VNIVERSIDAD NACIONAL
AVFNOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

"NAVEGACIÓN DIRIGIDA DE AGENTES VIRTUALES EN REALIDAD AUMENTADA"

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERÍA (COMPUTACIÓN)

P R E S E N T A:

MARCO ANTONIO SALAZAR FRANCO

DIRECTOR DE TESIS:
MAT. ANA LUISA SOLÍS GONZÁLEZ COSÍO

México, D.F. **2005**

A ti que me diste tu vida, tu amor y tu espacio.
A ti que cargaste en tu vientre dolor y cansancio.
A ti que peleaste con uñas y dientes.
Valiente en tu casa y en cualquier lugar.
A ti rosa fresca de abril, a ti mi fiel querubín.
A ti te dedico mis versos, mi ser, mis victorias.

A ti mis respetos señora, señora, señora.

A Dios, por permitirme vivir y lograr esta etapa en mi vida...

A mis padres y mis hermanos que me han apoyado y a quienes dedico este logro personal, haciéndolo un logro familiar...

A mis compañeros de generación, por vivir juntos esta experiencia y crecimiento profesional...

A la Mat. Ana Luisa Solís, por haber aceptado dirigir mi trabajo de tesis y por todos sus consejos y apoyo para la realización de este trabajo...

A los doctores Boris Escalante Ramírez, Jesús Savage Carmona, Fernando Gamboa Rodríguez y Ma. Elena Martínez Pérez por sus enseñanzas en la maestría, y por sus comentarios y aportaciones hechas para el enriquecimiento de este trabajo...

A todo el personal del IIMAS, particularmente a Lulú, Amalia, Violeta y Juanita, por su apoyo en todo momento durante toda mi estancia y para lograr esta meta...

A la Universidad Nacional Autónoma de México, que me dio un espacio para desarrollarme académica y profesionalmente y que siempre tendrá un lugar muy importante en mi formación...

A todos ustedes, mis más sinceras

GRACIAS

Índice

INTRODUCCIÓN

Recientemente la industria de los videojuegos ha tenido un gran auge. Los gráficos en ellos se han vuelto cada vez más sofisticados y combinándolos con una buena historia y buenos efectos de sonido atraen a más y más usuarios al apreciar su acercamiento con la realidad. Por mencionar solo algunos tenemos los clásicos de Final Fantasy que si bien no son apegados a la realidad en cuanto a una historia si lo son en cuanto al detalle de los personajes, y presentan gráficos sorprendentes que mejoran en cada entrega. Lo mismo ocurre con Resident Evil que se ha convertido en uno de los clásicos de terror y cuyas gráficas tratan de representar fielmente los modelos humanos[1].

Es precisamente en este punto donde el usuario siente el deseo de involucrarse más en esas aventuras gráficas, y es precisamente donde podemos incursionar en aras del mejoramiento de los métodos para la educación. Antes de continuar es necesario tener una idea sobre los conceptos *navegación dirigida*, *caracteres animados* y *realidad aumentada*. Los definiremos brevemente a fin de que se pueda tener una referencia de estos conceptos, mismos que serán abordados con mas detalle en sus respectivas áreas en este trabajo. Así mismo se recomienda al lector referirse al glosario que se encuentra al final del presente trabajo a fin de tener claros diversos términos que aparecen a lo largo del texto.

Brevemente entendamos navegación dirigida como la forma en que es posible que un carácter animado se mueva por un entorno ya sea real o virtual por un camino libre de obstáculos. Un carácter (personaje) animado es una representación tridimensional de una persona real, considerando en la medida de lo posible no sólo la apariencia física sino que además se trata de que el personaje tenga características tales como comportamiento y conocimiento de su entorno.
La realidad aumentada es un área de la realidad virtual cuyo objetivo es aumentar la información que posea un usuario, lo cual se logra sobreponiendo al mundo real (escena real) una escena generada por computadora (escena virtual).

Partiendo de esto imaginemos que mediante ciertos dispositivos tales como una cámara conectada a una PC y una pantalla o si es posible con unos lentes HMD podemos hacer que, con sólo observar por ejemplo una computadora con múltiples características, obtengamos en una pantalla o en los mismos lentes información acerca del funcionamiento de la computadora, lo cual puede aplicarse también a un equipo industrial donde se requiere aprender su funcionamiento, lo mismo puede pasar al estar frente a un monumento y saber cuando fue realizado, por qué, por quién, etc.

Este agente podrá navegar en el ambiente y el usuario podrá verlo y además tendrá interacción con él por medio de la voz, lo que es tema de otro trabajo de tesis. Todo esto formará un sistema de realidad aumentaba que permita una

[1] Estos juegos son propiedad de sus respectivos creadores y solo se mencionan a manera de ejemplo. http://www.capcom.com.jp

colaboración entre el usuario y el agente[12]. Ejemplos de navegación pueden verse en los videojuegos mencionados anteriormente, donde en usuario controla al personaje para que siga el camino que el usuario desea. En esos casos es el usuario el que determina la ubicación del personaje y su comportamiento de acuerdo a las reglas del juego; nosotros pretendemos que la relación entre el usuario y el agente sea de cooperación más que de subordinación

Actualmente ya hay algunas investigaciones que han derivado en algunos métodos para implementar esta navegación. Principalmente el trabajo que revisamos es el realizado por James J. Kuffner Jr.[1], quien trabajó en un método de navegación de meta dirigida para caracteres animados usando una planeación en tiempo real. Brevemente, este método lo que hace es dividir la tarea en dos partes: una donde se verifica que exista un camino desde un punto origen a un destino, y otra que se encarga de seguir esta ruta encontrada, controlando tanto el movimiento del personaje como considerando los obstáculos móviles que pudieran presentarse en su camino, como podría ser otro agente que habite el mismo ambiente. Esta navegación está planeada para entornos virtuales, no para un sistema de realidad aumentada como pretendemos nosotros.

La navegación implica un algoritmo que determine el camino a seguir por el carácter, si es que existe. El elaborar un escenario de realidad aumentada requiere determinar la forma en que el carácter virtual "existirá" en el mundo real. Como se describirá en el trabajo de tesis, ya hay quienes han hecho aportaciones a cada una de estas áreas, con sus características y limitantes. Por ejemplo, el algoritmo empleado por Kuffner[1] para la búsqueda de caminos es uno de los dos principales que existen. El A-Star (A*)[3,11], mismo al que le hicieron algunas modificaciones para sus propósitos; nosotros implementaremos las nuestras adecuándolas a nuestras necesidades.

El ARQuake[17] lleva ya un buen grado de avance en mezclar los personajes del juego con los escenarios reales, pero requiere de equipo muy especializado y costoso, al igual que otros métodos que se han realizado y que se revisarán en este trabajo de tesis.

OBJETIVOS

Así pues, nosotros trataremos de realizar un sistema que permita que un caracter animado pueda moverse por un camino libre de obstáculos por un escenario primero virtual y posteriormente de realidad aumentada. Una parte importante para ello es tratar de realizarlo con herramientas de alguna manera económicas y que no representen un gasto significativo o excesivo para lograr nuestra meta. Con ello garantizaríamos no sólo que este tipo de aplicaciones sea accesible a entidades educativas como escuelas o museos, queda abierta la posibilidad para que se aplique en otras áreas como pudiera ser la industria donde se requiera saber operar algún tipo de maquinaria, entre muchos otros.

Pretendemos la incorporación de agentes virtuales en entornos de realidad aumentada, considerando las limitantes que se presentan como la alineación de las cámaras virtual y real como se describirá más adelante. Con ello pensamos involucrar más al usuario ya que la interacción no se limita a texto o imágenes con algoritmos predefinidos que controlan la información que se presenta puesto que el tener un agente es como si existiera una especie de "guía" virtual con el que se pueda interactuar.

Estableceremos el método que resulte más adecuado para en primer lugar tener la representación virtual del entorno real, lo cual servirá para que el agente tenga conocimiento de su entorno y pueda determinar los caminos posibles cuando el usuario le presente una tarea. Esto en conjunción con otro trabajo de tesis que trata sobre la interacción con el agente por medio de la voz, llevara a la implementación de laboratorios virtuales que apoyen entre otros en el campo de la educación y la industria.

1. REALIDADES Y VIRTUALIDADES

Introducción

Antes de abordar el problema particular para el presente trabajo, es adecuado hacer una revisión de algunos de los conceptos con los que estaremos trabajando referentes a la realidad virtual.

En este capítulo revisaremos rápidamente que es la realidad virtual (RV) y las características que posee tales como interacción, inmersión y tridimensionalidad. Revisaremos los tipos de RV que existen y algunas herramientas a nivel de software y hardware que nos ayudan para crear un sistema de este tipo.

También revisaremos la realidad aumentada (RA) para ver sus características y marcar la diferencia que tiene con la RV. Expondremos los problemas y los tópicos a considerar al momento de pretender crear un sistema de este tipo.

Finalmente mencionaremos algunas posibles aplicaciones que tiene la RA y que podemos encontrar en nuestra vida diaria, aunque lo mas probable es que si no se está familiarizado con este tema nunca se haya percatado que existe ya entre nosotros.

1.1 Realidad Virtual

El término realidad virtual pudiera parecer algo confuso ya que se compone de dos conceptos:

Real: Aquello que tiene presencia física, que existe.
Virtual: Que tiene la capacidad de producir un efecto tal como una sensación, aún cuando físicamente no exista.

De ahí que el término se preste a confusión, derivando en términos como ciberespacio, realidad artificial, ambientes sintéticos, etc. Entre las diferentes definiciones que se aplican a la realidad virtual (RV), la que parece ser más precisa la define como un entorno generado por computadora, interactivo y tridimensional donde una persona queda sumergida (Aukstakalnis y Blatner, 1992[1]).

La RV permite al usuario interactuar con los escenarios tridimensionales generados por computadora al punto de que el escenario virtual se convierta en el escenario real para él y la computadora desaparezca de la mente del usuario.

Un sistema de RV tiene tres características principales:

Interacción: Permitir al usuario realizar acciones dentro del ambiente virtual. El sistema responde al usuario, creando una retroalimentación e interdependencia. El usuario debe ser capaz de navegar en el escenario virtual y debe de tener una ubicación en ese mundo.

Inmersión: Es la capacidad de hacer sentir al usuario que está dentro y forma parte del mundo virtual. Principalmente se enfoca en la información sobre la que se trabaja, olvidándose del resto que está a su alrededor. Con ello el usuario concentra su atención y convierte la información en experiencias.

Tridimensionalidad: Se refiere a la forma en que es modelado el mundo virtual. Esto involucra no solo la representación geométrica del escenario, también el sonido puede ser representado tridimensionalmente.

Todo esto debe ocurrir en tiempo real para que el usuario sienta los efectos claramente y acorde a sus actividades. Por ejemplo, si en este mundo virtual el usuario puede mover un objeto tal como una silla, el sistema debe ser capaz de realizar las tareas necesarias para ajustar la escena a los deseos del usuario. El objetivo principal de la RV es producir un ambiente virtual que difícilmente se pueda diferencial de uno real. En la actualidad podemos ver muchos ejemplos como los simuladores de vuelo o de manejo que sirven para que el usuario aprenda a controlar los dispositivos antes de enfrentarse a ellos en la realidad.

[1] Aukstakalnis, Steve y Blatner, David, "Silicon Mirage" Peachpit Press. 1992

Tipos de sistemas de RV

De acuerdo al modo en el que los sistemas de RV interactúan con el usuario, éstos se dividen en distintos tipos, donde se pueden mencionar:

a) Ventana hacia un mundo (Window on World - WoW)

Es cuando el mundo virtual se presenta únicamente en la pantalla de la computadora. Se consideran sistemas de RV sin inmersión. También se conocen como RV de escritorio. Pueden ofrecer una buena resolución y por lo general son de un costo no muy grande. Un ejemplo de esto está en la Fig. 1.1, donde se aprecia la ventana de un navegador Web cuyo contenido es una representación tridimensional de un CPU.

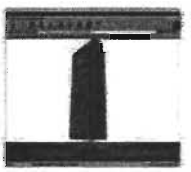


Figura 1.1 Sistema WoW[20]

b) Mapeo de video

Son sistemas que unen la entrada de video de la silueta del usuario con la computadora, interactuando en tiempo real con otros usuarios o imágenes generadas por la computadoras. Así, las acciones que el usuario realiza se reproducen en la pantalla de la computadora permitiendo desde afuera la interacción con los elementos de dentro. Este sistema ya presenta un cierto grado de inmersión. La Fig. 1.2 muestra un ejemplo de mapeo de este sistema.

Figura 1.2 Mapeo de video de las manos de un usuario

c) Sistemas inmersivos

En estos sistemas el punto de vista del usuario es sumergido completamente en el mundo virtual, por lo general mediante un HMD (Head Mounted Display) o en ocasiones se utilizan pantallas de proyección largas que envuelven la visión del usuario de manera que se consiga el efecto deseado. La Fig. 1.3 muestra un ejemplo de este sistema.

Figura 1.3 Sistema inmersivo[21]

La inmersión aquí puede ser de diferentes tipos:

1. Un solo usuario aislado (usando un HMD)
2. Una cabina que permita más de un usuario (pods, group, cab)
3. La caverna o cueva (Cave)

d) Telepresencia

Son sistemas que permiten que permiten controlar sensores remotos en el mundo real, uniéndolos con los sentidos del usuario. Campos como la medicina han empleado estos sistemas en el manejo de robos en cirugías que requieren mucha precisión. El más reciente ejemplo se encuentra en la exploración espacial de la superficie de Marte: el vehículo en Marte es controlado desde una estación en la Tierra.

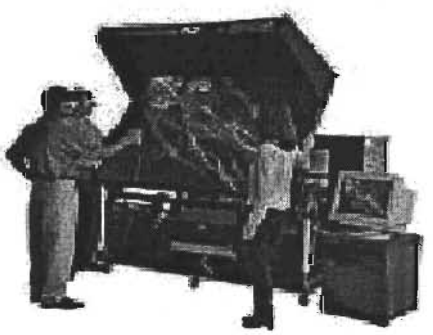


Figura 1.4 Equipo controlado por telepresencia[21]

e) Realidad mixta o aumentada

Sucede cuando se une la telepresencia con la RV, resultando en una realidad mixta o un sistema de simulación. La computadora genera salidas que se unen con las entradas de la telepresencia o puntos de vista del usuario. Este sistema se orienta a realzar las percepciones del usuario con respecto al mundo real. Para ello utiliza un tipo de HMD de visión transparente permitiendo al usuario ver la escena generada por computadora sobrepuesta a la escena real.

