0, -Incremento * Objetos(0).direccion.z,
Llanta_delantera_virtual, False
                    Caracter = "Q"
                    Enviar_datos
                End If
                Text8.Text = CDbl(Objetos(0).grados)
            End If
        ElseIf xPos <= Por_minX * JoyStick1.xMax Then

        Else

        End If
      ElseIf yPos >= Por_maxY * JoyStick1.yMax Then
        'Zona inferior
          If xPos >= Por_minX * JoyStick1.xMax And xPos <= Por_maxX * JoyStick1.xMax Then
            'Palanca abajo (Atrás y Shifth)
              If Llanta_delantera_virtual.grados >= (90 + Grados_Rotacion) And Incremento <> 0 Then
                'Transladar  hacia atrás y rotar en sentido horario
                  Rotar Objetos(0), -grados, Objetos(0).Centro, True, True
                  Translacion -Incremento * Objetos(0).direccion.X, 0, -Incremento * Objetos(0).direccion.z, Objetos(0),
True
                  Translacion -Incremento * Objetos(0).direccion.X, 0, -Incremento * Objetos(0).direccion.z,
Llanta_delantera_virtual, False
                  Rotar Llanta_delantera_virtual, -grados, Objetos(0).Centro, False, False
                  Caracter = "Q" 'Transladar hacia atrás
                  Enviar_datos
                  If Colision(Objetos(0), 0) Or Colision_real = True Then
                    'Revierte la translación y la rotación
                      Translacion Incremento * Objetos(0).direccion.X, 0, Incremento * Objetos(0).direccion.z,
Objetos(0), True
                      Translacion Incremento * Objetos(0).direccion.X, 0, Incremento * Objetos(0).direccion.z,
Llanta_delantera_virtual, False
                      Rotar Objetos(0), grados, Objetos(0).Centro, True, True
                      Rotar Llanta_delantera_virtual, grados, Objetos(0).Centro, False, False
                      Caracter = "P" 'Transladar hacia adelante
                      Enviar_datos
                  End If
                  Text8.Text = CDbl(Objetos(0).grados)
              ElseIf Llanta_delantera_virtual.grados <= 90 - (Grados_Rotacion) And Incremento <> 0 Then
                'Transladar hacia atrás y rotar en sentido antihorario
                  Rotar Objetos(0), grados, Objetos(0).Centro, True, True
                  Translacion -Incremento * Objetos(0).direccion.X, 0, -Incremento * Objetos(0).direccion.z, Objetos(0),
True
                  Translacion -Incremento * Objetos(0).direccion.X, 0, -Incremento * Objetos(0).direccion.z,
Llanta_delantera_virtual, False
                  Rotar Llanta_delantera_virtual, grados, Objetos(0).Centro, False, False
                  Caracter = "Q" 'Transladar hacia atrás
                  Enviar_datos
```

```
                    If Colision(Objetos(0), 0) Or Colision_real = True Then
                      'Revierte la translación y la rotación
                        Translacion Incremento * Objetos(0).direccion.X, 0, Incremento * Objetos(0).direccion.z,
Objetos(0), True
                        Translacion Incremento * Objetos(0).direccion.X, 0, Incremento * Objetos(0).direccion.z,
Llanta_delantera_virtual, False
                        Rotar Objetos(0), -grados, Objetos(0).Centro, True, True
                        Rotar Llanta_delantera_virtual, -grados, Objetos(0).Centro, False, False
                        Caracter = "P" 'Transladar hacia adelante
                        Enviar_datos
                    End If
                    Text8.Text = CDbl(Objetos(0).grados)
                  ElseIf Incremento <> 0 Then
                    'Transladar hacia atrás
                    Translacion -Incremento * Objetos(0).direccion.X, 0, -Incremento * Objetos(0).direccion.z, Objetos(0),
True
                    Translacion -Incremento * Objetos(0).direccion.X, 0, -Incremento * Objetos(0).direccion.z,
Llanta_delantera_virtual, False
                    Caracter = "Q" 'Transladar el carro real hacia atrás
                    Enviar_datos
                    If Colision(Objetos(0), 0) Or Colision_real = True Then
                      'Revierte la translación. Se supone el ambiente estático
                        Translacion Incremento * Objetos(0).direccion.X, 0, Incremento * Objetos(0).direccion.z,
Objetos(0), True
                        Translacion Incremento * Objetos(0).direccion.X, 0, Incremento * Objetos(0).direccion.z,
Llanta_delantera_virtual, False
                        Caracter = "P" 'Transladar hacia adelante
                        Enviar_datos
                    End If
                    Text8.Text = CDbl(Objetos(0).grados)
                End If
            ElseIf xPos <= Por_minX * JoyStick1.xMax Then
              'Palanca abajo a la izq y Shifth
            Else
            'Palanca abajo a la der y Shifth
            End If
        Else
          'Zona centro
            If xPos >= Por_minX * JoyStick1.xMax And xPos <= Por_maxX * JoyStick1.xMax Then
              'Palanca al centro y Shifth
            ElseIf xPos <= Por_minX * JoyStick1.xMax Then
              'Palanca a la izquierda (Izquierda y Shifth)
            Else
              'Palanca a la der (Derecha y Shifth)
            End If
        End If
      ElseIf CurButton = 2 Then 'El botón 2 está apretado
        If yPos <= Por_minY * JoyStick1.yMax Then
          'Zona superior
            If xPos >= Por_minX * JoyStick1.xMax And xPos <= Por_maxX * JoyStick1.xMax Then
              'Palanca arriba
            ElseIf xPos <= Por_minX * JoyStick1.xMax Then
              'Palanca arriba a la izq
            Else
              'Palanca arriba a la der
            End If
```

```
        ElseIf yPos >= Por_maxY * JoyStick1.yMax Then
          'Zona inferior
            If xPos >= Por_minX * JoyStick1.xMax And xPos <= Por_maxX * JoyStick1.xMax Then
              'Palanca abajo
            ElseIf xPos <= Por_minX * JoyStick1.xMax Then
              'Palanca abajo a la izq
            Else
              'Palanca abajo a la der
            End If
        Else
          'Zona centro
            If xPos >= Por_minX * JoyStick1.xMax And xPos <= Por_maxX * JoyStick1.xMax Then
              'Palanca al centro
            ElseIf xPos <= Por_minX * JoyStick1.xMax Then
              'Palanca a la izquierda y Control
                If Llanta_delantera_virtual.grados > Grados_Rotacion Then
                  'Rotar en sentido antihorario
                    Rotar Llanta_delantera_virtual, -Grados_Rotacion, Llanta_delantera_virtual.Centro, False, True
                    If Comm1.Checked = True Or Comm2.Checked = True Then
                      Rotar Llanta_delantera_real, -Grados_Rotacion, Llanta_delantera_real.Centro, False, True
                    End If
                    grados = 90 - Llanta_delantera_virtual.grados
                    If grados > 0 Then
                      grados = 0.1 * grados
                    Else
                      grados = -0.1 * grados
                    End If
                    Caracter = "X" 'Rotar la llanta delantera en sentido antihorario
                    Enviar_datos
                    Text9.Text = CDbl(Llanta_delantera_virtual.grados)
                End If
            Else
              'Palanca a la der y Control
                If Llanta_delantera_virtual.grados <= (180 - Grados_Rotacion) Then
                  'Rotar en sentido horario
                    Rotar Llanta_delantera_virtual, Grados_Rotacion, Llanta_delantera_virtual.Centro, False, True
                    If Comm1.Checked = True Or Comm2.Checked = True Then
                      Rotar Llanta_delantera_real, Grados_Rotacion, Llanta_delantera_real.Centro, False, True
                    End If
                    grados = 90 - Llanta_delantera_virtual.grados
                    If grados > 0 Then
                      grados = 0.1 * grados
                    Else
                      grados = -0.1 * grados
                    End If
                    Caracter = "Y" 'Rotar la llanta delantera en sentido horario
                    Enviar_datos
                    Text9.Text = CDbl(Llanta_delantera_virtual.grados)
                End If

            End If
        End If
    End If ' Fin de botón 2
  End If
End Sub
```

4.0.7 Detección de colisiones

Después de analizar los métodos para detección de colisiones se encontró que, si bien en este caso se podría necesitar de un **método exacto** para simular realidad en la colisión, el sistema necesita mucho tiempo de procesamiento debido a sus caracteristica; sin embargo al contar con sensores en el robot se pudo percibir que una colisión se detecta a una distancia considerable respecto del objeto, por lo que, para simular ese comportamiento, se puede utilizar el método de cajas envolventes que necesita un menor grado de procesamiento. Si no se quiere simular ese comportamiento se encontró que la libreria D3DX8 tiene desarrollada una función que detecta la intersección de una raya y una malla(que es un conjunto de triángulos). Al final se optó por implantar un sistema de detección de colisiones incremental el cual en primera instancia detecta la intersección de los círculos que rodean a cada objeto y si existe una intersección, de acuerdo con el algoritmo elegido, se prueba la caja circundante o la intersección exacta entre los objetos. Las funciones que se encargan de esto son: **Colision, Choque** y **Choque2**. Su código es el siguiente:

```
Private Function Colision(key As KeyFrame, Indice As Long) As Boolean

 Dim i As Single
 Dim Distancia As Double
 Colision = False

 'Verifica que el objeto Se Encuentre dentro del área de trabajo permitida(PLANO)

 If (key.esquina_inf_izq_min.z > plano(2).z) Then
   If (key.esquina_inf_izq_min.X > plano(2).X) Then
     If (key.esquina_sup_der_max.z < plano(1).z) Then
       If (key.esquina_sup_der_max.X < plano(1).X) Then
         For i = 0 To Num_Objetos - 1 'Verifica que la distancai entre el objeto en cuestión
                   ' y los demas  sea suficiente para no Traslaparse
           If i <> Indice Then
             Distancia = Distancia_Objetos(key.Centro, Objetos(i).Centro)
             If Distancia <= (key.radio + Objetos(i).radio) Then 'Se encuentra en área de choque
               If exacto.Checked = True Then ' Es para la elección del algoritmo de colisión
                 Colision = Choque2(key, Objetos(i)) 'En caso de Traslape se aplica una función más precisa
               Else
                 Colision = Choque(key, Objetos(i)) 'En caso de Traslape se aplica una función más precisa
               End If ' Se termina la elección del algoritmo de colisión
               If Colision = True Then
                 Exit Function
               End If
             End If
           End If
         Next
         Exit Function
       End If
     End If
   End If
 End If
 Colision = True 'El objeto cae fuera del área de trabajo
 End Function
```