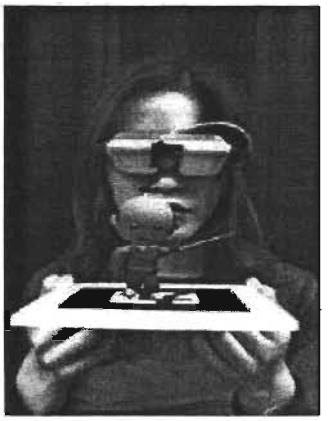


Figura 1.5 Realidad aumentada[22]

Componentes de un entorno virtual

Podemos distinguir dos grupos diferentes en la construcción de un escenario virtual:

Software: Desde el sistema operativo, programas de simulación, repositorio de datos, IA, etc.; los cuales facilitan la creación de dicho mundo y la coordinación de estímulos.

Hardware: Se refiere a la computadora en la que funcione el mundo y los posibles dispositivos que sirvan para interactuar en el.

Software

Dentro del software podemos mencionar: OpenGL, VRML, X3D, Java3D, 3DML, Iris Performer, QuickTime VR, OpenInventor, Crystal Space, entre otros. La Fig. 1.6 muestra algunos ejemplos de lo que se puede obtener con estos lenguajes.

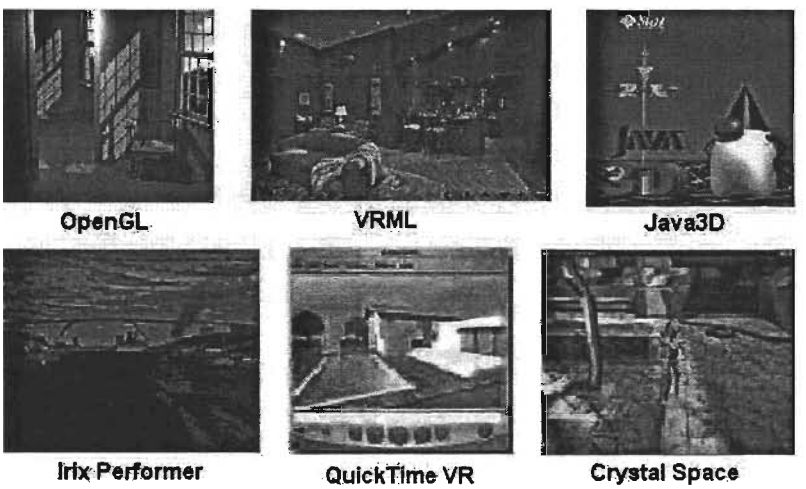


Figura 1.6 Ejemplos de Software empleado para RV[23]

Hardware

Existen diferentes dispositivos o periféricos que permiten introducirnos en una experiencia virtual, pudiendo distinguir los siguientes:

HMD (Head Mounted Display): Casco que posee audífonos , en ocasiones un micrófono, y dos monitores de cristal líquido que envía imágenes y sonidos al usuario. Además rastrea los movimientos de la cabeza y responde a ellos.

Figura 1.7 HMD[24]

GUANTE (Data Glove): Este guante está hecho de lycra y es recorrido por delgadas fibras ópticas, capaces de medir las torsiones de las articulaciones. Hace posible tocar objetos virtuales, enviando esa información a la computadora.

Figura 1.8 Data Glove

CAVE: La habitación estereoscópica immersiva (Computer-Animated Virtual Environment, CAVE) consiste en tres paredes y un suelo que son pantallas donde se proyectan imágenes estereoscópicas. Cuenta con ambientación sonora y un sistema que monitorea el movimiento.

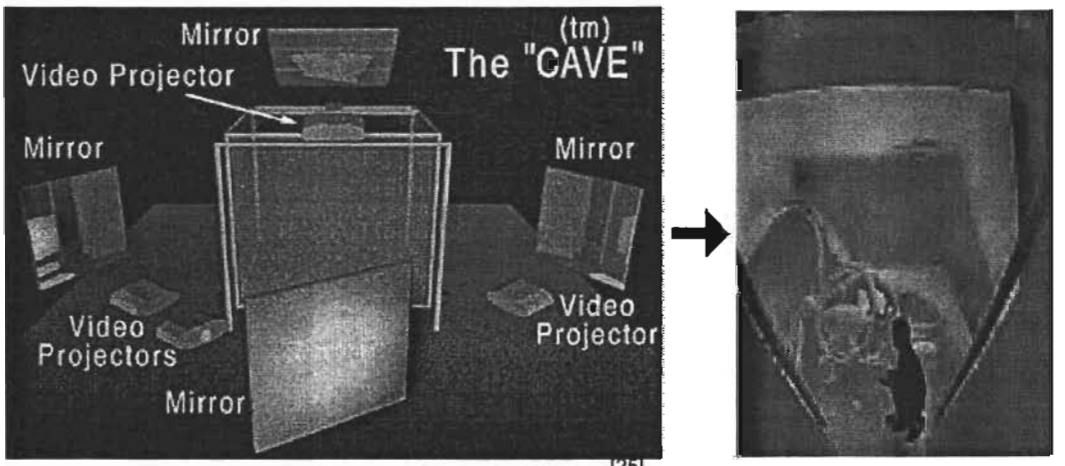


Figura 1.9 CAVE[25]

1.2 Realidad aumentada

La realidad aumentada es una rama cercana a la realidad virtual que pretende aumentar la información disponible para el usuario sobreponiendo a la escena real una escena virtual generada por computadora. Esta escena virtual puede componerse por texto, imágenes, sonidos o algún otro medio capaz de entregar información. El aumentar de esta forma la información de la que dispone el usuario puede ser útil para guiarlo en la operación de alguna máquina, presentarle información sobre la persona que está viendo o sobre alguna construcción, combinar imágenes de rayos x sobre el paciente, etc. Por todo esto puede emplearse en áreas como la medicina, ingeniería, entrenamiento militar, mantenimiento y ocio, entre muchas otras. Una de las principales diferencias entre la RV y la RA es la que se refiere al grado de inmersión del usuario en el escenario. La RV pretende que el usuario se sumerja completamente en la escena, mientras que en la RA la inmersión no es total ya que se tiene presente algún elemento del mundo real. En la RA el usuario tiene contacto con el mundo real y con el virtual. Milgram Takemura[2] (1994) propone una descripción que muestra la relación entre la RV y la RA, a lo que llama el continuo realidad-virtualidad:

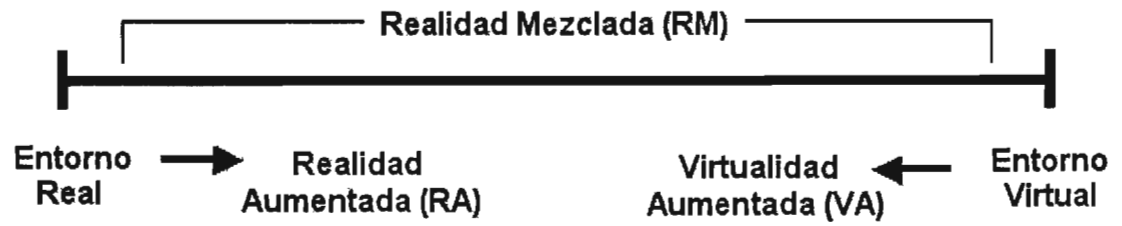


Figura 1.10 Relación de Milgram Takemura

[2] Milgram, P., H. Takemura, et al. (1994). Augmented Reality: A Class of Displays on the Reality-Virutality Continuum

El mundo real y el virtual están a los extremos de la línea. La RA es considerada como un primer paso de lo real a lo virtual, ya que ni todo es real ni todo es generado por computadora. Posteriormente aparece el término de virtualidad aumentada que se refiere a los sistemas donde se trabaja sobre el mundo virtual pero se incorporan algunos elementos del mundo real como mapeo de texturas de video sobre los objetos virtuales. El objetivo principal de la RA pretende que en algún momento el usuario no pueda distinguir la diferencia entre el mundo real y la parte aumentada.

Parte importante en la RA está en el seguimiento de las posiciones espaciales tanto del usuario como de los elementos reales que interaccionan con él para poder incorporar los elementos virtuales, ya sean de naturaleza visual, sonora o táctil.
Se considera a Ivan Sutherlan como el primero en concebir la idea de realidad aumentada, desarrollada en los cascos de los pilotos de vuelo simulado. Actualmente la RA ha sido ampliada a un sin fin de aplicaciones cada vez más interesantes.

Un problema muy difícil e importante de tratar es la fusión de la escena real con la parte que se genera por computadora.

Para la construcción de un sistema de RA son necesarios conocimientos en múltiples tecnologías para resolver todos los problemas: rastreo (tracking) de la posición de la persona, construcción e interpretación del modelo, reconocimiento de patrones, etc. Áreas como la graficación por computadora, visión por computadora e interfaces de usuario son algunas de las que intervienen en un sistema de RA.

En un sistema de realidad virtual tradicional hay que tener siempre en cuenta la posición y los movimientos del usuario y generar el mundo virtual en consecuencia. En el caso de un sistema de realidad aumentada el problema es mayor, porque al estar trabajando sobre el mundo real el ojo humano es mucho más sensible a los fallos de alineamiento entre la imagen real y la imagen generada por computadora. Para dar una sensación de realidad a la imagen final es necesario mejorar el sistema de rastreo con respecto a los sistemas tradicionales. La figura 1.11 muestra el proceso de un sistema de RA general.

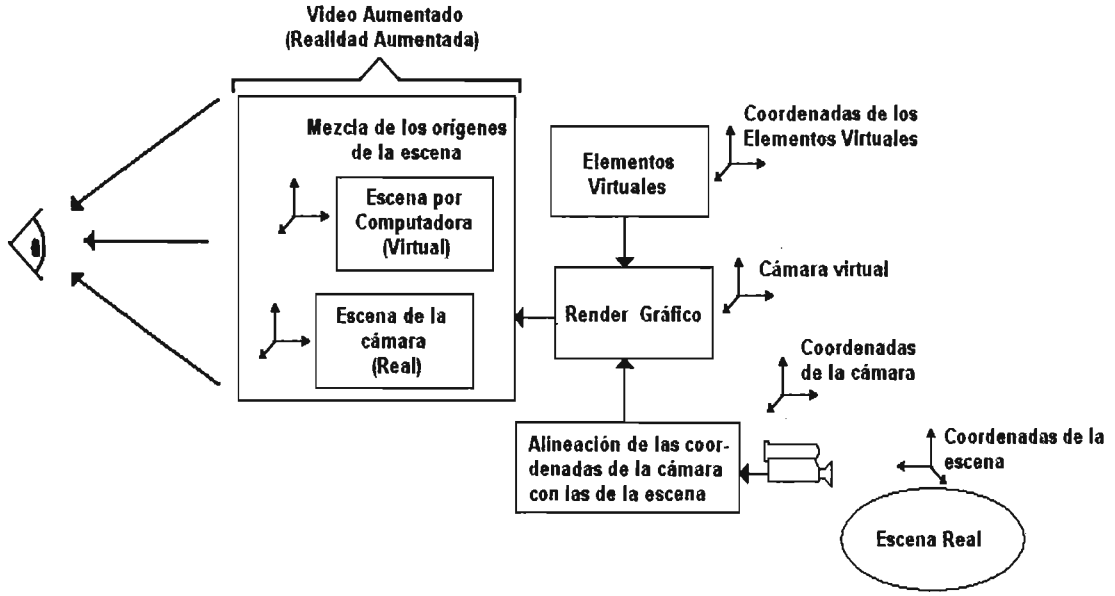


Figura 1.11 Sistema general de RA

1.3 Campos de aplicación de la RA

Son muchas las áreas en donde la RA ha entrado para enriquecer cada una de estas áreas. Solo mencionaremos algunas de estas de manera breve ya que se puede encontrar mucha información al respecto en libros y en la red.

Medicina

Es de mucha utilidad obtener una representación tridimensional a partir de una serie de imágenes obtenidas por ejemplo de TAC's o resonancias magnéticas para apoyar la labor de un cirujano. Por medio de la RA se pueden sobreponer las imágenes 3D a una escena captada por una cámara, y pasarlas por unos lentes o proyectarlas de manera tal que al cirujano le sea más fácil, por ejemplo, extirpar un tumor. En un ultrasonido se podría hacer que al ver el vientre de una mujer embarazada por medio de unos lentes especiales, se sobreponga a la imagen plana del feto una representación tridimensional de este, pareciendo como que realmente está dentro del vientre.

Entretenimiento

Un ejemplo muy claro es cuando vemos la sección de clima en las noticias. Esos mapas cambiantes frente a los cuales siempre está una persona describiéndolos, son pantallas azules o verdes que proyectan los mapas y se mezclan con la imagen del presentador para dar la impresión de que ambos existen en el mismo plano.

También ocurre en los partidos de fútbol, donde en el medio tiempo podemos ver elementos tridimensionales en el centro de la cancha anunciando algún producto, dando la sensación de que realmente existen, al menos mientras ninguno de los jugadores pasen por esa zona. Otro ejemplo en el campo de los videojuegos es el ARQuake, una implementación del juego de Quake llevada a la RA.

Entrenamiento militar

Cascos con cámaras y lentes que presentan a los soldados información tal como la posición de otros elementos de su equipo, de sus enemigos, de víveres, de centros de atención médica, etc.

Diseño industrial

Supongamos que se quiere construir un puente entre dos edificios. Se puede crear un modelo tridimensional del prototipo y por medio de la RA sobreponerlo de forma tal que parezca que realmente existe ya. De esta forma se puede apreciar cómo quedará finalmente el puente antes de ser construido y probablemente se detecten errores o mejoras al diseño.

2. ANTECEDENTES

INTRODUCCIÓN

En este capítulo revisaremos algunos trabajos que se han realizado en entornos de realidad aumentada y ambientes de colaboración mixta.

Hay que prestar particular atención en los problemas ineludibles que se presentan en ambientes de realidad aumentada como los revisados en este capítulo.

Revisaremos el ARQuake[17], una versión del juego Quake en un entrono de realidad aumentada. Desde aquí podemos notar el uso del ARToolkit, conjunto de librerías escritas en C y que permiten trabajar más fácilmente en realidad aumentada.

Tinmith[18] está mas relacionado con aspectos de hardware, y su principal uso es en un sistema de modelado en realidad aumentada.

El Magic Book[16] es una aplicación hecha por los creadores del ARToolkit[15] y que permite mostrar sus características de inmersión y de mezcla de elementos reales y virtuales.

Finalmente en este capítulo revisaremos el ARToolkit que se está convirtiendo rápidamente en una herramienta importante para desarrollar áplicaciones de realidad aumentada.

2.1 ARQuake

El ARQuake[4] es una versión del juego Quake[11] llevado a un entorno de realidad aumentada. El mundo físico es representado como un modelo que Quake puede entender. La información que se aumenta son los monstruos, las armas y demás objetos que tengan alguna importancia en el juego. Todo eso se envía al motor (engine) del juego, aunque al usuario sólo se le muestra la información que se aumenta por medio de unos lentes especiales, llamados *see-through*, mediante los cuales se puede ver tanto el mundo real como la información generada por computadora al mismo tiempo.

Para la parte del levantamiento del mundo 3D emplean el ARToolkit[15], que son unas librerías escritas en lenguaje C escritas por Mark Billinghurts y que ayudan a calcular la posición y orientación de la cámara relativa a marcas físicas en tiempo real. Estas marcas son cuadrados que contienen patrones en el centro. El ARToolkit determina la distancia y orientación de la cámara con respecto al patrón. De esta forma se puede determinar la posición de elementos generados por computadora en aplicaciones de RA. Con la ayuda de estos patrones se puede levantar la geometría que representará al mundo real en el entorno del Quake. La figura 2.1 muestra un ejemplo.

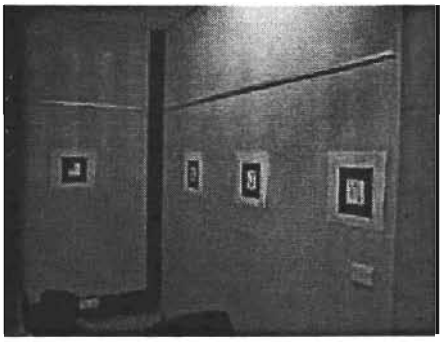


Figura 2.1 Patrones guía para el levantamiento del modelo 3D[26]

Emplean un hardware llamado Tinmith-4[18], el cual discutiremos más adelante, y que consiste en una computadora portátil montada en una mochila que va en la espalda donde se conectan todos los periféricos utilizados. Esto permite que el sistema pueda ser transportado en cualquier lugar, tanto en interiores como exteriores de edificios. La figura 2.2 muestra una persona usando el equipo mencionado.

Los desarrolladores del ARQuake[17] crearon un nivel al estilo Quake que representa una parte del campus de la Universidad del Sur de Australia. Ambos escenarios, el real y el virtual, están alineados de tal forma que coinciden durante el juego.

Las paredes y el techo modelados no se presentan al usuario durante el juego, sólo sirven internamente al engine del mismo. El usuario sólo ve monstruos, armas, piso y regiones de interés.

Figura 2.2 Tinmith-4 Backpack[26]

Aún cuando la alineación del mundo real con su representación en Quake sea muy exacta, se debe tener un perfecto ángulo de visión para evitar inconsistencias en el mundo virtual. El campo de visión del juego es de 90°, 45° para cada lado. Este campo de vista sufre de una distorsión causada por la lente de la cámara cuando se comparan objetos de Quake en el mundo real. El HMD que utilizan, I-Glasses, tiene 25° de campo de visión horizontal. Se hace un ajuste del HDM para trabajar en el Quake, cambiando el campo de visión del juego y escalando los elementos gráficos, por lo que finalmente el campo de visión manejado es de 25°, lo que corrige la distorsión mencionada.

El sistema de rastreo de la posición del usuario involucra un Sistema de Posicionamiento Global (GPS por sus siglas en inglés) combinado con un sistema basado en visión. El rastreo lo dividen en tres: construcciones cercanas y lejanas en exteriores, e interiores. El formato que manejan para la información de la posición es el WGS 84/UTM[18] así como los ángulos de head/pitch/roll para la información de la orientación. El uso de marcas visuales mejora la posición relativa y la orientación con respecto a la cámara. El equipo que utilizan es sofisticado y bastante caro. Tan solo el GPS que utilizan tenía un precio de 4 mil dólares en el 2001.

Para mejorar la exactitud cuando el usuario está cerca de edificios utilizan una extensión del ARToolkit que consiste en tener marcas (los patrones ya mencionados) de un metro de largo por uno de ancho, cuando lo normal es de apenas unos cinco o diez centímetros. Estas marcas se colocan en objetos reales como en las paredes como ya vimos, y la información que de ellas se obtiene ayudan no solo a levantar la geometría, sino que además permiten posicionarla en el juego. Para el caso de interiores, las marcas son de 19 cms cuadrados colocadas en diferentes puntos de manera tal que no importa hacia dónde mire el usuario, siempre vea al menos una de las marcas.

Con la cámara montada en el backpack a una altura de 1.70 mts y con un cuarto de 2.7 mts de altura, el campo de visión es de por lo menos 1.3 mts cuadrados, por lo que un tamaño de 10 cms cuadrados para las marcas es suficiente.

Estos son puntos importantísimos en este tipo de aplicaciones. El tamaño de los patrones influye irremediablemente en la capacidad del ARToolkit para encontrarlos o no, pero también el campo de visión que nos permite tener la cámara utilizada es un factor importante ya que entre mayor sea este campo mejor será la posibilidad de ver el mundo real en proporciones adecuadas. Una cámara con un lente pequeño quizá no tenga mucho alcance ni abarque mucho del campo a registrar, por lo que no sería una buena opción al momento de elegir el equipo.

En el 2001, los desarrolladores del Tinmith estimaban que el quipo completo tendría un costo superior a los 10,000 dólares, considerando la laptop, el GPS, el IS-300 para el rastreo, el HMD, además del costo del backpack, cámaras, cableado, controladores del hardware y la instalación del software. Aún cuando a la fecha algunos de estos precios han cambiado, así como las tecnologías empleadas, todavía no se puede hablar sobre la posibilidad de que un sistema de este tipo sea accesible al público.

2.2 Tinmith

El sistema Tinmith[!8] es una arquitectura que combina elementos de software y hardware para desarrollar RA. Está basado en software libre comprendiendo el sistema operativo Linux, herramientas GNU y librerías, el compilador GNU de C/C++, Xfree86, OpenGL, PostgreSQL para bases de datos, y Freetype para el empleo de fuentes.

Tinmith fue iniciado en 1998 y para 2001 era ya parte del trabajo de tesis doctoral de Wayne Piekarski[7]. El corazón del sistema es una laptop, la cual procesa la información proveniente de los diferentes sensores y dispositivos, retroalimentándose con el usuario que utiliza un HMD *see-through*. Elementos importantes son la capacidad de procesamiento del CPU, memoria, video, puertos E/S, y compatibilidad con Linux. Los HMD son también importantes, ya que sus características determinan aspectos como el campo de visión (FOV en inglés) que se refiere a que tantos elementos el usuario puede ver en relación con la distancia a la que se encuentre de lo que está observando, la calidad con la que presentan la imagen, el peso y por supuesto, el costo.

Para que se puedan dibujar los elementos que la computadora genera de manera que coincidan con el mundo real, es necesario saber en dónde se encuentra el usuario. Ellos utilizan equipos GPS y una forma de transmitir la información de la posición que estos regresan mediante un cable RS-232 hacia un puerto serial para que la computadora los procese. Estos datos contienen la información de latitud/longitud/altura referentes a la posición del usuario.

Otro aspecto importante es conocer información de head/pitch/roll del usuario para saber su orientación, es decir, hacia donde está viendo. Para ello se utilizan sensores de rastreo como ya vimos en el ARQuake. Este dispositivo es montado en el HMD y sus datos se envían hacia un puerto serial libre de la computadora. Otro componente es una tarjeta de red inalámbrica que opera a 11 Mb/seg.

La mayoría de los dispositivos se comunican vía un puerto serial, lo cual es un problema considerando que las computadoras, sobre todo las portátiles, vienen con un solo puerto serie, y a últimas fechas es muy raro encontrar puertos serie con el auge del USB y del IEEE 1394. Para librar este problema se hace uso de una tarjeta adaptadora PCMCIA que provee los puertos extra necesarios. En ese entonces era más difícil que un sistema Linux reconociera estos elementos, incluso a la fecha hay dispositivos que no son totalmente soportados por este sistema operativo.

La principal utilidad de Tinmith es para realizar un sistema de modelado en realidad aumentada, aunque como vimos también se emplea en el ARQuake. Las partes que lo componen son los que se listan a continuación. La figura 2.3 muestra este sistema.

- Laptop Dell Inspiron 8100, Pentium 3 a 1.2 MHz, con tarjeta gráfica GeForce2
- Trimble Ag132 GPS
- Intersense IS-300 con sensor de orientación, rastreo magnético con giroscopio
- Cámara Point Grey Firefly 1394 a 640x480 RGB a 15 fps
- Sony Glasstron HMD con display de 800x600 y 35° de FOV
- Convertidor PAL de SVGA a salida de TV para grabar en un cassette
- Guantes de bajo consumo de poder y caja controladora
- Adaptador serial Keyspan con 4 puertos serie a USB
- Hubs USB y Firewire
- Fuentes de poder integradas que proporcionan +5, +9 y +12 volts

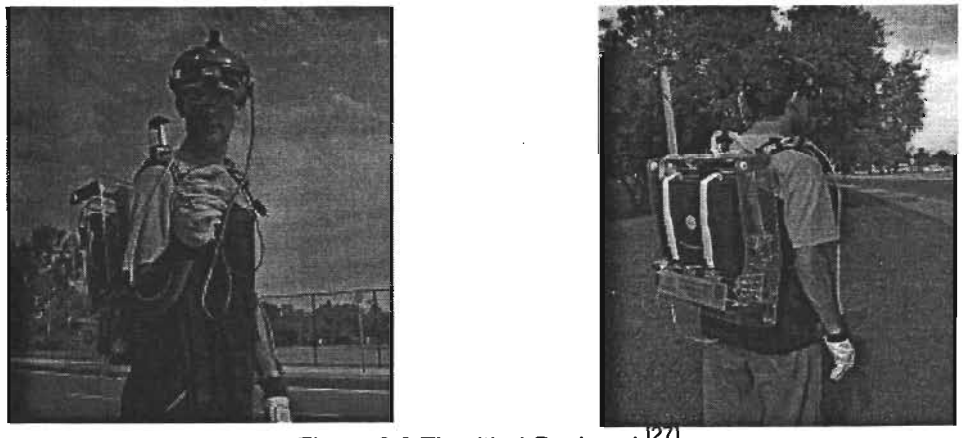


Figura 2.3 Tinmith-4 Backpack[27]

La técnica que desarrollaron para el sistema de modelado se llama Bread Crumbs[6] y permite a los usuarios capturar grandes características del piso de exteriores utilizando la mochila (backpack) arriba mencionado. Con ello un usuario puede capturar geometría tridimensional en cualquier lugar, a cualquier hora en cualquier parte de la Tierra, con interacciones de usuario basadas en manos y gestos faciales.

Estas técnicas tienen limitaciones, como que requiere una línea de vista directa hacia el objeto, lo cual no siempre es posible. Además recordemos que al estar trabajando en exteriores el sistema está expuesto a factores de iluminación, polvo, interferencia, entre otros.

Parte importante del sistema son unos guantes especiales que usan marcas que son rastreadas por la cámara, como en el ARQuake se hace con las marcas en los edificios, lo cual permite saber la posición del guante, además de que el sistema "entiende" los gestos(movimientos) que hacen los dedos por ejemplo, al sostener algún objeto. La figura 2.4 muestra mejor lo que se describe.

Figura 2.4 Sistema de modelado con RA[27]

La técnica Bread Crumbs está inspirada en el cuento de Hansel y Gretel quienes tiraban migajas de pan para poder regresar a casa. Imaginemos que queremos modelar algún elemento que se encuentra en la tierra, como puede ser un camino, un edificio, un pedazo de tierra, etc. Siguiendo esta idea basta con que el usuario camine alrededor del perímetro del objeto que desea modelar, y en los puntos que le interesen pone una marca de posición mediante el GPS. Cuando el usuario ha recorrido totalmente el perímetro el resultado es un polígono plano representando la superficie recorrida (o quizá rodeada), mismo que si se desea se puede extruir hacia arriba para modelar el objeto deseado.
Esta técnica se ha empleado para modelar caminos, estacionamientos, edificios pequeños y áreas del campus de la universidad de los creadores.

2.3 Magic Book

El Magic Book[9] involucra aspectos de la realidad física, la realidad aumentada y la realidad virtual inmersiva. Luce como un libro ordinario de historias con dibujos y texto pero tiene una característica muy particular. Quizá alguna vez vio libros que al abrirlos los dibujos "saltan" del libro. Pues bien, el Magic Book hace lo mismo pero con las nuevas tecnologías. El usuario puede ver el libro sin ningún equipo en especial y puede ser leído de esta forma como cualquier libro normal, o bien puede utilizar un HMD para ver el libro, el cual contiene marcas como las mencionadas en el ARQuake, y al verlas de ellas emergen diferentes elementos tridimensionales que representan parte de la historia del libro.

El usuario puede ser desde un participante externo que puede ver la escena completa que sale del libro, hasta ser un participante del mismo, ya que puede sumergirse en la escena y verla desde dentro. Cada usuario se representa como un avatar (personaje) dentro de la historia, y cada uno puede ver la escena desde su particular punto de vista.

El punto principal del Magic Book es la colaboración, y su interfaz maneja tres niveles, los cuales se pueden ver representados en la figura 2.5. mostrada más abajo

* Nivel 1: Objetos físicos. Los usuarios pueden revisar el libro al mismo tiempo.
* Nivel 2: Objeto de RA. Usando HMD's el usuario puede ver e interactuar con objetos virtuales que aparecen sobre las páginas del libro.
* Nivel 3: Espacio Virtual Inmersivo. Los usuarios pueden "volar" en el espacio virtual conjuntamente y verse cada uno representado como personajes virtuales en la escena.

Figura 2.5 De izquierda a derecha los tres niveles[28]

Un aspecto interesante es esto de que cada usuario pueda ver al otro desde su particular punto de vista. Si un usuario ve el libro en una forma externa, es decir simplemente en realidad aumentada, y otro usuario está inmerso en la escena y es representado como un avatar, el primer usuario puede ver al segundo como parte del libro. La figura 2.6 muestra claramente este ejemplo.

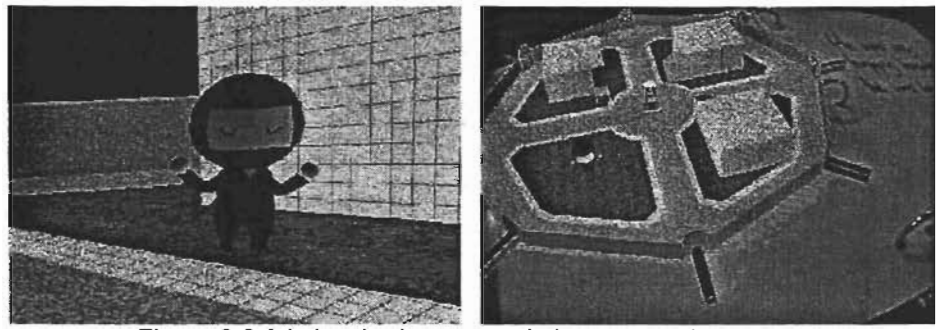


Figura 2.6 A la izquierda un usuario inmerso en la escena, a la derecha una vista externa de otro usuario[28]

Podemos distinguir tres componentes principales en el Magic Book. El primero es el libro físico con los patrones. El segundo es el HMD *see-through* que tiene unido una cámara de video que captura los patrones y hace posible el reconocimiento de los mismos por medio del software ARToolkit; además tiene un dispositivo InterSense InterTrax 30 a una agarradera que registra la posición y orientación de la cabeza del usuario. El tercer componente es un botón que está también en la agarradera donde se encuentra el tracker y es el que permite cambiar la forma de interacción con el libro: si el usuario está viendo la escena desde un punto de vista de RA normal, al presionar el botón puede sumergirse en la escena y ser parte de ella como ya se vio, y si lo presiona de nuevo regresa a la vista anterior sin problema. La figura 2.7 muestra a la izquierda los elementos del Magic Book, y a la derecha lo que el usuario está observando.