```
Private Function Choque(key As KeyFrame, ByRef key2 As KeyFrame) As Boolean

'Revisa si el objeto se ha colisionado con otro
Dim i As Long
Dim VertexList() As D3DVERTEX 'toma todoslos vertices de la malla
Dim vector_prueba As D3DVECTOR
Dim vector_direccion As D3DVECTOR ' Esta tomado del centro del objeto a la parte ultima del carrito
Dim Hit As Boolean ' Detecta si hubo una colisión o no
' Define el número de vertices de la malla
ReDim VertexList(key.Mesh.GetNumVertices) As D3DVERTEX
' constante de apoyo, obtiene los valores de lo vertices de la malla
i = D3DXMeshVertexBuffer8GetData(key.Mesh, 0, Len(VertexList(0)) * key.Mesh.GetNumVertices, 0, VertexList(0))
   If Not (i = D3D_OK) Then GoTo ErrOut 'Termina si no puede obtener los datos
      With vector_direccion ' define el vector de dirección
            .X = 0
            .Y = 1
            .z = 0
      End With
      If key2.esquina_inf_izq_min.X = key2.esquina_sup_der_max.X Or key2.esquina_inf_izq_min.Y =
key2.esquina_sup_der_max.Y Or key2.esquina_inf_izq_min.z = key2.esquina_sup_der_max.z Then
      'Entonces el objeto es plano en alguna dirección
         'La caja engrosa en una unidad
         key2.esquina_inf_izq_min.X = key2.esquina_inf_izq_min.X - 1
         key2.esquina_inf_izq_min.Y = key2.esquina_inf_izq_min.Y - 1
         key2.esquina_inf_izq_min.z = key2.esquina_inf_izq_min.z - 1
      End If
      For i = 0 To key.Mesh.GetNumVertices - 1  'revisa  cada uno de los vertices de la malla
         With vector_prueba
            .X = VertexList(i).X
            .Y = VertexList(i).Y
            .z = VertexList(i).z
         End With

         'Detección de las colisiones con sus parametros.

         Hit = D3DX.BoxBoundProbe(key2.esquina_inf_izq_min, key2.esquina_sup_der_max, vector_prueba,
vector_direccion)
         If Hit = True Then 'deteccion de la colisión
            Choque = True
            Exit Function
         End If
      Next i
         Choque = False
         Exit Function
ErrOut:
MsgBox "Ha ocurrido un error al obtener los vertices"
End
End Function


Private Function Choque2(key As KeyFrame, key2 As KeyFrame) As Boolean

'Revisa si el objeto se ha colisionado con otro
Dim i As Long
Dim VertexList() As D3DVERTEX 'toma todos los vertices de la malla
```

```
Dim inicio_ray As D3DVECTOR 'Punto donde inicia el ray
Dim ray_direccion As D3DVECTOR ' Vector unitario que define la dirección del ray
Dim Hit As Long, Face As Long, Count As Long    ' Detecta si hubo una colisión o no
Dim Distancia As Single, u As Single, v As Single 'Distancia parametrica de intersección del ray

' Define el número de vertices de la malla

ReDim VertexList(key.Mesh.GetNumVertices) As D3DVERTEX
' constante de apoyo, obtiene los valores de lo vertices de la malla
i = D3DXMeshVertexBuffer8GetData(key.Mesh, 0, Len(VertexList(0)) * key.Mesh.GetNumVertices, 0, VertexList(0))
  If Not (i = D3D_OK) Then GoTo ErrOut  'Termina si no puede obtener los datos
  'Una vez obtenidos los vértices de la malla se toman de 3 en 3 para tomar triangulos
    For i = 0 To key.Mesh.GetNumVertices - 2 'Barre todos los vértices de la malla
      inicio_ray = Construye_Vector(VertexList(i).X, VertexList(i).Y, VertexList(i).z)
      D3DXVec3Normalize ray_direccion, Construye_Vector(VertexList(i + 1).X - VertexList(i).X, VertexList(i + 1).Y -
VertexList(i).Y, VertexList(i + 1).z - VertexList(i).z)
      D3DX.Intersect key2.Mesh, inicio_ray, ray_direccion, Hit, Face, u, v, Distancia, Count
      If Hit = 1 Then
          If Distancia > 0 And Distancia < 1 Then 'Hubo una colisión
            Choque2 = True
            Exit Function
          End If
      End If
    Next
    'Si llega a este punto entonces no hay colision
    Choque2 = False
    Exit Function
ErrOut:
MsgBox "Ha ocurrido un error al obtener los vertices"
End
End Function
```

4.0.8 Interacción con el Robot Real

Comunicación serie

Para implementar la comunicación serie se optó por utilizar un control ActiveX que viene incluido como componente Visual Basic. Para poder establecer una comunicación a través de un puerto únicamente se necesita asegurar que el puerto en cuestión se encuentre cerrado, establecer los parámentros de comunicación y abrir el puerto para poder empezar a enviar y recibir datos. El tipo de comunicación que se puede establecer a través del puerto es por un ciclo cerrado o por eventos. En este caso se optó por la comunicación por eventos tratando de evitar la pérdida de datos. Los parámetros que rigen la comunicación en esta aplicación se definen en la función **Comm1_Click** que se muestra a continuación con el código que se refiere específicamente a ello en letras negras.

```
Private Sub Comm1_Click()
Colision_real = False
Listo = False
Codigo_Esperado = 118
Dim Temp As Double
If Comm1.Checked = True Then    'Se tenia comunicación establecida en Comm 1
```

```
    'Deshabilitar comunicacion
    comunicacion.Caption = "Conexión con Robot"
    Comm1.Checked = False
    Listo = True
    If MSComm1.PortOpen = True Then
       MSComm1.PortOpen = False
    End If
    With MSFlexGrid1
       .col = 1
       .Row = 3 'Posiciona la celda de dirección real
       .Text = ""
       .Row = 1
       .Text = ""
    End With

ElseIf Comm2.Checked = False Then   'No hay comunicación y se establece a través de Comm1

    'Establecer las condiciones de comunicación
    comunicacion.Caption = "Desconexión con Robot"
    Comm1.Checked = True
    'Inicialización de variables de puerto
    Caracter = "$"
    ' Configuración de enlace
    With MSComm1
       .CommPort = 1
       .Settings = "1200,N,8,1"
       .SThreshold = 0
       .RThreshold = 3 'Informar por cada 3 caracteres que llegan
       .InputLen = 0
       .InputMode = comInputModeBinary
    ' Abrir el puerto.
       .PortOpen = True
    End With
    'Iniciar Posicion de Carro Real
    Translacion Objetos(0).Centro.X - Carro_real.Centro.X, 0, Objetos(0).Centro.z - Carro_real.Centro.z,
Llanta_delantera_real, False
    Translacion Objetos(0).Centro.X - Carro_real.Centro.X, 0, Objetos(0).Centro.z - Carro_real.Centro.z, Carro_real, False
    Temp = Objetos(0).grados - Carro_real.grados
    Rotar Carro_real, Temp, Carro_real.Centro, False, True
    Rotar Llanta_delantera_real, Temp, Carro_real.Centro, False, False
    With MSFlexGrid1
       .col = 1
       .Row = 3 'Posiciona la celda de dirección real
       .Text = "(" & CSng(Carro_real.direccion.X) & "," & CSng(Carro_real.direccion.z) & ")"
       .Row = 1
       .Text = "(" & CInt(Carro_real.Centro.X) & "," & CInt(Carro_real.Centro.z) & ")"
    End With
    Timer2.Enabled = True
    Enviado = "C"
Else 'Se tiene una comunicación a través de Comm2 y se desea cambiar a Comm1
    If MSComm1.PortOpen = True Then 'Se cierra el puerto para cambiar a Comm1
       MSComm1.PortOpen = False
    End If
    'Establecer las condiciones de comunicación
    comunicacion.Caption = "Desconexión con Robot"
```

```
    Comm2.Checked = False
    Comm1.Checked = True
    'Inicialización de variables de puerto
    Caracter = "$"
    ' Configuración de enlace
    With MSComm1
      .CommPort = 1
      .Settings = "1200,N,8,1"
      .SThreshold = 0
      .RThreshold = 3 'Informar por cada 3 caracteres que llegan
      .InputLen = 0
      .InputMode = comInputModeBinary
    ' Abrir el puerto.
      .PortOpen = True
    End With
End If
Picture1.SetFocus 'Se pasa el foco a picture 1
End Sub
```

En la aplicación se cuenta con otra función llamada **Comm2_Click** que se encarga de la comunicación pero a través del puerto Comm2. Recordando que se debe seleccionar uno u otro ya que no se tiene contemplada la comunicación con el robot a través de los dos puertos al mismo tiempo.

En la función anterior se establece que se esperarán hasta 3 caracteres recibidos en el puerto para que la aplicación los lea e interprete. La función que se encarga de leer e interpretar los datos es **MSComm1_OnComm**. Básicamente toma la cadena de caracteres y determina el tipo de datos a los que se refiere tal cadena y toma las acciones necesarias sobre el estado de la comunicación o sobre la posición del robot real en la aplicación. El código de la función se muestra en las siguientes líneas:

```
Private Sub MSComm1_OnComm()
Dim Recibido As Variant  'Cadena de caracteres que se recibieron
Dim Valor_byte As Byte   'Valor expresado en byte
Dim posicion As Integer 'Valor de comparación posicional
Dim Encoder_derecha As Integer 'Almacena el valor de las vueltas de los encoders
Dim Encoder_izquierda As Integer
Dim Angulo_Norte_Actual As Double  'Grados respecto del norte magnético
Dim i As Integer 'Contador
Dim Codigo As String
Dim Codigo_Anterior As String
Select Case MSComm1 CommEvent

Case comEvReceive   ' Recibido nº RThreshold de caracteres.