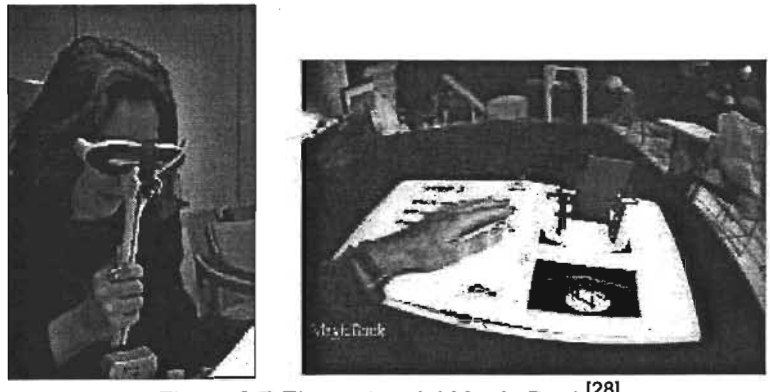


Figura 2.7 Elementos del Magic Book[28]

La salida de la cámara se conecta a una computadora Silicon Graphics O2. La salida de video de la O2 se conecta al HMD. De esta manera se mezcla el video y los elementos virtuales. La computadora se encarga tanto del reconocimiento de los patrones como de la generación de los elementos tridimensionales a una velocidad entre los 15 y 20 fps, dependiendo de la complejidad de la escena.

Consideremos que también se está trabajando con video en tiempo real, lo cual ya marca requerimientos de equipo que soporte este procesamiento. Actualmente la mayoría de las computadoras pueden trabajar con estos elementos, ya sea una O2 o una PC.

2.4 ARToolkit

El ARToolkit[13] es una librería escrita en lenguaje C que permite desarrollar aplicaciones de realidad aumentada. Utiliza técnicas de visión computacional para calcular la posición de la cámara y la orientación relativa a los patrones, permitiendo sobreponer objetos virtuales en esos patrones, a menudo utilizados como cartas o pedazos de papel. Provee un rastreo rápido y preciso y corre en diversas plataformas: Silicon Graphics, Linux y Windows.

En cada uno aunque la implementación varíe la funcionalidad es la misma y el desempeño depende de la configuración del hardware.
ARToolkit soporta tanto RA por video como por dispositivos *see-through*. En el primero la RA se da al sobreponer los elementos virtuales directamente en el video en tiempo real capturado por la cámara. La otra forma consiste en utilizar dispositivos como los HMD *see-through*, los cuales permiten ver el mundo real sin la necesidad de pasar el video que captura la computadora, y a la vez permite desplegar directamente el contenido virtual en ellos. Esta segunda forma requiere una calibración más complicada de la cámara.

ARToolkit es distribuido bajo la licencia General Public License (GPL) y es gratis para aplicaciones no comerciales. Si se desea utilizar en aplicaciones comerciales se debe contactar a alguno de sus desarrolladores, como Mark Billinghurst, para detalles de la licencia.

Los requerimientos básicos para desarrollar aplicaciones con el ARToolkit es una fuente de video como una videocámara o una cámara web y una computadora que permita a esa fuente de video conectarse. Lo más común es utilizar una cámara USB que tenga los controladores adecuados de acuerdo a la plataforma donde se utilice.

ARToolkit trabaja de la siguiente manera. Primero el video capturado se transforma en una imagen binaria en blanco y negro basada en niveles de luminosidad. Luego se buscan regiones cuadradas en esta imagen; se encuentran muchas áreas que no son necesariamente los patrones que se buscan, por lo que cada región es comparada con patrones que se debieron haber capturado con anterioridad y registrado de tal forma que el sistema los reconozca. Cuando encuentra una región que coincide con un patrón ARToolkit utiliza el tamaño y orientación del patrón para calcular la posición de la videocámara relativa al patrón. Una matriz de 3x4 guarda esta información y se usa luego para establecer la posición de la cámara virtual. Aquí puede prestarse a confusión el hablar de la cámara real y la virtual cuando en realidad son la misma, o mejor dicho comparten el mismo espacio. La cámara real es la que captura el video, y la virtual está en la

misma posición ya que ambas están viendo hacia la misma dirección, al mismo objeto. De esta forma las gráficas generadas se dibujan precisamente sobre los patrones. Se utiliza OpenGL para establecer la posición de la cámara virtual y dibujar el contenido virtual.

La figura 2.8 muestra el esquema del funcionamiento del ARToolkit, y la figura 2.9 muestra una aplicación práctica del mismo.

ARToolkit viene con todo un conjunto de herramientas para desarrollar aplicaciones. Tiene herramientas que sirven para capturar patrones propios y utilizarlos en nuestras aplicaciones, otras que perrniten calibrar la cámara y los HMD see-through si se utilizan, así como código que se puede utilizar como base para el desarrollo de aplicaciones. Todo eso y más se puede encontrar en el sitio de ARToolkit[15].

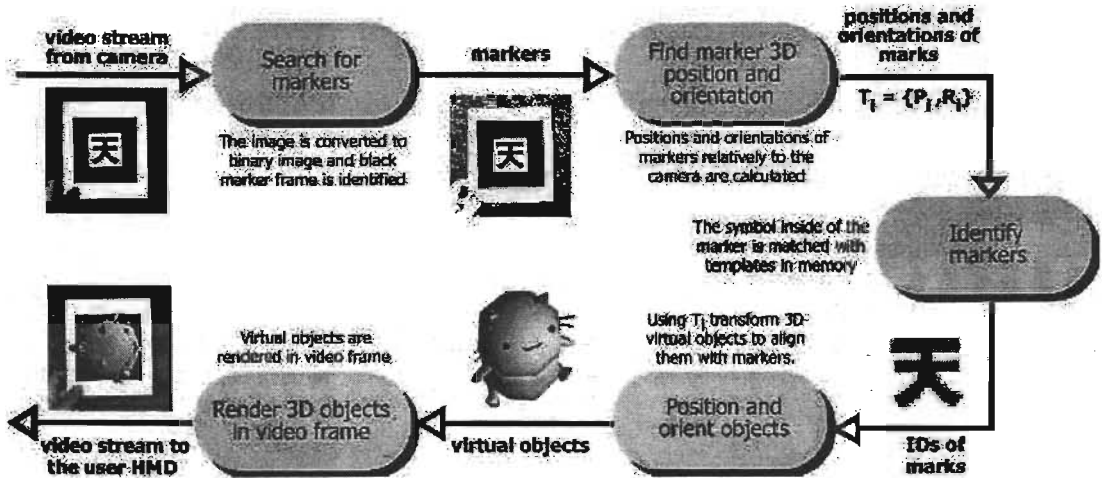
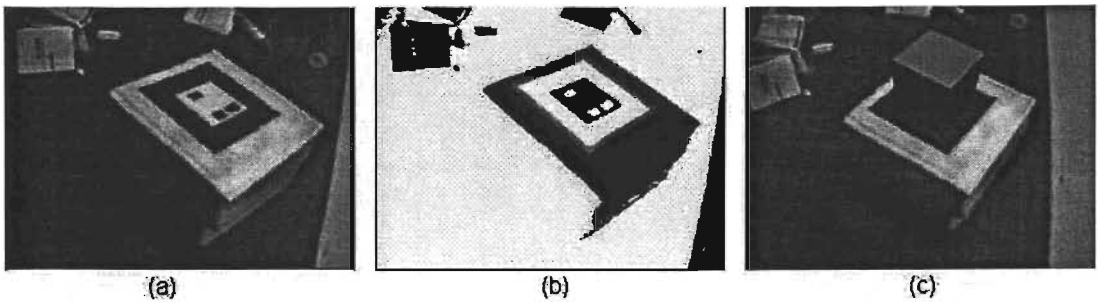


Figura 2.8 Funcionamiento del ARToolkit[29]



(a) (b) (c)

Figura 2.9 ARToolkit en acción
(a) Captura de video,(b) Captura y reconocimiento de regiones,
(c) Figura 3D sobrepuesta al patrón[29]

3. Búsqueda de Caminos

INTRODUCCIÓN

Ya que deseamos lograr una navegación dirigida que controle el movimiento de nuestro personaje por el ambiente, lo que necesitamos es comenzar por encontrar un método que nos ayude en la búsqueda de caminos. Existen varios algoritmos que se utilizan tanto en la programación de videojuegos como en la programación de robos móviles. Revisaremos los necesarios para establecer una comparación que justifique la elección de uno de ellos como el algoritmo a utilizar.

Antes de comenzar es necesario señalar que comenzaremos por entender el algoritmo e implementarlo en un entorno bidimensional antes de hacer la representación virtual. Esto nos ayuda a entender mejor las partes que intervienen y así es más fácil pasarlo después al ambiente en el que lo necesitamos y adaptarlo según se requiera.

Básicamente en lo que se refiere a la búsqueda de caminos (o pathfinding en inglés) nos encontramos con que son dos los que se eligen comúnmente: el algoritmo de Dijkstra[3,14] y el de A*[3,14].

Como veremos en este capítulo, Dijkstra es capaz de encontrar una ruta entre un punto inicial A y un punto final u objetivo B, aunque tiene un alto costo computacional y en cuanto a tiempo se refiere. Por otro lado, A* requiere de menos cálculos para encontrar un camino que solucione el problema de ir de A hasta B. Otro aspecto importante es saber si es que realmente existe un camino entre esos dos puntos, no podemos enfrascarnos en repeticiones infinitas del algoritmo si realmente no existe una ruta que los conecte. Ambos algoritmos permiten conocer esto en un número finito de pasos.

Al final se presenta una comparación de un caso sencillo entre el algoritmo de Dijkstra y el A* a fin de dejar claro las diferencias de uno sobre el otro. Además se presentan también un par de ejemplos que muestran como el algoritmo se puede adaptar a cualquier representación de un cuarto, ya sea muy simple o muy compleja e inclusive inusual.

3.1 Algoritmo Dijkstra

El algoritmo de Dijkstra[3.14] encuentra la ruta más corta entre dos nodos en un grafo donde cada borde tiene pesos no negativos. Dada la ruta más corta entre *s* y cada uno de un conjunto dado de nodos, existe otro nodo *x* tal que la ruta más corta de *s* a *x* va de *s* a v_i a *x*. Ese es el vértice *x* que minimiza la distancia $(s,v_i)+w(v_i,x)$ donde w(i,j) es la distancia del borde *i* al *j* y la distancia (i,j) es la distancia entre la ruta más corta entre ellos.

Para ver esto un poco más claro, veamos un ejemplo. Supongamos que queremos viajar de una ciudad a otra que se encuentra lejos. Si se utiliza un mapa se podrían encontrar las posibles rutas para realizar el viaje pero ¿cuál sería la ruta más corta?.
Podemos utilizar el algoritmo de Dijkstra para hallar la longitud del camino mínimo entre las dos ciudades, que serían los nodos de nuestro grafo. Consideremos un grafo que contiene *n* nodos: *v1,v2,v3,...,vn*; es necesario encontrar la longitud del camino mínimo desde *v1* hasta *vn*. Aún cuando el algoritmo obtiene el camino más corto desde un nodo a otro también genera las longitudes de caminos mínimos desde *v1* hasta todos los demás nodos del grafo, es decir, analiza todas las posibilidades. Se asocia un valor de distancia a cada uno de los nodos del grafo, que representa el peso de los bordes que mencionamos antes. Al comenzar se selecciona el nodo inicial *v1* y se da el cero a la distancia más corta desde ese nodo hasta si mismo. De lo que se trata es hallar el nodo que se encuentre más próximo a *v1*. A continuación se busca el nodo que esté inmediatamente más próximo a *v1*, y así sucesivamente hasta un momento tal que el nodo *vn* será el siguiente a *v1*, habiendo encontrado entonces el camino más corto. A medida que se van hallando nodos sucesivamente más alejados de *v1*, se van colocando en un conjunto *S*.

Consideremos que el grafo de la Fig. 3.1 representa nuestro problema

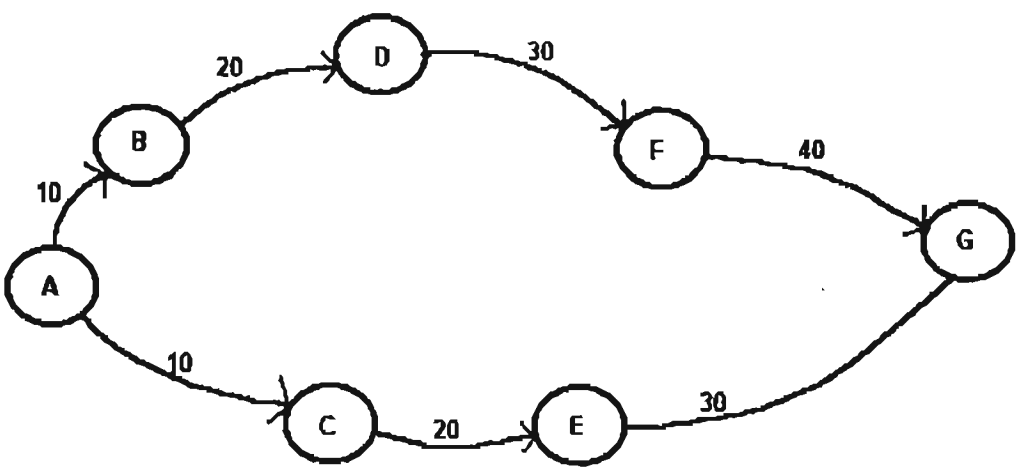


Figura 3-1 Grafo para un caso de Dijkstra

Si quisiéramos ir desde A hasta G, y asignamos un peso de 10 para cada borde (línea) y que aumente conforme se aleja de A, tendríamos primero que generar todos los pesos para cada uno de los bordes, y después encontraríamos que el camino más corto es A-C-E-G. Esto Puede parecer muy obvio y quizá para este caso no será necesario el algoritmo de Dijkstra; sin embargo, si nuestro problema comienza a crecer esta puede ser una buena opción. La parte negativa es que este algoritmo calcula los pesos de todos los nodos en el grafo, lo que representa una gran desventaja en grafos grandes con muchos nodos ya que implica tiempo de cómputo para realizar todos los cálculos. Además hace cálculos innecesarios porque no se detiene cuando alcanza el nodo objetivo sino hasta que ha revisado todos los nodos y determina el camino más corto. Este aspecto es importante si se considera que se desea aplicar este algoritmo en un sistema en tiempo real ya que seguramente no es el único proceso que intervendrá.