  Timer2 Enabled = False 'Deshabilita la espera
  'Interpretar datos  recibidos
  Recibido = MSComm1 Input
  Codigo = CStr(Recibido(0))
  Text6.Text = CStr(Codigo) & CStr(CInt(Recibido(1))) & CStr(CInt(Recibido(2)))
  'VERIFICAR QUE EL CÓDIGO RECIBIDO SEA EL ESPERADO

    Timer2.Enabled = False 'Deshabilita la espera
```

```
    If Codigo = 117 Then 'Se requieren sensores u =117
      'Calcular el estado de los sensores
      Valor_byte = CByte(Recibido(1))
      posicion = 1
      MSFlexGrid1.Row = 4 'Inicia llenado de datos de sensores
      Colision_real = False
      For i = 0 To 5 'Barre el byte de sensores
        MSFlexGrid1.col = i + 1
        If Valor_byte And posicion Then 'El sensor en curso está activado
          MSFlexGrid1.CellBackColor = vbRed
          'Colision_real = True
        Else 'El sensor en curso está desactivado
          MSFlexGrid1.CellBackColor = vbWhite
        End If
        posicion = posicion * 2 'Cambia de posicion
      Next
      Caracter = "C"
      Enviar_datos
      Codigo_Esperado = 114
      Exit Sub
    ElseIf Codigo = 100 Then 'Interpretar los grados d = 100
      If Primera_vez = True Then  'Sólo cambia cuando inicia la transmisión
        Grados_Norte = 256 * (CInt(Recibido(1))) + (CInt(Recibido(2)))
        Angulo_Norte_Anterior = Grados_Norte
        Angulo_Norte_Actual = Grados_Norte
        Primera_vez = False
      Else
        Angulo_Norte_Actual = 256 * (CInt(Recibido(1))) + (CInt(Recibido(2)))
      End If
      Rotar Llanta_delantera_real, Angulo_Norte_Actual - Angulo_Norte_Anterior, Carro_real.Centro, False, True
      Rotar Carro_real, Angulo_Norte_Actual - Angulo_Norte_Anterior, Carro_real.Centro, False, True
      With MSFlexGrid1
        .col = 1
        .Row = 3 'Posiciona la celda de dirección real
        .Text = "(" & Mid(CStr(CDbl(Carro_real.direccion.X)), 1, 6) & "," & Mid(CStr(CDbl(Carro_real.direccion.z)), 1, 6) &
")"
        .col = 2
        .Text = CStr(Angulo_Norte_Actual)
      End With
      Angulo_Norte_Anterior = Angulo_Norte_Actual
      Caracter = "C"
      Enviar_datos
      Codigo_Esperado = 114 'recibido
      Exit Sub
    ElseIf Codigo = 101 Then 'Interpretar los encoders e=101
        Encoder_derecha = CInt(Recibido(1)) 'Obtiene la cantidad de desplazamiento
        Encoder_izquierda = CInt(Recibido(2))
        MSFlexGrid1.Row = 1 'Posiciona la celda de centro real
        MSFlexGrid1.col = 2
        MSFlexGrid1.Text = CStr(Encoder_derecha) & CStr(Encoder_izquierda)
        'Encoder derecha obtiene el desplazamiento final
        Encoder_derecha = (Encoder_derecha + Encoder_izquierda) / 2 'Obtiene el promedio de desplazamiento
        Encoder_derecha = Encoder_derecha * 4.3 ' * Centímetros por cada pulso acumulado
        If Secuencia = "P" Then
          'Interpretar retroceso
```

```
            Translacion Encoder_derecha * Carro_real.direccion.X, Encoder_derecha * Carro_real.direccion.Y,
Encoder_derecha * Carro_real.direccion.z, Carro_real, False
            Translacion Encoder_derecha * Carro_real.direccion.X, Encoder_derecha * Carro_real.direccion.Y,
Encoder_derecha * Carro_real.direccion.z, Llanta_delantera_real, False
          Else
            'Interpretar avance
            Translacion -Encoder_derecha * Carro_real.direccion.X, -Encoder_derecha * Carro_real.direccion.Y, -
Encoder_derecha * Carro_real.direccion.z, Carro_real, False
            Translacion -Encoder_derecha * Carro_real.direccion.X, -Encoder_derecha * Carro_real.direccion.Y, -
Encoder_derecha * Carro_real.direccion.z, Llanta_delantera_real, False
          End If
          'Actualizar la posición del carro real
          MSFlexGrid1.col = 1
          MSFlexGrid1.Text = "(" & CInt(Carro_real.Centro.X) & "," & CInt(Carro_real.Centro.z) & ")"
          Caracter = "C"
          Enviar_datos
          Codigo_Esperado = 114
      ElseIf Codigo = 118 Then 'Buffer Vacio v = 118
          Caracter = "C"
          Enviar_datos
          Codigo_Esperado = 114
          Listo = True 'Rehabilita la atención al teclado o joystick
      ElseIf Codigo = 114 Then ' Recibido r = 114
          Caracter = "C"
          Enviar_datos
          Codigo_Esperado = 118
      Else
         MSComm1.Output = "N"
         Timer2.Enabled = True
         Enviado = "N"
         Text7.Text = "N"
      End If
End Select

End Sub
```

Se desarrolló una función que se encarga de enviar al robot todas las peticiones de datos y movimientos; además deshabilita temporalmente la atención al teclado o joystick hasta que un dato proveniente del robot vuelva a habilitar los dispositivos de conducción. La función que se encarga de esto es **Enviar_datos**:

```
Public Sub Enviar_datos()
Dim Caracter2 As String
If MSComm1.PortOpen = True Then
   If Caracter <> "$" Then
      Caracter2 = Caracter 'Guarda el valor original
      If Caracter = "P" Or Caracter = "Q" Then

         Valor_Secuencia = Valor_Secuencia + 1
         If Llanta_delantera_virtual.grados <> 90 Then
            'La llanta está inclinada
            Valor_Secuencia = 0
            MSComm1.Output = "D"
            Enviado = "D"
            Listo = False
```

```
        Codigo_Esperado = 100
        Timer2.Enabled = True 'Inicia la cuenta
        'Loop de Retraso
        Do
          DoEvents
        Loop Until Listo
        MSComm1.Output = "E"
        Enviado = "E"
        Listo = False
        Codigo_Esperado = 101
        Timer2.Enabled = True
        'Loop de Retraso
        Do
          DoEvents
        Loop Until Listo
        Enviado = "E"
    ElseIf Secuencia = Caracter Then
      'Verificar si no se ha pasado del limite
        If Valor_Secuencia >= 3 Then
          'Pedir Encoders
            Valor_Secuencia = 0
            MSComm1.Output = "D"
            Enviado = "D"
            Listo = False
            Codigo_Esperado = 100
            Timer2.Enabled = True 'Inicia la cuenta
            'Loop de Retraso
            Do
            DoEvents
            Loop Until Listo
            MSComm1.Output = "E"
            Enviado = "E"
            Listo = False
            Codigo_Esperado = 101
            Timer2.Enabled = True
            'Loop de Retraso
            Do
              DoEvents
            Loop Until Listo
        End If
    Else
      'Secuencia diferente, pedir encoders
        Valor_Secuencia = 0
        Secuencia = Caracter
        MSComm1.Output = "D"
        Enviado = "D"
        Listo = False
        Codigo_Esperado = 100
        Timer2.Enabled = True 'Inicia la cuenta
        'Loop de Retraso
        Do
          DoEvents
        Loop Until Listo
        MSComm1.Output = "E"
        Enviado = "E"
        Listo = False
```

```
            Codigo_Esperado = 101
            Timer2.Enabled = True
            'Loop de Retraso
            Do
              DoEvents
            Loop Until Listo
        End If
    End If
    Codigo_Esperado = 114
    MSComm1.Output = Caracter2 'Se envia un dato y debe esperar hasta recibir datos
    Enviado = Caracter2
    Text7.Text = Caracter2
    Listo = False 'Deshabilita cualquier movimiento
    Timer2.Enabled = True 'Inicia la cuenta
  End If
End If
End Sub
```

Formato de datos

Para comunicarse con el robot fue necesario establecer un código que definiera las instrucciones que ejecutaría el robot, así como un código y orden en que se respondería a tales instrucciones. De esta manera, arbitrariamente se escogieron los siguientes códigos ASCII para que el robot ejecute las tareas asociadas:

ENVIO			
HEX	DEC	Caracter	Tarea
44	68	D	Brujula (Grados respecto al norte magnético)
45	69	E	Encoders (Avance en centimetros
50	80	P	Adelante
51	81	Q	Atrás
58	88	X	Rotar sentido Antihorario
59	89	Y	Rotar sentido Horario
43	67	C	Correcto(confirmación de datos)
4E	78	N	No Correcto(datos erróneos)

En retribución el robot regresa los datos en bloques de tres caracteres cuyo formato acordado es el siguiente:

RECEPCIÓN					
HEX	DEC	Caracter 1 (código)	Carácter 2	Carácter 3	Datos
64	100	d	Byte alto	Byte bajo	Grados de brújula
65	101	e	Encoder Der.	Encoder Izq.	Pulsos de distancia
72	114	r	0	0	Recibido(ok)
75	117	u	Byte sensores	0	Estado sensores
76	118	v	0	0	Buffer vacio (listo)

El byte de sensores llega en el siguiente formato:

X	X	Sensor6	Sensor5	Sensor4	Sensor3	Sensor2	Sensor1

Ajustes entre imagen real y virtual

Se está consciente que la simulación del movimiento del robot real y el movimiento del robot virtual diferirán en algún momento e inclusive en magnitudes posiblemente no aceptables debido a las imprecisiones mecánicas del robot, por lo que la precisión del ajuste en las pruebas realizadas determinan las relaciones avance/instrucción y grados/instrucción y con ello la calidad de la simulación. Es por lo anterior que necesitamos proporcionar al usuario una forma de llevar el robot virtual al lugar dónde se encuentra el robot real en el momento que las posiciones de ambos difieran en gran medida. El código que se encarga del ajuste se encuentra en la función **Form_Load** y se encuentra resaltado en negritas.

```
Private Sub Form_Load() 'Primer proceso a ejecutar
Me.WindowState = 2
Me.Show 'Asegura que la forma se muestre
bRunning = Inicializacion() 'Inicializa el dispositivo direct 3d y confirma si puede o no seguir
Incremento = 1.72
Do While bRunning 'Termina hasta que  brunning sea falso. Cuando se haga pick en la forma
'controles de ActivecX
  If GetTickCount() - FPS_LastCheck >= 1000 Then 'Calcula el tiempo entre cada frame para los cuadros por segundo
    FPS_Current = FPS_Count 'Actualiza las cuentas
    FPS_Count = 0
    FPS_LastCheck = GetTickCount()
  End If
  FPS_Count = FPS_Count + 1 'Incrementa la cuenta
  ' Actualiza los grados a rotar en cada llamada a teclado ó joystick
DoEvents 'Permite que Windows atienda eventos
  If Teclado Checked = True And cargando = False Then
    If (Comm1.Checked = False And Comm2.Checked = False) Or ((Comm1.Checked = True Or Comm2.Checked =
True) And (Listo = True)) Then  'No se tiene comunicación entonces se puede atender
      ' Ó se tiene comunicación pero está listo para enviar datos
      ' Ajusta los dos carros ( real y virtual) cuando se alejan más de sus radios
      If MSComm1.PortOpen = True And (Carro_real.radio + Objetos(0).radio) >=
Distancia_Objetos(Carro_real.Centro, Objetos(0).Centro) Then
        Dim Temp As Double
        Translacion Carro_real.Centro.X - Objetos(0).Centro.X, 0, Carro_real.Centro.z - Objetos(0).Centro.z,
Llanta_delantera_virtual, False
        Translacion Carro_real.Centro.X - Objetos(0).Centro.X, 0, Carro_real.Centro.z - Objetos(0).Centro.z,
Objetos(0), False
        Temp = Carro_real.grados - Objetos(0).grados
        Rotar Objetos(0), Temp, Carro_real.Centro, False, True
        Rotar Llanta_delantera_virtual, Temp, Objetos(0).Centro, False, False
      End If
      Atender_Teclado
    End If
    If Comm1.Checked = True Or Comm2.Checked = True Then
      If Avance >= Incremento * 3 Or Avance <= -Incremento * 3 Then ' se pide avance real
        'Mover este valor permite afinar el calculo de la ubicacion real del robot
        Avance = 0 'Reiniciar avance
```

```
            Caracter = "D"
            Enviar_datos
          End If
        End If
      End If
      If Esperando_Pick = True Then  'La vista debe ser 2D
        D3DXMatrixLookAtLH matView, vector_camara, vector_punto, Construye_Vector(0, 0, 1)
        D3DDevice.SetTransform D3DTS_VIEW, matView
      Else 'La vista debe ser 3D
        D3DXMatrixLookAtLH matView, vector_camara, vector_punto, Construye_Vector(0, 1, 0)
        D3DDevice.SetTransform D3DTS_VIEW, matView
      End If
      Render  'Render para picture 1
      D3DDevice.Present ByVal 0, ByVal 0, 0, ByVal 0

      D3DXMatrixLookAtLH matView, vector_camara2, vector_punto, Construye_Vector(0, 0, 1)
      D3DDevice.SetTransform D3DTS_VIEW, matView

      If Check1.Value = 1 Then
        Render
      Else
        D3DDevice.Clear 0, ByVal 0, D3DCLEAR_TARGET Or D3DCLEAR_ZBUFFER, 0, 1#, 0
      End If
      D3DDevice.Present ByVal 0, ByVal 0, Picture2.hWnd, ByVal 0
    Loop 'Termina el loop
    Unload Me