3. 2 Algoritmo A* (A-Star)

El algoritmo A*[3,14] (A estrella) o como algunos lo llaman, la *"estrella de la búsqueda de caminos"* ya que se considera el mejor para encontrar el camino más corto de un punto A hacia un punto B. Es el más socorrido en la programación de juegos, aunque se puede aplicar en otras áreas.

Un aspecto importante que hay que resaltar es que este algoritmo permite considerar las características del terreno por el que se moverá nuestro personaje; así podemos considerar los obstáculos que se encuentran o si se trata de un terreno de fácil tránsito como concreto o si es imposible de pasar caminando como agua, etc. Comencemos a revisar el algoritmo a fin de entender cómo trabaja y por qué fue elegido como nuestro algoritmo de búsqueda.

Debemos comenzar por considerar la representación del entorno por el que nuestro personaje se moverá. A* considera que el entorno (el cual puede ser un cuarto o un laboratorio, por ejemplo) está contenido en una malla, por lo general cuadriculada, a la que se le llama mapa. Esto ayuda mucho ya que permite tener una representación 2D de nuestro entorno 3D, lo que nos ahorra muchos recursos de la computadora. Esta malla no necesariamente tiene celdas cuadriculares, pueden ser hexagonales por ejemplo, pero se consideran las primeras por la simplicidad que representan. Así este mapa puede representarse fácilmente en una matriz bidimensional. Cada elemento de la malla representa un nodo en nuestro mapa. Por otro lado, necesitamos considerar que tenemos nuestro punto inicial A y nuestra meta B. La figura. 3.2 ilustra mejor estos conceptos.

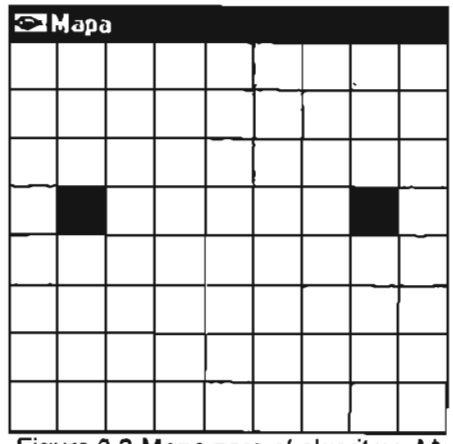


Figura 3.2 Mapa para el algoritmo A*

Consideremos que nuestro punto de inicio se encuentra en el cuadro verde y nuestra meta es el cuadro rojo. Antes de avanzar más hay que revisar cómo es que decidimos dividir nuestro mapa en cuadros y cuál es su proporción con respecto al entorno real. Si partimos de la idea de que lo que queremos lograr es situar un agente virtual en nuestro entorno real para lograr un sistema de RA, debemos considerar el espacio que nuestro personaje ocupará en esa escena. Si tratamos de asemejar la dimensión del agente con la de una persona real nos damos cuenta que un área de 50x50 cms. es suficiente para situarla. Esto nos lleva a que por ende el resto de nuestro mapa se verá dividido en cuadros del mismo tamaño por llevar una congruencia. Entonces, si en una localidad o posición de nuestro mapa existe algún obstáculo, ya sea una pared u otro objeto como una mesa, un librero, etc., utilizaremos cuadros negros para representar estos obstáculos, los cuadros (bloques) suficientes para abarcar al objeto en sí. Esto trae pros y contras. Por un lado puede ser que en uno de estos bloques quepa bien un objeto tal como una pequeña mesa de 30x30 cms., pero ¿qué pasa si un objeto abarca apenas un poco más de un bloque? Por ejemplo, una mesa de 1 mt. de largo por 60 cms de ancho apenas es cubierta por dos bloques a lo largo, restando 10 cms. a lo ancho por cubrir. Por otro lado, si hiciéramos los bloques más pequeños, quizá de 10x10 cms. por decir algo, podríamos abarcar mejor a los objetos, pero necesitaríamos más de ellos para representar a nuestro personaje y nuestro mapa estaría dividido en muchos cuadros. También es cierto que bloques más pequeños podrían hacer más suave el camino que recorra el personaje, pero también implica un mayor número de cálculos. Al final lo que está claro es que las consideraciones en cuanto a representación quedan a criterio del diseñador. Para nuestro caso decidimos utilizar bloques de 50x50 cms. utilizando los necesarios para cubrir nuestro obstáculo. En el caso de la mesa de 1 mt. x 60 cms. utilizaríamos pues cuatro bloques negros para representar ese objeto ya que lo que más nos interesa es que nuestro agente no colisione con los obstáculos y que no se traslape con ellos; además, recordemos que el trabajo de nuestro agente será el de un guía o instructor, por lo que no necesita estar muy cerca del objeto ya que no puede manipular nada del ambiente real.

Así que en el caso de la Fig. 3.2, tendríamos un mapa de 4.5 mts. de ancho por 4 mts. de alto. Por otro lado hay que tener presente que esto es la representación gráfica del mapa. Recordemos que la representación para el algoritmo se hace en una matriz bidimensional ya que el algoritmo trabaja con los índices de esta matriz y no con el tamaño en sí de los cuadros. Por lo tanto, tendríamos una matriz de 8 por 7, comenzando desde el 0.

Regresando al algoritmo, necesitamos los siguientes elementos:

* Una lista abierta, que es donde colocaremos los nodos vecinos por visitar de un nodo padre. Estos nodos pueden ser parte del camino o no, eso se determinará al realizar la búsqueda.
* Una lista cerrada, que es donde colocaremos los vecinos ya visitados y que nos llevan a la solución.
* Un vecino es un nodo adyacente a un nodo padre.
* El nodo padre es aquél nodo que tomamos como base para buscar el mejor camino en un momento determinado. El saber quién es el padre de un nodo nos ayudará al final para encontrar el camino que nos de la solución.

Hasta este punto, tendríamos algo como lo de la Fig. 3.3.

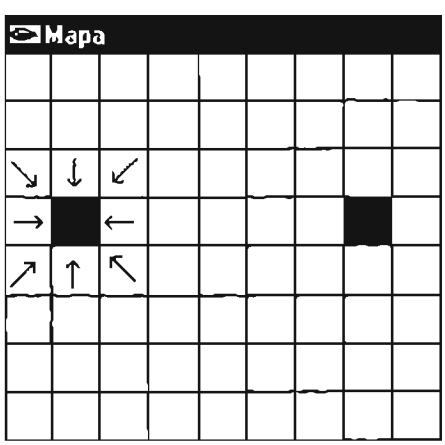


Figura 3.3 Representación de los elementos de A*

Las flechas apuntan hacia su padre. Como se ve, cada nodo tiene ocho posibles vecinos, y son posibles porque solo se considera como vecino aquel que sea un nodo libre o transitable. Conforme se van eligiendo los vecinos se van "pesando", lo que es la parte medular del algoritmo. Consideremos que tenemos una función que determina el peso de un nodo, la cuál está dada por

$$F = G + H$$

Donde

* G representa el costo de ir desde el nodo padre actual hacia alguno de sus hijos o vecinos. Esto se hace considerando la G del padre más la propia G del nodo, la

cual como regla se maneja de 1.0 para movimientos horizontales y verticales y de 1.4 para los diagonales. Los movimientos se consideran del centro de un nodo padre al centro de un nodo hijo. El 1.4 se dice que es porque es la raíz cuadrada de 2, y es lo que costaría ir del centro del padre al centro del hijo en diagonal. A efecto de reducir la carga del procesador al trabajar con números flotantes, multiplicamos por 10 y tenemos valores de 10 y 14 para cada uno de los movimientos como se describió.

* H representa el costo de ir desde el nodo vecino actual hasta el nodo final B, lo que se conoce como la heurística. Rápidamente, la heurística es una suposición puesto que no tenemos certeza de la distancia hasta que encontramos el camino. Además, la heurística no distingue entre nodos transitables (libres) o no transitables (obstáculos), puesto que lo que está suponiendo es la distancia en sí que falta para llegar a la meta, sin importar el estado del nodo.

* F representa la suma de estos costos. Es muy importante, ya que determina el nodo a elegir en la corrida del algoritmo en busca de la solución y va guiando al algoritmo para elegir el mejor de los caminos como se verá más adelante

Existen diferentes métodos para calcular H. Nosotros decidimos emplear del método Manhattan[3,14], donde se calcula el número total de nodos que tienen que ser recorridos horizontal y verticalmente para llegar al nodo final B desde el nodo actual, sin importar si se cruza un obstáculo. Luego se multiplica el total por 10. Al final de este capítulo se presentan un poco más a detalle el aspecto de las heurísticas.

La lista abierta se debe mantener ordenada todo el tiempo para localizar rápidamente el nodo con la F más baja y que es el que va determinado el camino final a seguir. Nosotros mantenemos la lista ordenada de la forma siguiente: cada vez que insertamos un nodo en la lista abierta, lo comparamos con el último en ella de manera tal que al inicio de la lista tenemos al nodo con la F más alta y al final de ella tenemos al de la F más baja. Esto es muy útil pues en el peor de los casos haremos N-1 comparaciones y en el mejor de ellos basta con una sola comparación para saber si es necesario o no cambiar el último nodo insertado en la lista. Al buscar basta fijarnos al final de la lista para obtener el nodo deseado.

Un caso que se presenta con frecuencia es cuando dos nodos tienen la misma F. Para nosotros no representa mayor problema ya que al insertar un nodo y comparar su F con la del último nos damos cuenta que son iguales y que no es necesario hacer un cambio. De hecho, el elegir uno u otro sólo afecta en el aspecto en que esto determina la ruta a elegir y, de cualquier manera y como se verá más adelante, siempre se encuentra el mejor de los caminos.

El algoritmo A* es como se describe a continuación:

1. Añadir el nodo inicial A a la lista abierta. Por ser el inicial hacemos su F=0 y su padre es un valor nulo, ya que es el inicio de la solución.
2. Mientras la lista abierta no esté vacía, repetir lo siguiente:
    a) Buscar en la lista abierta el nodo con la F más baja y lo consideramos como el cuadro(nodo) padre actual.
    b) Sacamos al nodo padre de la lista abierta y lo metemos en la lista cerrada.
    c) Si ese nodo es nuestro objetivo (el nodo final B), detenemos la búsqueda y generamos el camino; de lo contrario seguimos.
    d) Obtenemos los nodos vecinos de ese nodo. Para cada uno de ellos ese será el nodo padre.
    e) Si ese nodo vecino es un nodo no transitable o si ya se encuentra en la lista cerrada, lo ignoramos.
    f) De lo contrario revisamos si el nodo vecino no se encuentra en la lista abierta, se añade y se guardan su F, G y H. Si ya se encontraba, revisamos si es ahora un mejor camino que antes. Si no lo es no hacemos nada, pero si lo es debemos cambiar su anterior padre por el actual, con lo que tenemos que recalcular su G y por tanto su F.

Si se llega al nodo final, la forma de obtener el camino es fácil: sólo se comienza en ese nodo final y se va preguntando hacia atrás por su padre, y luego por el padre de este, y así sucesivamente hasta llegar al nodo inicial que es padre de alguien. Así obtenemos nuestro camino y ya es sólo cuestión de re-ordenarlo ya que por la forma en que preguntamos tenemos un camino del nodo final al nodo inicial. También prendemos una bandera que avise que se encontró el camino.

Si no se pudo alcanzar el camino fue porque en la lista abierta ya no quedaron nodos por visitar. En ese caso se prende una bandera avisando que no existe un camino entre esos dos puntos A y B.

Ahora bien, ¿a qué nos referíamos cuando en el paso 2.f mencionamos un mejor camino? En algún momento del algoritmo, y dependiendo de su complejidad y de dónde se encuentren nuestros nodos A y B, cuando intentemos ingresar un nodo a la lista abierta podemos encontrar que ese nodo ya existía anteriormente en ella y que por tanto ya tiene un padre. ¿Hubo un error? Todo lo contrario. Esto no hace más que indicar una posible mejor alternativa para el camino buscado. Lo que debemos hacer es comparar la G que ya tenía ese nodo con la G que tendría si su padre fuera el nodo actual. Llamémoslas G_1 y G_2 respectivamente. Entonces lo que tendríamos que averiguar es

Si $G_1 <= G_2$
- No hacemos nada

De lo contrario
- Cambiamos el padre anterior por el nuevo padre
- Hacemos $G_1 = G_2$ ya que su padre cambió

- Recalculamos F y revisamos la lista para ver si tiene que ser reordenada por el nuevo cambio

Esta parte puede ser confusa y si no se entiende bien desde el inicio puede derivar en un camino mal planeado y que no es el mejor.

Comencemos a ver la fuerza del algoritmo. Consideremos que tenemos un mapa como el de la Fig. 3.2; para hacerlo un poco más interesante le añadimos unos bloques entre el nodo inicial y final para que la búsqueda no sea tan sencilla, con lo que tendríamos algo como lo de la Fig. 3.4.

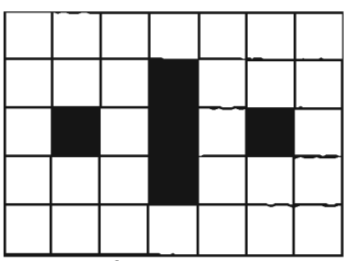


Figura 3.4 Mapa de ejemplo del A*

Siguiendo el algoritmo, al inicio tendríamos lo que muestra la Fig. 3.5

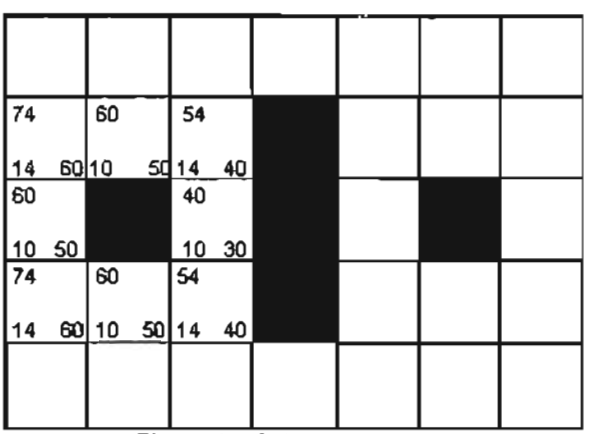


Figura 3.5 Primeros vecinos

En la izquierda superior se encuentra F, G esta debajo y H está a la derecha de G. En este caso todos los nodos apuntan al nodo A (el nodo verde) como su padre. Tomemos el nodo a la derecha: su G=10 ya que es un movimiento horizontal.; H=30 que es el número de nodos que faltan por visitar para llegar al nodo B: (3 horizontales + 0 verticales) x 10 = 30; con eso tenemos una F=40. Hacemos lo mismo con el resto de los nodos vecinos y vemos algo que ya se mencionó antes: dos nodos con la misma F. También nos damos cuenta que los nodos más alejados al destino tienen una F mayor. Así el primer nodo que sacaríamos seria el de la F=40 y después alguno de los nodos con F=54, según el orden en que

metamos los nodos en nuestra lista. Parece ser una regla comenzar a buscar los vecinos partiendo del nodo vecino encima del nodo padre.