    End Sub
```

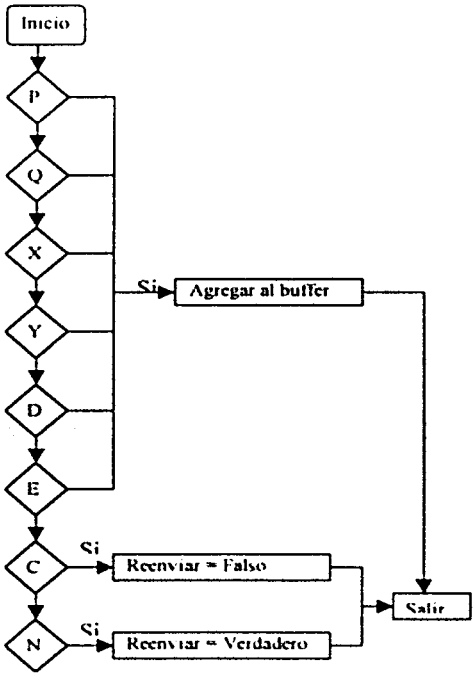
De lo que se encarga esta parte de la función es llevar las dos geometrías al mismo lugar y seguir con la simulación desde un estado en el que coinciden los dos carros.

5.0 REPORTE DE PRUEBAS DESARROLLADAS

5.0.1 Pruebas de ajuste de la llanta delantera

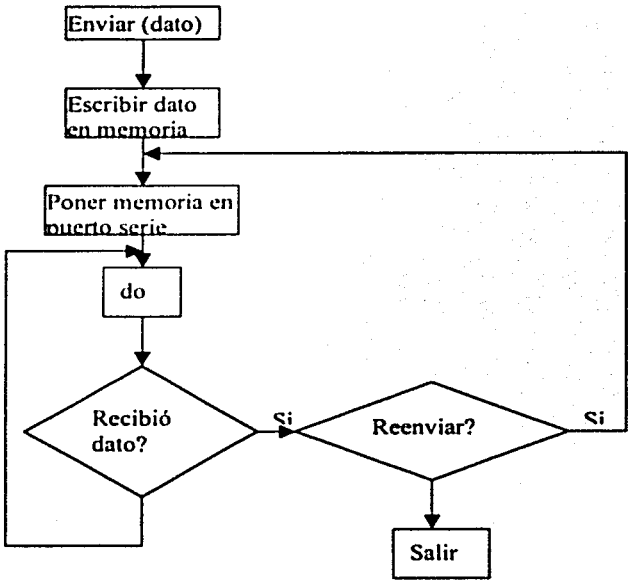
Para desarrollar estas pruebas fue necesario conocer la forma en que el robot recibe y manda las peticiones provenientes del software de simulación. A continuación se muestra un diagrama de flujo simplificado de cómo resuelve el robot las peticiones de movimiento.

Recepción de datos (Interrupción)



Figura(17). Cuando el robot recibe una petición de interrupción, atiende los datos provenientes del puerto serie. Clasifica el dato que le llegó y si es una instrucción de movimiento la agrega al buffer, si es una confirmación (C) ó no confirmación (N) actualiza el valor de la variable de reenvío.

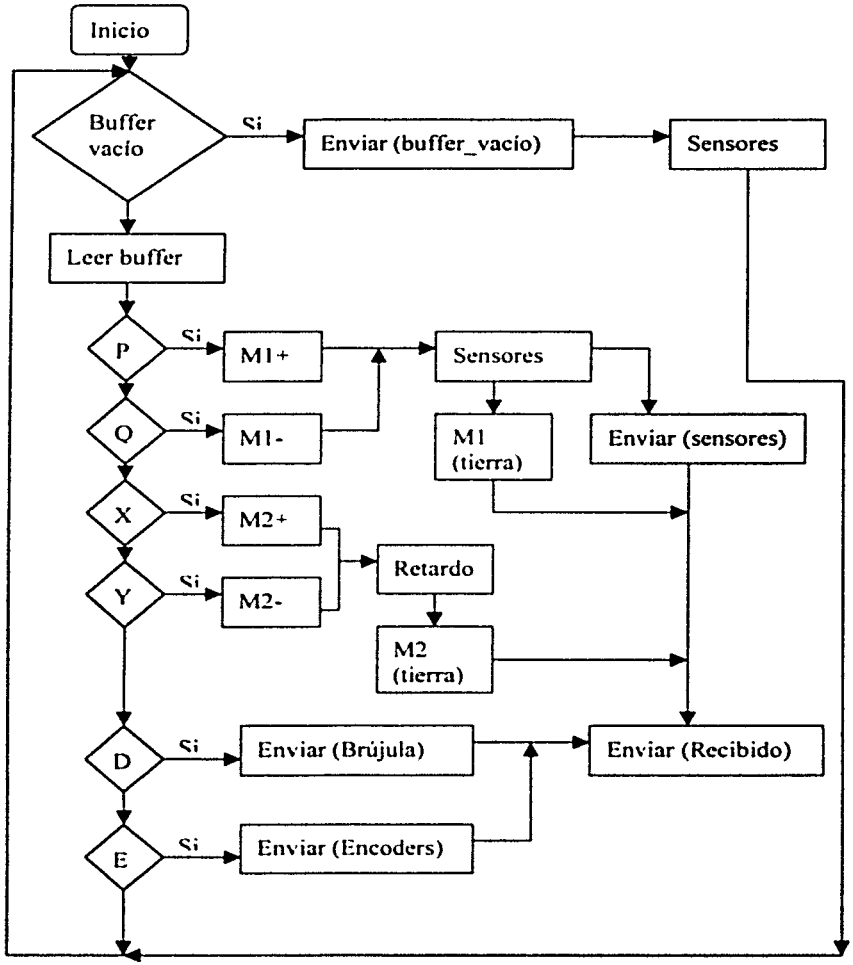
Enviar Datos por el puerto serie



Figura(18). La función de envío de datos espera la confirmación por parte del software para poder atender la siguiente petición de movimiento

Programa Principal

Figura(19). La rutina principal lee un buffer de recepción en cuyo caso se atienden las peticiones de movimiento. La rutina de sensores rompe el bucle si se encuentra un sensor encendido y envía el estado de ellos



Las primeras pruebas que se realizaron para calibrar la interacción con el robot se enfocaron en encontrar el avance en centímetros que se logra por cada instrucción de desplazamiento hacia delante que proporciona la computadora; con lo anterior se logró encontrar la distancia de avance que se debía simular en el programa. Se disponía de tres tiempos de activación del motor que proporcionaban avances en diferentes magnitudes y se eligió aquella instrucción que proporciona un menor tiempo de activación pero perceptible. Así, una vez elegido aquel movimiento menor pero perceptible, se hicieron pruebas de avance para determinar la relación avance/instrucción. En una distancia equivalente a una vuelta de la rueda delantera del robot(46.3 cm), se obtuvieron los siguientes resultados :

No. Prueba	Pulsos
1	11
2	11
3	12
4	12
5	12
6	12
7	12
8	13
9	14
10	14

Por tanto, el avance esperado por cada instrucción se simulará en 3.76cm.

La siguiente prueba que se realizó fue la prueba de retroceso cuyo método fue el mismo que para la prueba de avance. Los resultados para el retroceso son:

No. Prueba	Pulsos
1	14
2	14
3	15
4	15
5	15
6	15
7	15
8	16
9	16
10	16

Por tanto, el avance esperado por cada instrucción se espera en 3.066 cm. Con lo anterior se decidió simular predeterminadamente en 3.3 cm para avance y retroceso.

Después se realizó la prueba de ajuste de rotación de la llanta delantera para lo cual la prueba se dividió en rotación en sentido horario y antihorario. Tanto para la rotación en

sentido horario como antihorario se consideró un ángulo de 90° y se obtuvieron los siguientes resultados:

Horario

No. Prueba	Pulsos
1	47
2	48
3	47
4	48
5	47
6	47
7	47
8	47
9	47
10	47

Para este caso la relación ángulo/instrucción es de 1.91 (grados/instrucción) para el sentido horario.

Antihorario:

No. Prueba	Pulsos
1	47
2	46
3	46
4	47
5	47
6	45
7	46
8	46
9	46
10	46

Por tanto la relación ángulo/instrucción es de 1.95 (grados / instrucción en sentido antihorario.Así, se simulará predeterminadamente en 1.93 grados / instrucción.

Durante el desarrollo de las pruebas encontramos una serie de problemas y condiciones que limitan la constancia en los resultados obtenidos en cada evento los cuales se puntualizan a continuación:

- Tal es el caso del estado de la batería. Debido a que la pila gasta hasta 3 A/s por cada período individual (instrucción ejecutada en forma aislada), la potencia de la pila se puede ver disminuida a los pocos eventos realizados si esta no se encuentra cargada a toda su capacidad, en cuyo caso los avances empiezan a variar en una magnitud considerable, lo que provoca un desajuste total.

- El ajuste de los sensores es primordial para que los avances puedan calibrarse correctamente. Si los sensores tienen calibrada una distancia muy grande en su campo de acción, se obtendrán constantes interrupciones y por tanto cortes en el avance que se esté ejecutando. Así, se recomienda que los sensores tengan calibrada una distancia muy pequeña o desactivar los sensores, cuando se pretende ajustar la relación de distancia recorrida.

- La aceleración del robot, cuando recibe instrucciones continuas de avance o retroceso, provoca que los avances calibrados no correspondan con los reales. Esto se quiere resolver cambiando el tipo de motor que se encarga del movimiento adelante–atrás. El motor actual no cuenta con un reductor de velocidad y ello provoca la aceleración.

Así, cuando se quiera utilizar este software y se necesiten hacer las pruebas de ajuste, se deben tomar en cuenta estos factores que pueden cambiar constantemente.

CONCLUSIONES

Se concluye que se ha terminado una aplicación que permite manejar a distancia un robot que se mueve en ambientes estáticos, que pueden ser peligrosos o inaccesibles para el ser humano como pueden ser radiación, electricidad de alta potencia, temperaturas extremas, gases peligrosos, espacios pequeños, etcétera. A través de esta tele-operación se puede arriesgar sólo al robot dentro de esos ambientes hostiles logrando que el operador quede fuera de cualquier peligro pero aportando su destreza en la conducción y en toma de decisiones.

Las ventajas principales que se pueden encontrar en la utilización de este software se puntualizan a continuación:

- La posibilidad de tener una vista 2D y 3D logra que el usuario pueda orientarse mejor y planee los recorridos. La vista 2D da un mapa del ambiente y la vista 3D posiciona al usuario dentro del ambiente.
- La utilización de sensores se vuelve parte esencial en el éxito de la conducción cuando el robot se acerca demasiado a un objeto considerado o no dentro del ambiente haciendo que el apoyo visual se sustituya por los datos del estado de los sensores.
- La conducción puede ser elegida por el usuario a través del joystick o del teclado que en la mayoría de las PC's están disponibles.
- La posibilidad de elegir entre una cuadrícula o una textura como piso para comodidad del usuario.
- La animación de la llanta delantera, para que el usuario tenga una mejor idea de la dirección en la que se moverá el robot.
- El reloj se agrega para que el usuario pueda medir el tiempo de los recorridos y con ello pueda determinar si es viable mantener al robot o no dentro del ambiente.
- El cuadro de ajuste permite hacer las modificaciones que adecuarán los avances y rotaciones dependiendo del tipo de motor que se tenga.
- La existencia de dos métodos para la detección de colisiones permite al sistema adecuarse a la exactitud que se requiera. El método exacto proporciona una mejor simulación de colisión.

- El usuario puede escoger el software de graficación que prefiera para la creación de objetos a insertar, siempre y cuando este software pueda guardar en formato .3ds, además estos archivos deberán ser convertidos al formato .X utilizando el convertidor proporcionado (Conv3ds.exe) u otro.

- Puede ejecutarse en un sistema que utilice un mínimo de hardware y Windows 9X, sin sistemas especializados que reduzcan su disponibilidad y aumenten su costo.

RECOMENDACIONES

Estas recomendaciones son producto de las pruebas desarrolladas y de la experiencia que se tuvo con el manejo del robot:

- Para un mejor rendimiento de este software es necesario mejorar el sistema electromecánico del robot tal como un motor que consuma menor corriente evitando así que la batería se descargue en poco tiempo ya que esto puede provocar errores en la transmisión de datos.

- El método de inserción de los objetos debería permitir rotarlos dentro del ambiente además de poder seleccionar y borrar objetos que no se deseen.

- Agregar un aviso sonoro en el software cuando los sensores detecten un objeto cercano.

- Eliminar los encoders de las llantas traseras del robot e incluir un par de encoders de mayor precisión en la llanta delantera; un encoder debe medir la traslación de la llanta delantera y otro debe medir el ángulo de rotación de la misma.