Cuando tomemos el siguiente nodo, el de F=40, nos daremos cuenta de que los tres vecinos a su derecha son obstáculos por lo que los ignoramos. El de su izquierda es su padre y ya está en la lista cerrada, por lo que lo ignoramos también. Los cuatro restantes ya están en la lista abierta, así que comparamos si representan ahora un mejor camino que antes; nos damos cuenta que no es así, por lo que no hacemos nada. Aunque aparentemente no hicimos nada en este punto, ya que no se añadieron nodos a la lista abierta ni se encontró un mejor camino, en realidad si se hizo algo: sacar el nodo con F=40 de la lista abierta y lo metimos en la cerrada, por lo que al regresar a buscar el nodo con la F más baja en la lista abierta elegimos el que tienen la siguiente F que es 54. Por el orden en que los metimos, el que se elegiría es el que está de bajo del de F=40. Avanzando más, tendríamos la Fig. 3.6.

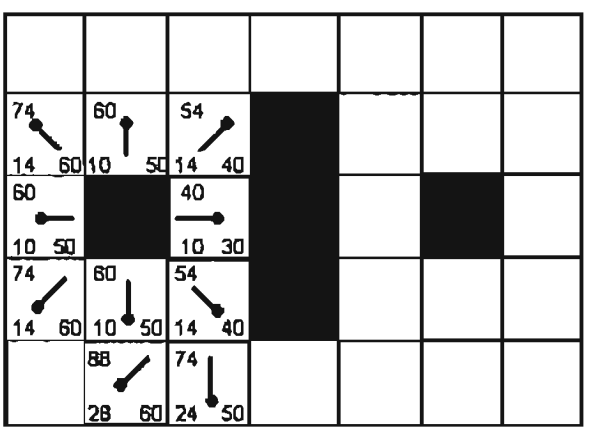


Figura 3.6 Algoritmo A* en una etapa más avanzada

Ponga particular atención en el segundo nodo debajo del inicial: Tiene una G=28 y su padre está arriba ala derecha en diagonal Ahora revise la Fig. 3.7.

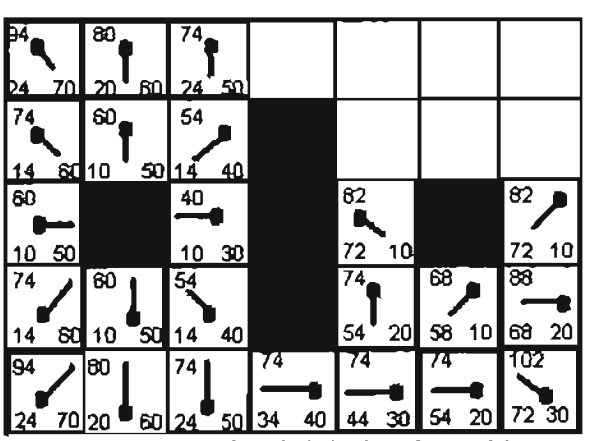


Figura 3.7 Final del algoritmo A*

¿Qué pasó? Su G ahora es de 20 y su padre es el que está sobre él. Esto pasó en algún momento en la corrida del algoritmo, donde se comparó la G que tenia con su anterior padre y la que tendría con el nuevo padre. Antes era de 28 (G_1) y con el nuevo padre sería de 20 (G_2), que es 10(la G del padre) + 10 del movimiento vertical. Como 28 no es ni menor ni igual que 20, cambiaron el padre y por ende la G y F de ese nodo. En este caso no afectó al camino final, pero en muchos otros casos un aspecto como este hace la diferencia.

Como decíamos, el camino final se obtiene al ir preguntando por el padre del nodo final hasta llegar al nodo inicial, como se ve en la Fig. 3.8. Note también que en nuestro camino no incluimos el nodo inmediatamente a la derecha de nuestro nodo inicial, que fue con el que comenzamos a buscar el camino. Esto pasó porque el nodo que está debajo de este tiene como padre también el nodo inicial, y cuando vamos preguntando por el padre de cada nodo para obtener el camino nos detenemos en cuanto un nodo tiene como padre al nodo inicial.

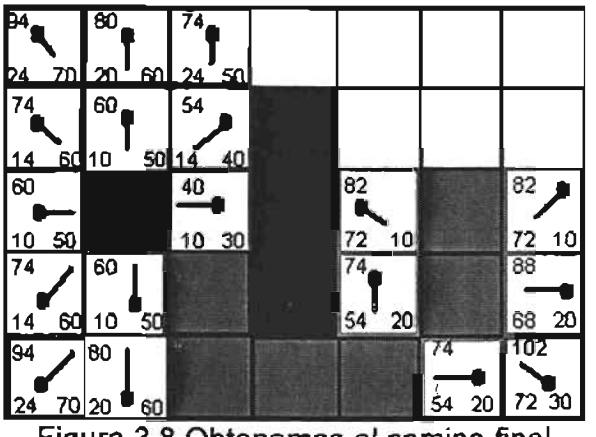


Figura 3.8 Obtenemos el camino final

Hay que tener presente que al permitir movimientos diagonales en nuestra búsqueda es posible que se hagan movimientos inapropiados.; por ejemplo, si el primer nodo vecino que visitamos en nuestra búsqueda hubiera sido un bloque negro (un obstáculo) el camino sería el mismo, pero estaríamos haciendo una especie de movimiento fantasma, ya que al momento que nuestro personaje siguiera el camino necesariamente atravesaría la esquina de ese bloque, por lo que estaría atravesando el objeto que se encontrara ahí.

La Fig. 3.9 muestra este caso. Note como aunque el camino existe y es el mismo que habíamos encontrado presenta el problema de que el borde del obstáculo debería impedir el paso de nuestro personaje para evitar un movimiento inadecuado al seguir el camino.

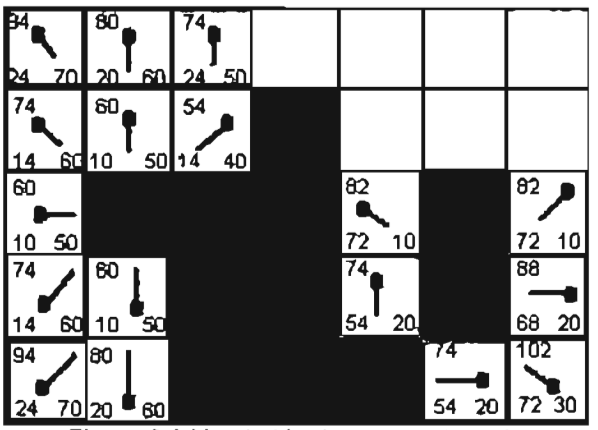


Figura 3.9 Un obstáculo en nuestra ruta

Considerando que un movimiento así no debe ser posible para lograr una ruta más realista, nosotros hemos agregado las siguientes modificaciones al algoritmo :

1. Un movimiento diagonal sólo es permitido si sus dos nodos vecinos inmediatos son libres. La Fig. 3.10 muestra para nuestro nodo inicial (cuadro verde) los vecinos diagonales permitidos, enmarcados en verde (a su izquierda), y los nodos diagonales prohibidos, enmarcados en rojo y marcados con una X (a su derecha).

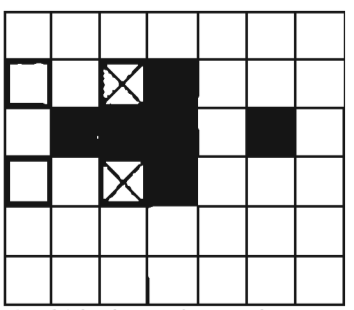


Figura 3.10 Vecinos diagonales permitidos

2. Para ayudar al algoritmo en la decisión de qué vecino considerar cuando se presenta el caso de dos vecinos con la misma F, a la heurística le sumamos el valor absoluto de la diferencia horizontal entre la meta y ese nodo vecino. De esta forma aumentamos el peso para un nodo y lo disminuimos para el otro, consiguiendo la F más pequeña. Nuestra función quedaría entonces como sigue:

$$F = G + (H + abs(yF-yP))$$

Donde yF es la posición horizontal de nuestro nodo final, lo que corresponde a la posición en Y del nodo en nuestra matriz que nos representa nuestro entorno como ya vimos. Así mismo yP representa la posición horizontal de nuestro nodo vecino actual.

Cabe realzar que si bien esta redefinición de la función ayuda para muchos casos que se puedan presentar con nodos con la misma F, para otros no habrá una diferencia significativa cuando la diferencia sea la misma para cada yP de nuestros vecinos.

Veamos algunos ejemplos del algoritmo.

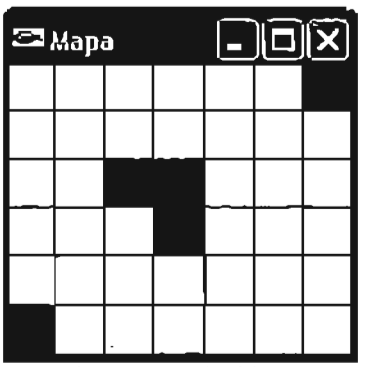


Figura 3.11a Problema

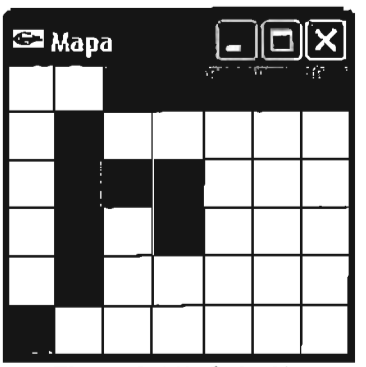


Figura 3.11b Solución

Si en este mismo problema colocáramos un obstáculo en algún nodo por donde ahora sabemos cruza el camino encontrado, tendríamos algo como lo siguiente

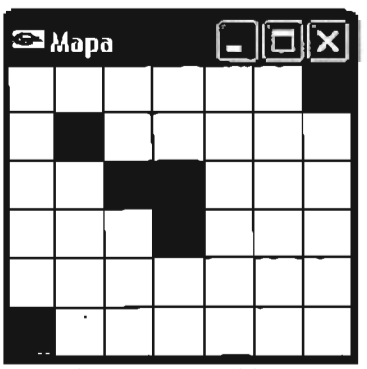


Figura 3.12a Problema

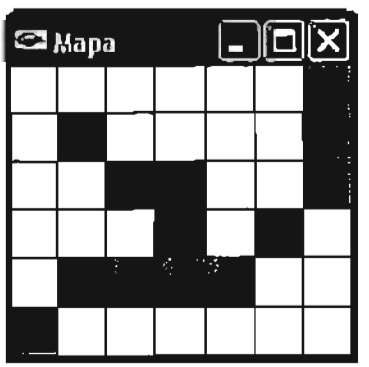


Figura 3.12b Solución

El algoritmo encuentra el nuevo camino , que como vemos es un nodo más corto que el que encontró antes de incluir el nuevo obstáculo.
Esto nos dice algo muy importante. El algoritmo es adaptable al mapa, el cuál puede ser cambiante en tiempo real. Podríamos tener obstáculos móviles en nuestro entorno que afectarían al algoritmo según se muevan por nuestro mapa.

Las figuras 3.13 y 3.14 muestran más ejemplos de la aplicación del algoritmo. Particularmente el mapa de la figura 3.14 es muy inusual y no obstante el algoritmo funciona perfectamente.

Se necesitó menos de un segundo para que el algoritmo encontrara el camino en cada caso, y consideremos también lo que toma para que se dibuje la ruta encontrada y la diferencia en tamaño de cada mapa.

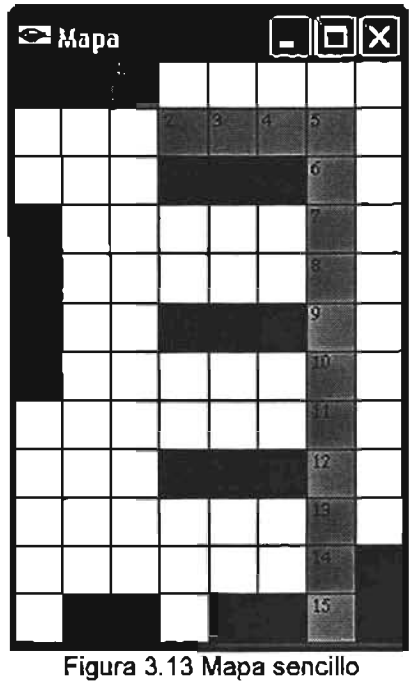


Figura 3.13 Mapa sencillo

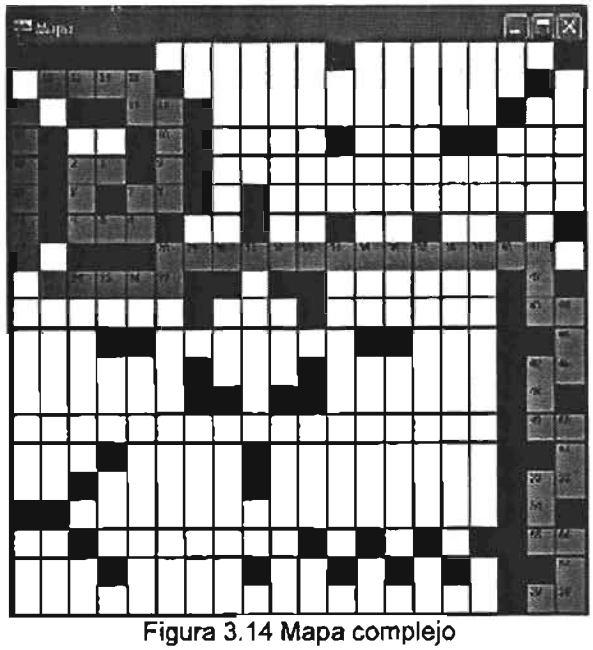


Figura 3.14 Mapa complejo

3.3 A* contra Dijkstra

El algoritmo de Dijkstra se considera como una versión menor del algoritmo A*, ya que en el primero no aparece nunca ninguna heurística, por lo que se podría considerar que H es constante con un valor de cero. La heurística hace la gran diferencia.

Una pregunta clave es: ¿Qué tanta diferencia en cuanto a costo computacional existe entre uno y otro algoritmo?

Para resolverla nos auxiliamos de un applet en Java que se encuentra en la red realizado por Bryan Stout[1] donde podemos utilizar el método de Dijkstra y el A* con la heurística dada por el método Manhattan; decidimos compararlos y encontramos que para el caso de las figuras 3.15 y 3.16 que son el mismo problema y representan al A* y Dijkstra respectivamente; al utilizar el algoritmo A* se necesitaron 25 segundos, incluyendo el tiempo para ir dibujando las etapas del algoritmo y el camino final, mientras que para el caso del algoritmo de Dijkstra se necesitó de 3 minutos 20 segundos.

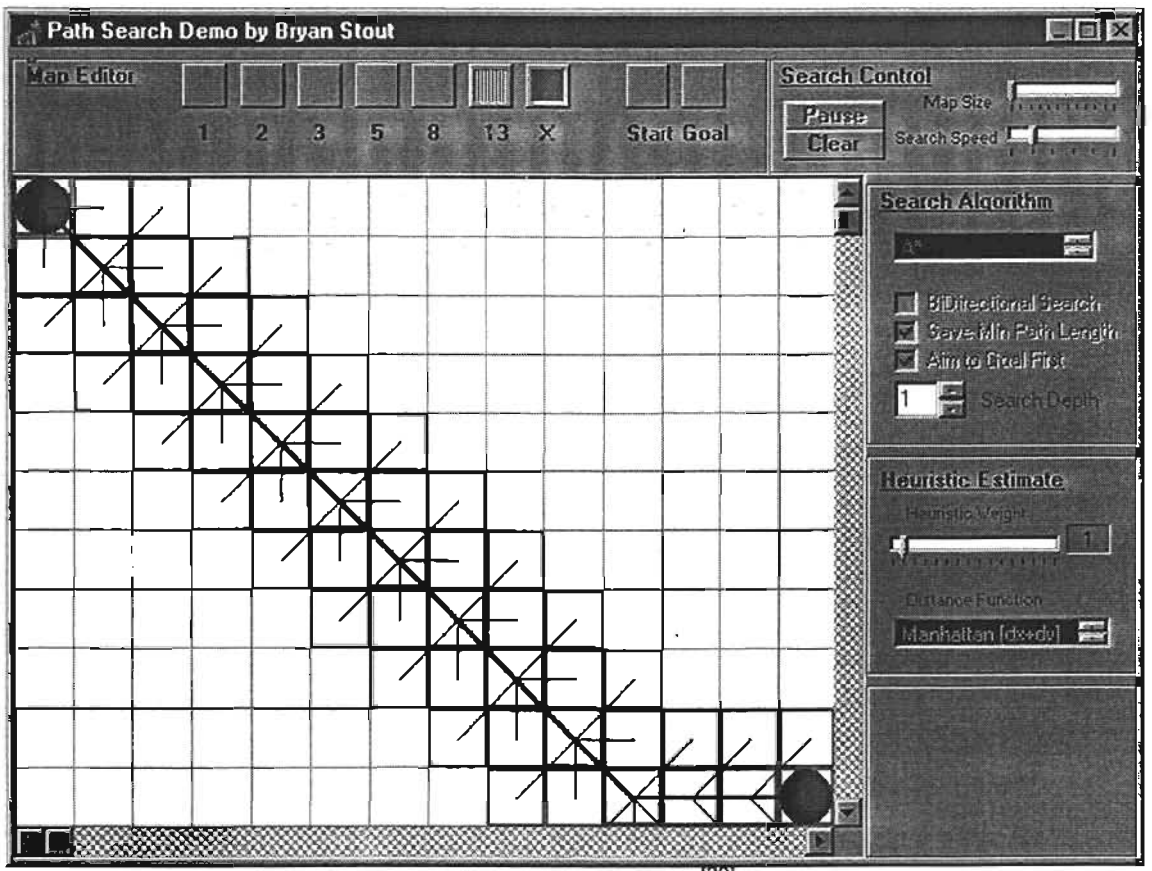


Figura 3.15 Algoritmo A* [30]

[1] http://www.gamasutra.com/features/19970801/pathfinding.htm

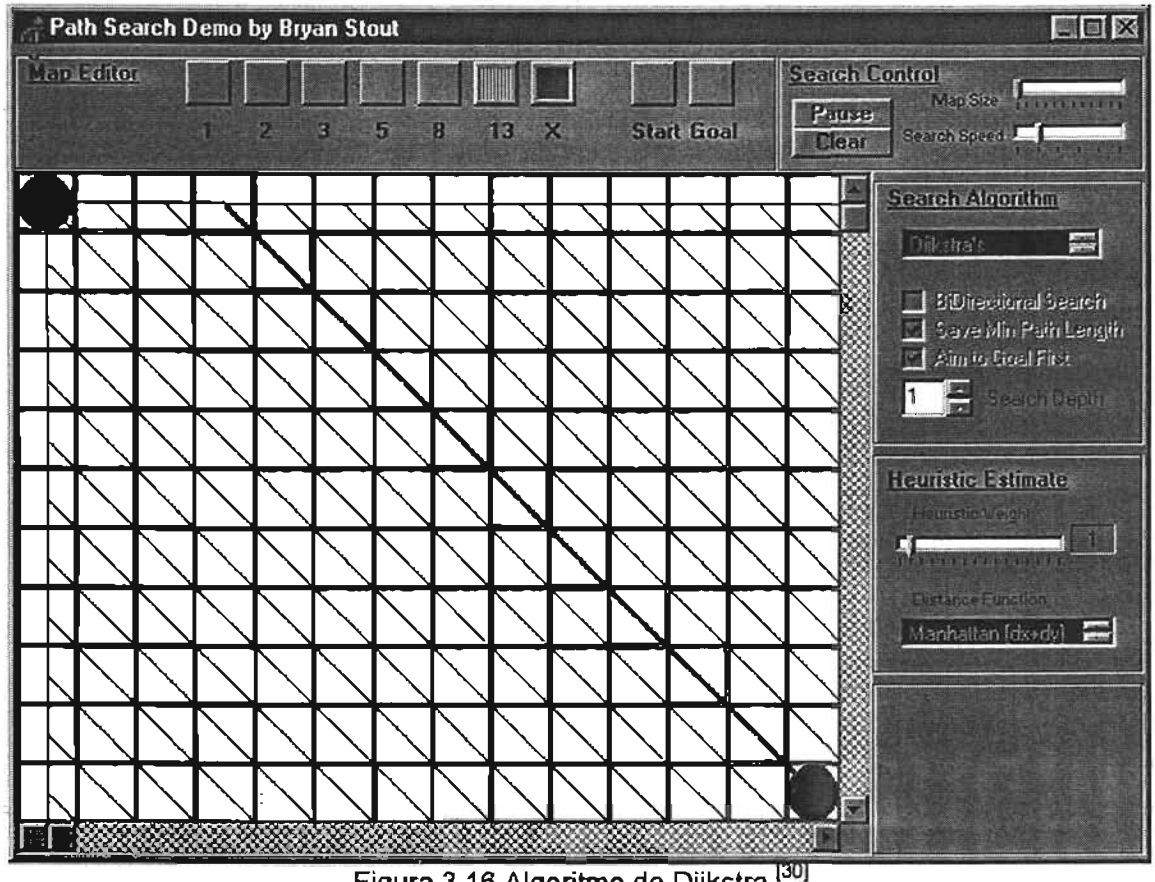


Figura 3.16 Algoritmo de Dijkstra [30]

El por qué de la diferencia se ve claramente. Mientras que en A* sólo se fueron comparando los nodos que más posiblemente llevaban a la solución, en Dijkstra no se termina de comparar cuando ya se encontró la solución, el algoritmo se detiene cuando ha revisado todos los nodos y es hasta entonces que puede determinar el mejor camino entre los dos puntos. Fue por ello que elegimos el algoritmo A* como nuestro algoritmo para solucionar el problema de la búsqueda de caminos, y por eso le dedicamos la mayor parte de este trabajo a su desarrollo y entendimiento.

3.4 Heurísticas

Como ya se vio, la heurística*[3.14] es una de las partes importantes del algoritmo A*. A continuación presentamos las dos que principalmente se emplean con este algoritmo, aunque no son las únicas.

Distancia Manhattan (Manhattan Distance)

Es la que nosotros empleamos y la que por lo general todos emplean. Se llama método Manhattan porque es como calcular el número de manzanas que hay desde un lugar a otro, donde no se puede cruzar atravesando en diagonal una manzana. Cuando calculamos H ignoramos cualquier obstáculo que intervenga. Es una estimación de la distancia que queda y no de la distancia actual, es por eso que se llama heurística, una suposición. Para conocer esta distancia debemos saber cuanto nos falta por movernos horizontal y diagonalmente desde nuestro nodo vecino actual hasta nuestro nodo final. Esto se expresa con la ecuación siguiente

$$H = (abs(f.x - p.x) + abs(f.y - p.y)) * 10$$

Donde

abs	=	valor absoluto
f.x	=	el valor en x de nuestro nodo final
f.y	=	el valor en y de nuestro nodo final
p.x	=	el valor en x de nuestro nodo vecino actual
f.y	=	el valor en y de nuestro nodo vecino actual

El multiplicar por 10 se debe a que por ser movimientos horizontales y verticales solamente, y como lo definimos al calcular G, estos tienen un costo de 10.

Distancia diagonal (Diagonal Distance)

Si en el mapa se permiten movimientos diagonales, el considerar esta H puede mejorar el rendimiento. Por ejemplo, en un mapa de 4x4 bloques el ir de la esquina (0,0) a (3,3) usando Manhattan sería ir primero hacia abajo y luego a la derecha, mientras que en este caso se podría ir directamente en diagonal. La ecuación para esta estimación está dada por

$$H = max(abs(f.x - p.x), abs(f.y - p.y))$$

En nuestro caso hemos optado por emplear la distancia Manhattan tanto para movimientos horizontales, verticales y diagonales. Esto debido a que al utilizar la distancia diagonal, el camino resultante no es el óptimo ya que resulta en una serie de movimientos zigzagueantes, como se demuestra en las figuras 3.17 y 3.18.

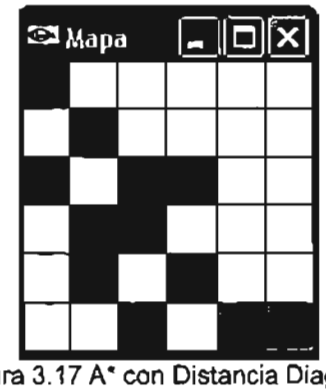


Figura 3.17 A* con Distancia Diagonal

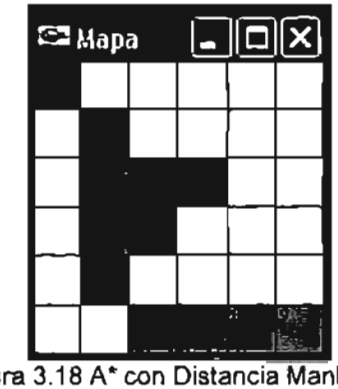


Figura 3.18 A* con Distancia Manhattan

Como vemos, con la heurística dada por Manhattan obtenemos un camino más apegado a lo que sería el movimiento de una persona o un robot quizá. Es un camino más natural.

Finalmente, el diagrama de bloques para la búsqueda de caminos quedaría como muestra la figura 3.19.

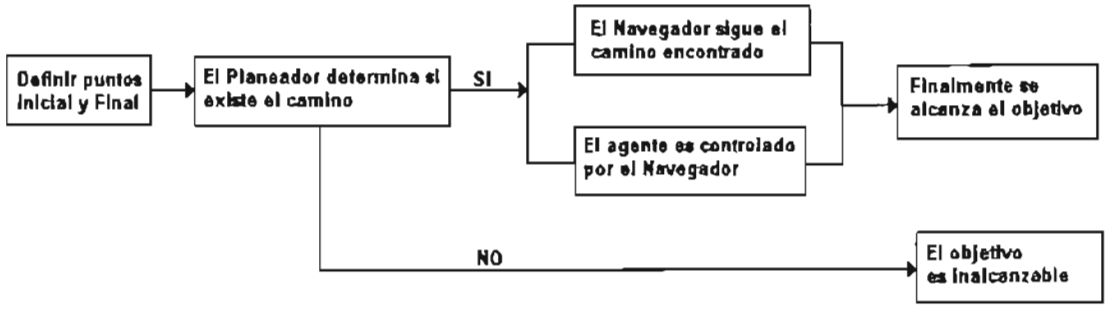


Figura 3.19 Diagrama de Bloques de la búsqueda

4. EL PASO A LO VIRTUAL

INTRODUCCIÓN

En el capítulo anterior encontramos el algoritmo que le permitirá a nuestro agente virtual moverse por su entorno, el cuál se representó de una manera bidimensional.

El siguiente paso es que nuestro agente exista en un entorno virtual antes de pensar tenerlo en un ambiente de realidad aumentada. Para ello necesitaremos varios elementos. Por un lado será necesaria una representación virtual de elementos reales, y por el otro necesitaremos encontrar la forma de tener un agente virtual que no sólo siga el camino dado, sino que además pueda caminar, es decir que incorpore movimiento en sus extremidades.

La representación del entorno puede ser tan detallada como se quiera, y en nuestro caso modelamos nuestro ambiente en 3DStudio Max[1] y mediante un proceso de lectura de los archivos .3ds lo incorporamos a nuestro programa, teniendo así el ambiente por donde se moverá nuestro agente.

Para la representación de nuestro agente nos valdremos de la herramienta de software Cal3D[19], que es una librería escrita en C que nos permite crear un personaje en 3DStudio Max y exportarlo a su formato propio para poder incorporarlo a nuestro ambiente virtual y manipular las animaciones del movimiento de nuestro agente.

Después sólo hará falta incorporar el algoritmo encontrado en el capítulo 3 para tener un escenario virtual donde habitará nuestro agente.

[1] 3Dstudio Max es propiedad de Discreet y se puede usar por un periodo de evaluación. http://www.discreet.com

4.1 Análisis y representación del entorno real

El siguiente nivel de nuestro proyecto es construir el ambiente para nuestro agente. El ambiente puede representarse con el nivel de detalle como se desee. En nuestro caso práctico lo que haremos será representar objetos reales situados en las posiciones donde no hay acceso, de la misma manera que los cuadros negros bidimensionales vistos en el capítulo 3 nos servían para la representación bidimensional.

Un aspecto importante es ver si realmente nuestro algoritmo funcionará independientemente de la representación del entorno, que a partir de este capítulo será tridimensional. Para representar nuestro entorno, el proceso se divide en dos partes. La primera es modelar nuestro entorno en 3Dstudio Max, el cuál nos permite ajustar sus unidades de trabajo a unidades métricas, por lo que podemos trabajar a escala como lo veníamos haciendo, tomando como base la representación bidimensional para comenzar a levantar nuestro ambiente. El siguiente paso es incorporar un importador de escenas en formato .3ds a nuestro proyecto.