- En caso que no se opte por lo anterior, cambiarlo por un sistema óptico proporcionado por un ratón con una lente adaptada ya que este es suficientemente preciso y confiable en sus mediciones.

- Hacer más precisa la mecánica de rotación mejorando el sistema de engranes de la llanta delantera.

- Cambiar el motor que se encarga de la traslación para eliminar la aceleración del robot cada vez que reciba instrucciones continuas de movimiento, esto con el objetivo de poder asegurar que el avance esperado por cada instrucción se cumpla independientemente de la secuencia de las peticiones de movimiento.

- El procesamiento del estado de los sensores debe realizarse fuera de la rutina de atención a las peticiones de movimiento ya que esto provoca imprecisiones en los movimientos debido a la interrupción constante de la rutina de sensado. Ya sea que la rutina de sensado la realice otro procesador ó se pueda interrumpir externamente al actual sólo en el momento que se detecte un obstáculo.

FUTURO

Se tiene contemplado que este sistema sea integrado con otros proyectos también desarrollados con Visual Basic en lo que respecta al manejo de robots con el objetivo de hacer un solo proyecto multifuncional que abarque la tele-operación, planeación de rutas, reconocimiento de color, etcétera. Las mejoras más próximas que se planean realizar son:

- Agregar la animación del brazo mecánico y poder manipularlo a través de los dispositivos de manejo actuales.

- Mejorar la toma de datos posicionales.

APÉNDICES

APÉNDICES

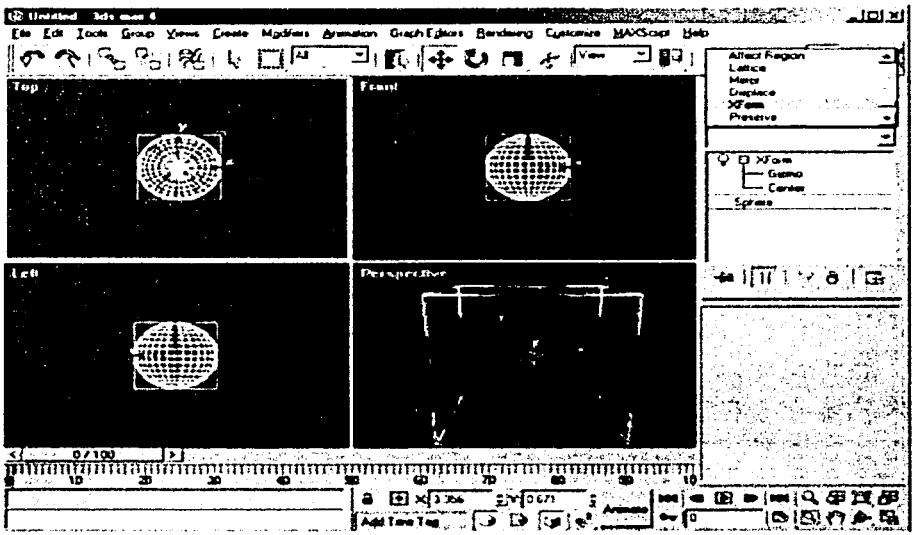
Apéndice 1

Requisitos de los archivos *.3DS

En este apartado vamos a ver varios consejos que serán de gran utilidad para la creación de objetos tridimensionales mediante 3D Studio Max(3DS max).

1. No debemos crear animaciones que incluyan explosiones de objetos, utilizaciones del *Bend y Taper*, y demás transformaciones que no sean del tipo rotación o translación, puesto que DirectX no las soporta en su formato de archivos.

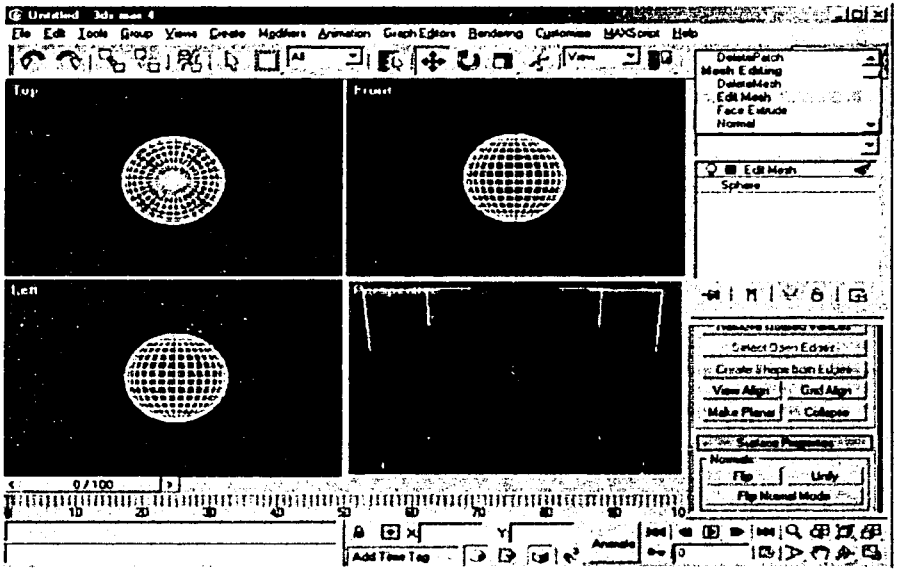
2. Cuando efectuemos escalado de objetos en el editor, debemos aplicar al *Gizmo del Mesh* un modificador Xform, pues de lo contrario, el resultado del objeto en el archivo X será impredecible aunque en 3DS max lo veamos correctamente.



3. Si importamos archivos 3DS de 3D Studio, debemos comprobar que al convertirlos a .x mediante Conv3ds, las normales de los objetos se han establecido correctamente, pues de lo contrario, habrá caras del objeto que se ven desde un punto de vista de

cámara y desaparecen desde otro punto de vista diferente. El problema radica en que los objetos pueden aparecer correctamente en 3DS max y luego aparecer erróneos al efectuar la conversión.

Para ajustar las normales de los objetos, debemos aplicar un modificador Edit Mesh en las caras que no sean visibles por los archivos .x y efectuar un Flip en las normales de dichas caras.



4. Siempre que exportemos el objeto resultante a formato 3DS también es conveniente guardar el trabajo como MAX, puesto que 3DS pierde todos los modificadores aplicados al objeto y no podríamos modificarlo en un futuro.

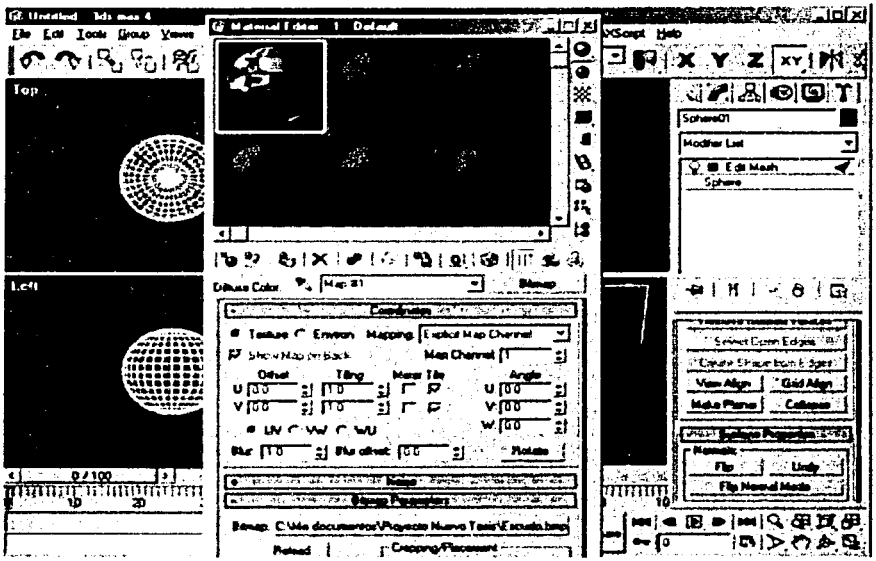
5. Cuando apliquemos texturas a un objeto, seleccionaremos en el editor de materiales un componente de tipo Bitmap para el mapa difuso. Seguidamente, cargaremos el gráfico necesitado y aplicaremos la textura con los siguientes parámetros activados:

   a. Coordenadas: Textura
   b. Mapeado: Explicito UVW 1.
   c. Parámetro: UV

En la siguiente imagen se pueden apreciar con detalle los parámetros utilizados para aplicar la textura:

Una vez definidos estos parámetros, sólo nos queda definir un modificador al Gizmo del objeto(UVW Map). Seleccionaremos el tipo de Wrapper para aplicar la textura y habremos acabado el trabajo con las texturas. Los Wrappers que podemos asignar son los siguientes:

- Planar
- Cilíndrico
- Esférico
- Shrink wrap
- Box
- Face



6. Al asignar enlaces entre objetos(links), no debemos usar Dummys puesto que no son soportados aún por el formato X.

7. En los links entre objetos. Debemos poner atención a los pivotes asignados, puesto que pueden modificarse al realizar la conversión al archivo .X y obtener resultados imprevistos.

Apéndice 2

Conv3ds para la importación de objetos tridimensionales.

Junto con el Software Development Kit(SDK) de DirectX se incluye esta funcional aplicación. **Conv3ds** se encarga de convertir modelos tridimensionales por 3D Studio (archivos .3ds) al formato nativo de DirectX (archivos .X).

Conv3ds se ejecuta desde la línea de comandos del MS-DOS. Podemos correr el programa sin ninguna opción, y se generará un archivo .X conteniendo una jerarquía de frames.

Por ejemplo:

Conv3ds Nombre_de_Archivo.3ds

Esta línea producirá un archivo llamado Nombre_de_Archivo.x después, sólo tenemos que utilizar el método insertar(este método es del programa desarrollado). A continuación se detallan los comandos que se utilizaron para la exportación de los archivos:

Conv3ds.exe –m

Utilizaremos la opción -m para crear archivos X que contengan simples objetos sin animaciones de ningún tipo.

Conv3ds –m File.3ds

Conv3ds.exe –s

La opción –s nos permite especificar un factor de escalado que se aplicará a todos los objetos convertidos del archivo 3DS. Por ejemplo, el siguiente comando hace todos los objetos tres veces más grandes.

Conv3ds –s3 File.3ds

Conv3ds.exe –t

La opción –t especifica que el archivo X resultante no contendrá información sobre las coordenadas de texturas.

Conv3ds.exe –N

La opción –N especifica que el archivo X resultante no contendrá información sobre las normales de los objetos. En este caso, cuando se carguen los archivos X, se generarán automáticamente las normales de cada uno de los objetos incluidos.

Conv3ds.exe –c

La opción –c especifica que el archivo X producido no debería tener información sobre las coordenadas de texturas. Por defecto, si utilizamos la opción –m, el objeto resultante contendrá(0,0) en las coordenadas de texturas UV en el caso de que el objeto 3ds no tuviera coordenadas para las texturas.

Conv3ds.exe –f

La opción –f especifica que el archivo X producido no contendrá información sobre las transformaciones en las matrices de los objetos.

Conv3ds.exe –z y Conv3ds.exe –Z

Las opciones –z y –Z permiten ajustar el valor del color de la cara alpha de todos los materiales referenciados por los objetos del archivo X. Por ejemplo, el siguiente comando obliga a conv3ds a incrementar el valor de 0.1 a todos los valores alpha por debajo de 0.2.

Conv3ds –z0.1 –Z0.2 File.3ds

Para substraer 0.2 de todos los valores alphas haremos lo siguiente:

Conv3ds –z"-0.2" –z1 File.3ds

Conv3ds.exe –o

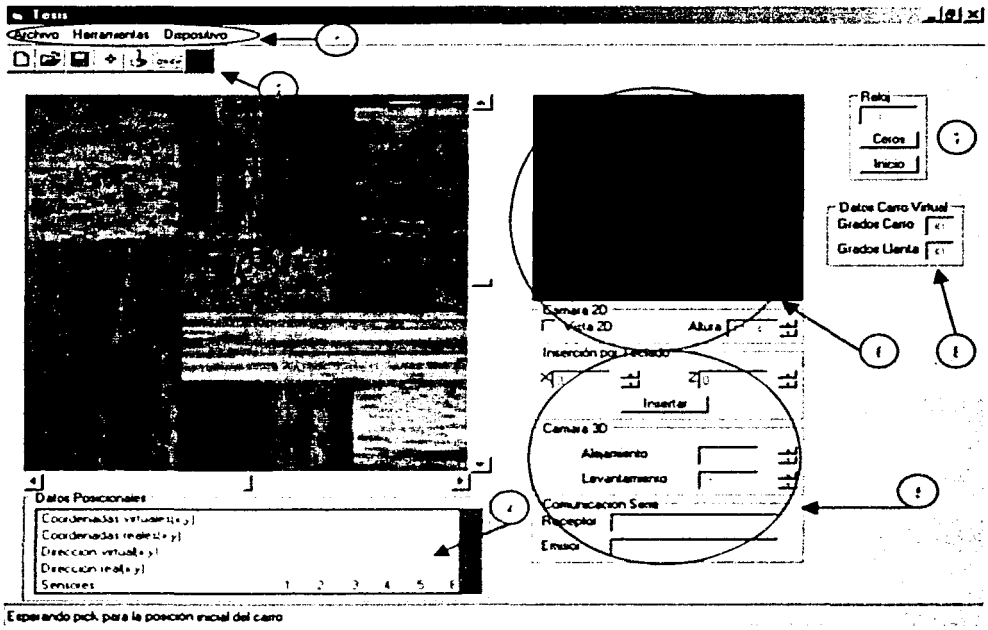
La opción –o permite especificar el nombre del archivo X que se producirá al realizar la conversión.

Todos los objetos producidos por **Trans3d** y **Lighwave** necesitan invertir el sentido de las caras de los objetos cuando son convertidos a archivos X de DirectX. Además no contienen información sobre las normales

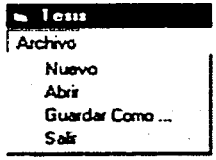
Como ayuda en la creación de archivos X, en el caso de no ver los objetos convertidos, podríamos utilizar la opción de escalado de objetos –s con un factor de 100. Esto incrementará la escala de los objetos resultantes en el archivo X.

MANUAL DE USUARIO

Este manual esta desarrollado con la finalidad de describir al usuario las tareas que pueden ser realizadas por este software y los pasos adecuados de las mismas para obtener un funcionamiento correcto. Comenzaremos con la explicación general de la interface la cual se muestra a continuación:



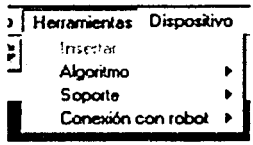
1.- En esta barra de menús se muestran tres opciones principales **Archivo**, **Herramientas** y **Dispositivo**, que tienen submenús. Para el menú **Archivo**, se tienen:



Cuyas funciones son:

Nuevo. Inicia el plano esperando que se especifique la posición inicial del robot.
Abrir. Carga un ambiente previamente diseñado en esta aplicación.
Guardar Como. Permite crear un archivo y una carpeta que contienen la información esencial que guarda el ambiente sobre el que se esta trabajando, para poder cargarlo en otra ocasión.
Salir. Finaliza con la aplicación.

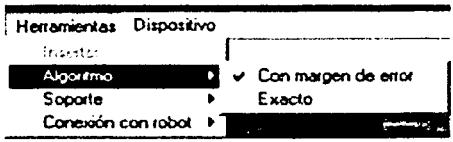
Para el menú **Herramientas** se tienen los siguientes submenús:



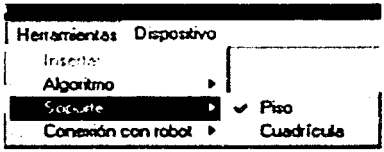
Donde sólo se tiene una tarea inmediata que es:

Insertar. Permite agregar un elemento geométrico previamente diseñado al ambiente sobre el que se trabaja. Esta función no está habilitada si no se ha dado la posición inicial del carro virtual.

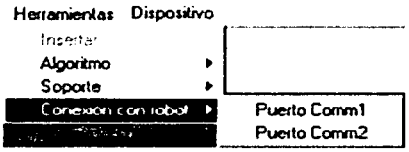
Con submenús se encuentran las demás opciones que para el caso de ***Algoritmo*** sólo permite seleccionar entre dos formas de trabajar la detección de colisiones que es a través del algoritmo *Con margen de error* que se refiere a la detección por cajas envolventes y el algoritmo *Exacto* que utiliza intersección de rayas y triangulos. El submenú es:



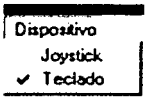
Para ***Soporte*** se tienen dos opciones que son *Piso* y *Cuadrícula*. La primera opción muestra una textura que soporta el ambiente y la segunda muestra sólo una cuadrícula como soporte. El submenú correspondiente se muestra a continuación:



Los submenús de ***Conexión con robot*** pueden ser para elegir el puerto serie por el que se desea comunicar el robot, las opciones son *Puerto Comm1 y Puerto Comm2* para el puerto serie 1 y 2 respectivamente. Una vez seleccionado algún puerto El texto Conexión con robot se cambia por Desconexión con Robot y tiene la funcionalidad de que ahora se busca cortar la comunicación con el robot ó intercambiar el puerto por el que se está comunicando. Para desconectar el robot se debe hacer Click sobre la opción que se encuentre activada (con una palomita) ó hacer Click sobre la opción del puerto que se desea. El submenú para este caso es:



Por último se tiene en la barra el menú **Dispositivo** el cuál permite escoger entre uno y otro medio de conducción del carrito el cuál puede ser ***Teclado*** o ***Joystick***. El submenú es:



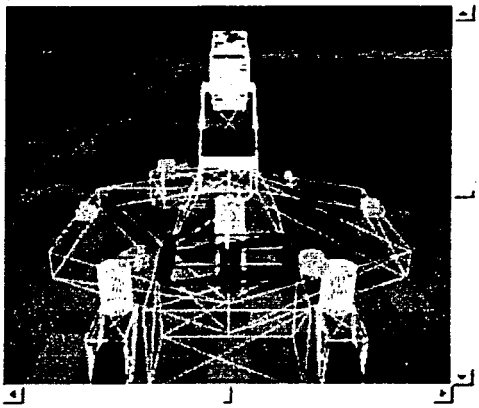
2.- El área de botones es para que el usuario tenga un acceso rápido a las aplicaciones como lo son:

De izquierda a derecha las funciones de cada botón son análogas a las descritas para la barra de menús:

a) **Nuevo**
b) **Abrir**
c) **Guardar Como**
d) **Insertar**
e) **Joystick**
f) **Teclado**
g) **Soporte**

3.- Esta ventana muestra la vista superior del ambiente cuando se va a especificar la posición inicial del robot virtual ó cuando se va a insertar un nuevo objeto al ambiente (esta inserción puede ser por mouse o teclado). En otro caso, en esta ventana se puede ver el ambiente tridimensional desde un punto cercano al robot virtual.

Cuando se eligió insertar un nuevo objeto al ambiente, se debe especificar el punto de inserción del objeto, en está ventana se puede dar un Click para especificar esa posición de inserción.



4.- **Los Datos Posicionales** consisten en una tabla que muestra información como: *Coordenadas virtuales(x,y)* y *Coordenadas reales(x,y)* que representan el centro de cada uno de los robots, *Dirección virtual(x,y)* y *Dirección real(x,y)* donde X representa el seno e Y representa el coseno director del vector director del robot en el ambiente. Inicialmente la dirección en cada robot se define orientada al eje positivo Y. El renglón de *Sensores* muestra el estado de los sensores, la celda se muestra en color rojo cuando se activa alguno de ellos.

Datos Posicionales

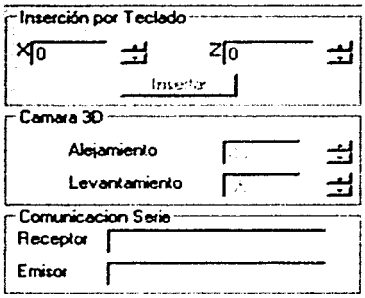
Coordenadas virtuales(x,y)	(12,13)						
Coordenadas reales(x,y)							
Dirección virtual(x,y)	(0,1)						
Dirección real(x,y)							
Sensores		1	2	3	4	5	6

5.- **La Inserción por Teclado** proporciona dos campos donde se pueden especificar de manera exacta las coordenadas donde se desea tener la posición inicial del robot o del nuevo objeto a insertar en el ambiente. Cabe señalar que las coordenadas que se pueden especificar deben encontrarse en el rango de -1000 a 1000 que son las coordenadas que ocupa el plano tanto horizontal como vertical. El centro del sistema de coordenadas se encuentra al centro del plano.

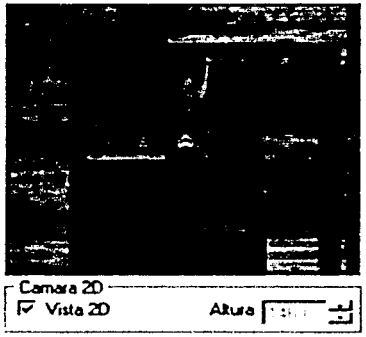
La **Cámara 3D** permite modificar la posición desde la cual el usuario ve el ambiente sin dejar de ver al eje central del robot, la cámara solo se mueve en su misma dirección.

El apartado de **Comunicación Serie** únicamente permite al usuario verificar que la comunicación se establece correctamente mostrando los datos que *Emite* y *Recibe* el puerto serie.

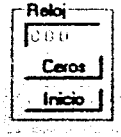
En la siguiente figura observaremos todos los apartados:



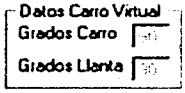
6.- **La parte de Cámara 2D** sirve como un apoyo para que el usuario pueda orientarse mejor en el ambiente. La *Altura* es modificable y sólo cambia el punto desde el cual se ve.



7.- **La parte del Reloj** sólo es un apoyo para saber el tiempo de recorrido, el usuario puede activarlo en el momento que desee.



8.- El apartado para **Datos Carro Virtual** es apoyo del usuario para saber las posiciones de la llanta y para saber cuantos grados tenemos que girar para un nuevo avance. Además nos sirve para iniciar fisicamente el robot real en cuanto a la inclinación de la llanta delantera y la dirección en que se debe encontrar.



Ejemplificando el uso de la interface a continuación se describen los dos modos de uso esperado del software:

Modo Simulación sin Conexión

En este caso se espera que el usuario únicamente utilice el software sin manejar a través del puerto serie el robot. Así, para utilizar de esta forma el programa se deben seguir los siguientes pasos.

1) Iniciar el programa.
2) Especificar la posición inicial del robot virtual ya sea a través de un Click en la ventana de vista 3D ó escribiendo las coordenadas en los espacios para inserción por teclado.
3) Elegir el dispositivo de conducción que se prefiera ya sea Joystick o Teclado. El dispositivo predeterminado es el teclado.
4) Habilitar si se prefiere la vista bidimensional y ajustar la altura de la cámara.
5) Ajustar a modo el alejamiento y levantamiento de la cámara 3D.
6) Mover el robot virtual. Para moverlo se deben cumplir las siguientes condiciones:

   - Hacia delante se debe presionar Shift + Flecha Arriba, para el teclado ó el Botón 1 + Palanca Arriba para el caso del joystick.
   - Hacia atrás se debe presionar Shift + Flecha Abajo, para el teclado ó el Botón 1 + Palanca Abajo para el caso del joystick.
   - Para rotar la llanta delantera en sentido antihorario se debe presionar Ctrl. + Flecha Izquierda ó Botón2 + Flecha Izquierda para el teclado y joysteik respectivamente.
   - Para rotar la llanta delantera en sentido horario se debe presionar Ctrl. + Flecha Derecha ó Botón2 + Flecha Derecha para el teclado y joystick respectivamente.

Opcionalmente se puede, en cualquier momento elegir la inserción de un nuevo objeto al ambiente ó elegir cargar un ambiente previamente diseñado. Si se elige cargar un ambiente

se debe comenzar desde el paso 2. En caso de una inserción, se debe especificar la posición de inserción del nuevo objeto.

Modo Simulación con Conexión

En este caso se espera que el usuario únicamente utilice el software sin manejar a través del puerto serie el robot. Así, para utilizar de esta forma el programa se deben seguir los siguientes pasos.

1) Iniciar el programa.
2) Especificar la posición inicial del robot virtual ya sea a través de un Click en la ventana de vista 3D ó escribiendo las coordenadas en los espacios para inserción por teclado.
3) Elegir el dispositivo de conducción que se prefiera ya sea Joystick o Teclado. El dispositivo predeterminado es el teclado.
4) Habilitar si se prefiere la vista bidimensional y ajustar la altura de la cámara.
5) Ajustar a modo el alejamiento y levantamiento de la cámara 3D.
6) Establecer conexión a través de algún puerto serie disponible. Se debe asegurar que el robot se encuentre previamente encendido así como en la posición y dirección lo más cercana posible a la que se está simulando en la aplicación.
7) Mover el robot virtual. Para moverlo se deben cumplir las siguientes condiciones:

   - Hacia delante se debe presionar Shift + Flecha Arriba, para el teclado ó el Botón 1 + Palanca Arriba para el caso del joystick.
   - Hacia atrás se debe presionar Shift + Flecha Abajo, para el teclado ó el Botón 1 + Palanca Abajo para el caso del joystick.
   - Para rotar la llanta delantera en sentido antihorario se debe presionar Ctrl. + Flecha Izquierda ó Botón2 + Flecha Izquierda para el teclado y joystcik respectivamente.
   - Para rotar la llanta delantera en sentido horario se debe presionar Ctrl. + Flecha Derecha ó Botón2 + Flecha Derecha para el teclado y joystick respectivamente.

Se recomienda que antes de establecer una conexión se cargue previamente el ambiente prediseñado o se hayan insertado los objetos que lo conformen para asegurar que el robot virtual y real comenzaran en una posición valida. Si se ha cometido un error al momento de establecer la comunicación por el puerto la aplicación se parará es decir si se selecciono el puerto Comm1 por ejemplo y el robot aún no estaba encendido, la simulación estará esperando que el robot envíe el dato de listo, por ello la aplicación se para. Esto se soluciona desconectando la comunicación.

Opcionalmente se puede elegir la inserción de un nuevo objeto al ambiente ó elegir cargar un ambiente previamente diseñado. Si se elige cargar un ambiente se debe comenzar desde el paso 2. En caso de una inserción, se debe especificar la posición de inserción del nuevo objeto.

Además se recomienda que antes de realizar la conexión con el robot se verifique la distancia de alcance de los sensores, ya que a mayor distancia el movimiento del robot es más lento debido a las constantes interrupciones durante el movimiento.

Cuando el robot esta a punto de tener una colisión su avance es mucho más lento debido a que está dentro del rango de colisión y el avance se ve constantemente interrumpido, es por ello que su avance es mínimo.

Es recomendable que la vista bidimensional sólo sea activada cuando el usuario necesite tener una ubicación general de su entorno, ya que cuando esta se activa el consumo de recursos es mayor y la simulación es más lenta así como el movimiento del robot si es que se encuentra conectado.

Requerimientos del sistema

Para que el software pueda ser utilizado correctamente las características mínimas del equipo son:

- Sistema Operativo Windows 9X
- Tarjeta de video de 2M
- Resolución de pantalla 800 X 600 a 256 colores
- Memoria Ram de 32 MB
- Versión DirectX 8.1
- Procesador Pentium a 100 MHz ó equivalente

Cabe señalar que que con estas características se obtiene un rendimiento pobre pero funcional.

Sin embargo se recomienda que para obtener un rendimiento óptimo el equipo tenga las siguientes características:

- Sistema Operativo Windows Me
- Tarjeta Aceleradora de video AGP de 32Mb
- Resolución de pantalla 1024 X 768 en Color de 16 bits
- Memoria Ram de 128MB
- Versión DirectX 8.1
- Procesador Pentium III a 700 MHz ó equivalente

Nota: Estos requerimientos se basan únicamente en la experiencia obtenida al utilizar este software en diferentes equipos.

BIBLIOGRAFÍA GENERAL

Herramienta de desarrollo de videojuegos:

Páginas de Internet:
General:
http://www.divsite.net/phorum/index.php?f=2
http://www.divgame.com/es/index.html
http://www.jdejuegos.com
http://linux.cem.itesm.mx/xjuegos/de/donde_empezar.htlm
http://linux.cem.itesm.mx/xjuegos/de/donde_empezar00.htlm
http://linux.cem.itesm.mx/xjuegos/de/donde_empezar01.htlm
http://linux.cem.itesm.mx/xjuegos/de/donde_empezar02.htlm
http://www.galeon.com/diynet/ayuda/ayuda.html
Fénix:
http://fenix.sourceforge.net/bdocs/fenix.html

Lenguajes de programación más utilizados en la programación de ambientes gráficos:

Páginas de Internet:

Ensamblador
C
C++
Java
Visual Basic

Interfaces para programación de aplicaciones más utilizadas en ambientes gráficos:

Páginas de Internet:

http://linux.cem.itesm.mx/xjuegos/
Allegro
http://www.allegro.cc
OpenGL
www.opengl.org
http://is6.pacific.net.hk/~edx
http://www.chez.com/scribe/en/opengl
www.opengl.org
http://www.ligthwave3d.com/news/press/DirectX8.3-22-01.html
http://www.libsdl.org/opengl/index.php
SDL
http://linux.oreillynet.com/pub/a/linux/2001/09/21/sdl.html?page=1
http://linux.oreillynet.com/pub/a/linux/2001/09/21/sdl.html?page=2
http://linux.oreillynet.com/pub/a/linux/2001/09/21/sdl.html?page=3
http://www.libsdl.org/languages.php

DirectX
http://www.microsoft.com/directx/
http://www.mvps.org/vbdx/
http://www.microsoft.com/windows/directx/default.asp
http://www.exhedra.com/DirectX4VB/Tutorials/DirectX8/GR_Lesson04.asp
http://www.exhedra.com/DirectX4VB/TUT_DX8Start.asp
http://msdn.microsoft.com/library/en-us/dx8_vb/directx_vb/Intro/DX8WhatsNew.asp?fram
http://www.planet-source-code.com/xq/ASP/txtCodeId.2577/lngWId.3/qx/vb/scripts/Show
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dx8_vb/directx_vb/Intro/DX8WhatsNew.asp?fram
http://msdn.microsoft.com/library/en-us/dndxgen/html/directxovrvw.asp?frame=true
http://www.microsoft.com/presspass/press/2000/Nov00/DirectXSupportPR.asp

Libros:

Constantino Sánchez Ballesteros. Ed. Alfaomega, "Programación multimedia avanzada con DirectX", 1992

Casey Larijani,Ed. McGraw-Hill , "Realidad Virtual, L". Casey Larijani, Ed. McGraw-Hill, 1994, pág 31

Software de graficación:

Páginas de Internet:
LigthWave 3D:
http://www.3dyanimacion.com/modules.php?name=Content&pa=showpage&pid=1
http://www.3dyanimacion.com/modules.php?name=Sections&op=listarticles&secid=1
Softimage 3D:
http://www.softimage.com
http://www.avid.es/novedades/notas/soft-yoda.html
3D Studio max
http://www.3dyanimacion.com/modules.php?name=Sections&op=listarticles&secid=3
http://www.unav.es/cti/manuales/3DstudioMax/indice.html
General:
http://www.interserver.com.ar/host/osiris/3d/links.htm
http://usuarios.lycos.es/max3d/

ActiveX:

Páginas de Internet:

http://www.terra.com/informatica/que-es/activex.cfm
http://www.diatel.upm.es/~adasilva/tweb/trabajosTecnologiaWeb/ActiveX%20-%20Luis%20Manuel%20Cruza%20Roldan/activex/el_componente_activex.htm
http://www.diatel.upm.es/~adasilva/tweb/trabajosTecnologiaWeb/ActiveX%20-%20Luis%20Manuel%20Cruza%20Roldan/introduccion/ole.htm

http://www.diatel.upm.es/~adasilva/tweb/trabajosTecnologiaWeb/ActiveX%20-%20Luis%20Manuel%20Cruza%20Roldan/activex/el_contenedor_activex.htm

Libros:

Date Becker ,Ed. Marcombo , "El gran Libro de 3D Studio Max". 1997

Transformaciones geométricas:

http://www.info-ab.uclm.es/asignaturas/42538

Conceptos de Graficación para desarrollo de aplicaciones:

Páginas de Internet:

http://www.esc.ucm.es/csc/Usuarios/Formacion/Conline/Intro3D/chapter1.0.html#Toc-Ch

Realidad Virtual:

Libros:

Casey Larijani,Ed. McGraw-Hill , "Realidad Virtual, L". Casey Larijani, Ed. McGraw-Hill, 1994

Dimitris N. Chorafas/Heinrich Steinmann, Ed. Prentice-Hall Hispanoamericana , "Realidad Virtual(Aplicaciones Prácticas en los Negocios y la Industria", 1995

Páginas de Internet:

http://www.rvltda.com/Info_RV.htm
http://www.utp.ac.pa/seccion/topicos/realidad/cap3.htm
http://www.utp.ac.pa/seccion/topicos/realidad/cap2.htm
http://highland.dit.upm.es:8000/echeva/docs/visvrml.html
http://ftp.etsimo.uniovi.es/links/vr.html
http://www.activamente.com.mx/vrml
http://www.unitec.edu.co/biblioteca/rv_principal.html
http://www.utp.ac.pa/seccion/topicos/realidad/index.html
http://www.utp.ac.pa/seccion/topicos/realidad/cap1.htm
http://www.utp.ac.pa/seccion/topicos/realidad/cap212.htm
http://www.utp.ac.pa/seccion/topicos/realidad/cap213.htm
http://www.utp.ac.pa/seccion/topicos/realidad/cap2.htm
http://www.utp.ac.pa/seccion/topicos/realidad/cap3.htm
http://www.utp.ac.pa/seccion/topicos/realidad/cap4.htm
http://www.activamente.com.mx/vrml
WWW.VIRTUAL-TECH.ES
WWW.USERS.INYCOM.ES/~AGONZALEZ/INDEX.HTM
WWW.NOVA3D.COM

WWW.NICOSIO.COM
WWW.CINE.MEGAMEGA.NET

Tiempo Real:

Algoritmos para la detección de colisiones:

Libros:

Stefan Gotts. Collision Queries using Oriented Bounding Boxes.
Tesis de la Universidad de Carolina del Norte U.S. 2000. 192 p.

Thomas Möller et. al.. Fast, minimum storage Ray/Triangle Intersection.
Universidad Tecnológica de Chalmers.

Páginas de Internet:

http://www.acm.org/jgt/papers/MollerTrumbore97/
http://www.programatica.com/
ftp://rtfm.mit.edu/pub/faqs/graphics/algorithms-faq

Puerto Serie:

Manejo de DirectX con los diferentes softwares:

http://www.ligthwave3d.com/news/press/DirectX8.3-22-01.html
http://www.icarusindie.com/tricks/directX8/extras/direct3d/tools/3dsmax3/
http://www.mvps.org/vbdx/tutorials/index.html
http://www.mvps.org/vbdx/tutorials/compilex/index.html
http://www.mvps.org/vbdx/tutorials/exportx/index.html

Qué es la teleoperación:
http://www1.ceit.es/asignaturas/control1/Proyectos/Teleop2D/teleoperacion.htm