No basta con tener la geometría descriptiva del entorno ya que esto no le dice nada a nuestro agente. Debemos tener una base de datos que nuestro agente pueda consultar de manera que se entere cuáles regiones son libres, cuáles prohibidas y en cuáles se encuentran nuestros objetos, que forman nuestra lista de destinos posibles.

En el caso bidimensional utilizamos las mismas herramientas que las que veremos a continuación. Las explicamos hasta este punto ya que la parte fuerte del capítulo 3 era encontrar nuestro algoritmo para la búsqueda de caminos.

4.2 Descripción del entorno

Recordando a partir de la figura 3.11 del capítulo 3, vemos que tenemos una representación bidimensional de nuestro entorno. Esto lo conseguimos con el editor que se puede ver en la figura 4.1. Con este editor podemos hacer varias cosas. Por un lado podemos crear un mapa o recuperar uno ya creado anteriormente, el cuál nos sirve para que nuestro sistema sepa las regiones libres y aquellas que contienen obstáculos. También podemos realizar la búsqueda de caminos entre un par de puntos arbitrarios, como se ve en la figura 4.2.

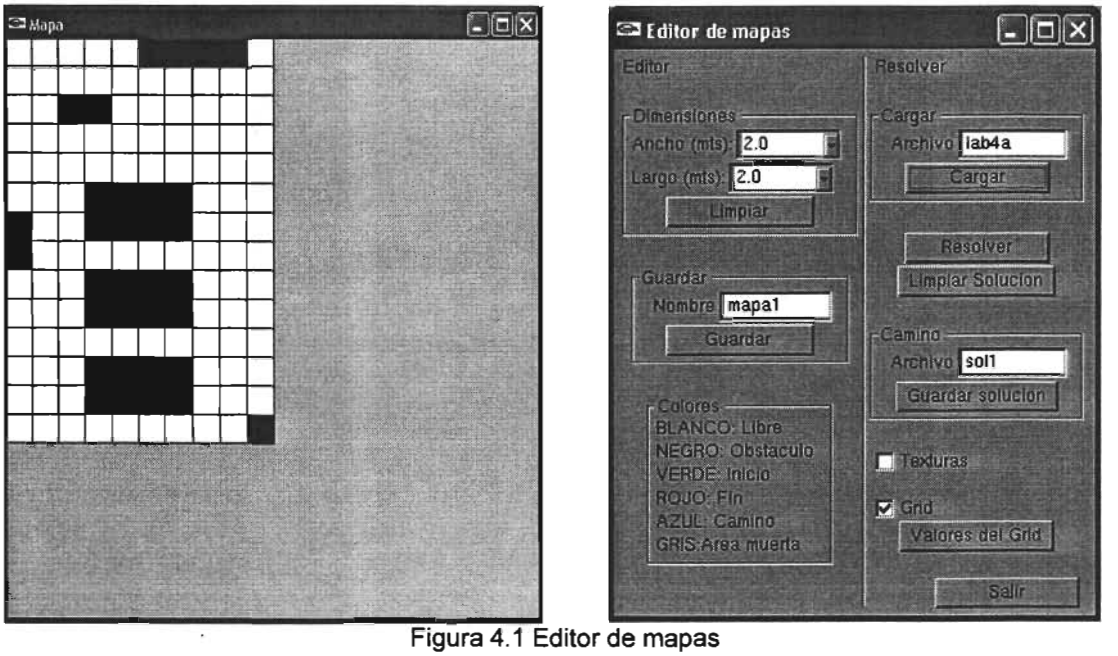


Figura 4.1 Editor de mapas

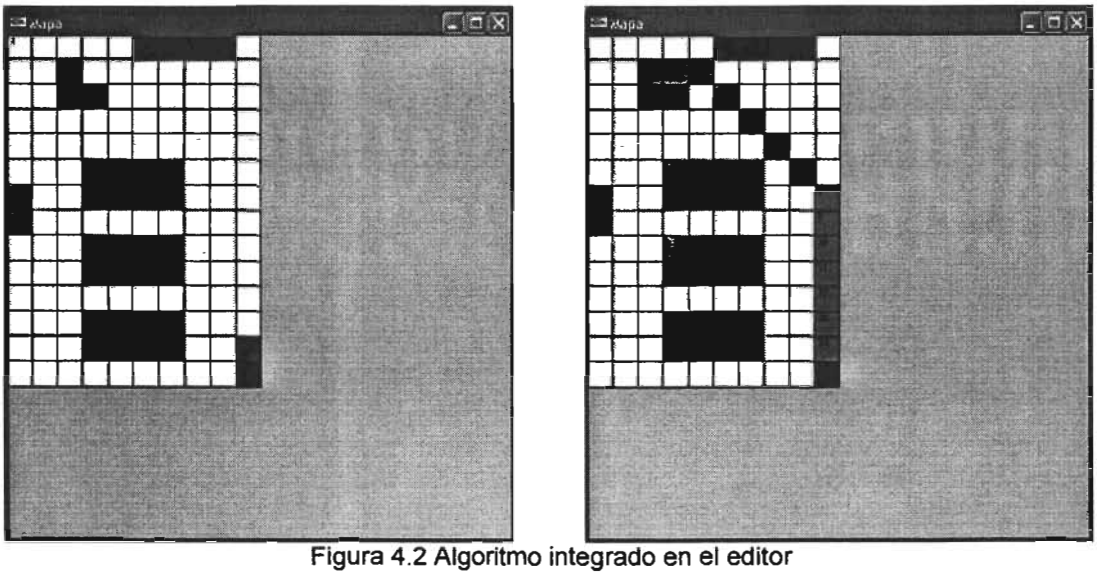


Figura 4.2 Algoritmo integrado en el editor

El área gris que rodea a nuestro mapa se utiliza cuando trabajamos con mapas más grandes que el del ejemplo. En nuestro editor cada cuadro, del color que sea, representa 50 cmts. Del mundo real como ya mencionamos en otro capítulo. Podemos tener un mapa de hasta 10 mts x 10 mts en nuestro editor, aunque se puede aumentar si se requiere.

Lo que obtenemos con nuestro editor es un archivo en texto plano con la información de la tabla 4.1

5.000000 7.000000
1 1 1 1 1 2 2 2 2 1
1 1 1 1 1 1 1 1 1 1
1 1 2 2 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 2 2 2 2 1 1 1
2 1 1 2 2 2 2 1 1 1
2 1 1 1 1 1 1 1 1 1
1 1 1 2 2 2 2 1 1 1
1 1 1 2 2 2 2 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 2 2 2 2 1 1 1
1 1 1 2 2 2 2 1 1 1
1 1 1 1 1 1 1 1 1 2

Tabla 4.1 Descripción del mapa

El formato es el siguiente: La primer línea contiene lo que sería el ancho y largo de nuestro mapa en metros, el cuál para el caso del ejemplo es de 5 mts x 7 mts. Como cada cuadro del mapa sólo nos representa medio metro, tendremos 5*2 líneas horizontales y 7*2 líneas verticales de datos. De la segunda línea en adelante, el número 2 nos indica que en esa región de nuestra matriz (recordemos que internamente nuestro sistema representa el mapa como una matriz bidimensional) existe un obstáculo, y por consiguiente el 1 nos representa una región libre.

Esto es una parte de la descripción lógica del entrono, ahora debemos conocer en dónde se encuentran nuestros diferentes elementos en la escena y que nos representan un posible destino. Para ello utilizamos el editor de objetos que muestra la figura 4.3

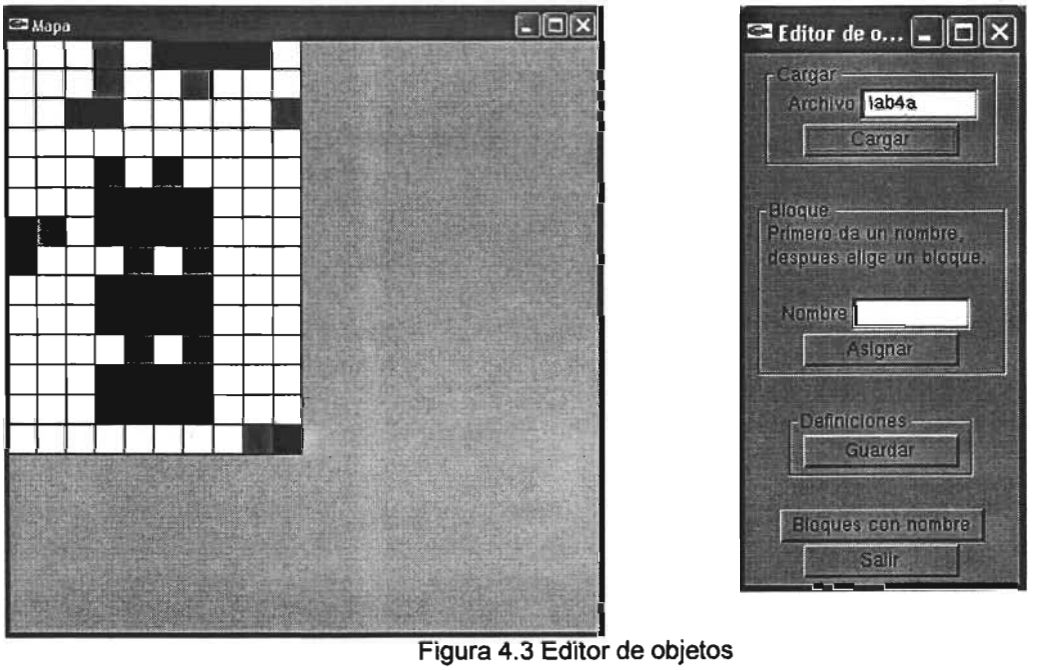


Figura 4.3 Editor de objetos

Este editor nos permite darle un nombre a cualquier región libre de nuestro mapa. Este nombre está relacionado con lo que encontramos en un punto en particular, por ejemplo un escritorio o una computadora. Hay algo que debemos tener muy presente: los objetos que estarán en nuestra escena y que representan por lo tanto un objetivo potencial cada uno de ellos, aún cuando su posición se puede marcar libremente por lo general se interpretan como obstáculos, ya que por ejemplo un escritorio no puede ser alcanzado directamente por nuestro agente puesto que no debe atravesarlo. Luego entonces, los bloques negros no sólo nos representan obstáculos tales como paredes, mesas o demás, sino que también representan a los objetos a los cuales pretendemos que nuestro agente llegue.

Entonces, ¿Cómo llegar a un objetivo si se considera un obstáculo? Para resolverlo debemos considerar que no se debe llegar al objeto en sí, sino más bien a un punto anterior inmediato que le permita a nuestro agente estar frente al objeto en cuestión. Por ejemplo, en el caso de la figura 4.4a muestra el punto 3,2 donde consideramos que se encuentra un escritorio. Para que nuestro agente llegue a él no podría hacerlo directamente puesto que es un obstáculo físico, así que cambiamos el destino al cuadro superior, que está en el punto 3,1 de nuestro mapa, como se ve en la figura 4.4b.

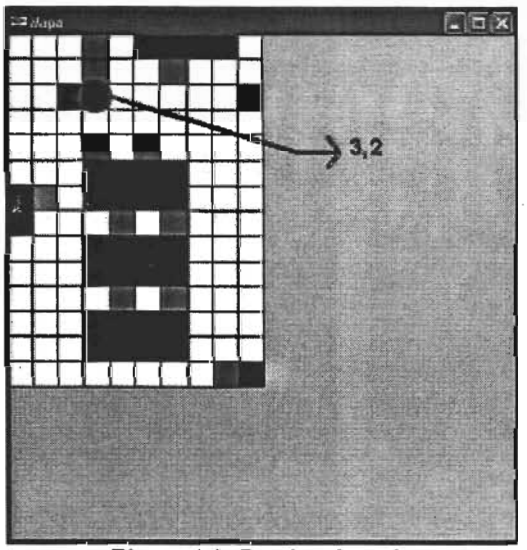


Figura 4.4a Destino deseado

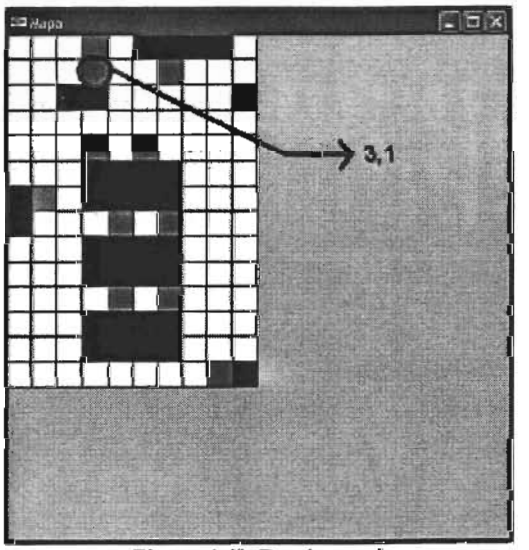


Figura 4.4b Destino real

El resto de cuadros azules representan más objetos, los cuáles están descritos en el archivo de texto plano que nos entrega nuestro editor y cuya estructura se puede ver en la tala 4.2.

La primer línea nos dice cuántos objetos tenemos en nuestra escena. Cada una de las líneas siguientes es un conjunto de tres datos que nos dan el nombre del objeto y su posición x,y dentro de nuestra matriz.

11
escritorio 3 1
librerochico 1 6
librero_grande 6 1
puerta 9 2
pc1 3 4
pc2 5 4
pc3 4 7
pc4 6 7
pc5 4 10
pc6 6 10
basura 8 13

Tabla 4.2 Descripción de los objetos

Con estos elementos ya tenemos los datos necesarios para describir a nuestro entono. Veamos ahora lo referente a nuestro agente virtual.

4.3 Descripción del agente

Hablar de un agente virtual es hablar de un avatar, de una representación tridimensional de una persona dentro de nuestro entorno simulado. Este avatar tiene características físicas, comportamientos y habilidades que permiten diferenciar un avatar de otros en un mismo escenario.
Para nuestro caso particular nuestro agente debe tener características físicas semejantes a una persona del mundo real, tales como altura, género, cabello, extremidades, etc. Nos referiremos a los comportamiento como los eventos de animación que posee, como pueden ser el caminar, el señalar, etc. Las habilidades del agente le permitirán interactuar en el entorno donde habita, como pueden ser el encontrar caminos a seguir hacia una meta en particular, el poder hablar y escuchar, etc,
Además de todo esto, el agente debe contar con una fuente de datos lo más enriquecida posible de información del mundo que habita, como lo es la descripción del entorno que vimos en el punto 4.2.

Particularmente lo relacionado al manejo de agentes (o mas conocidos como caracteres o personajes) presentan diferentes aspectos importantes que se deben considerar. Una parte importante es tener una herramienta que permita modelar a nuestro agente, pero además debemos contar con otra herramienta (o la misma usada para modelarlo, si es que lo permite) que nos permita elaborar las animaciones (comportamientos) necesarias para nuestro agente, y tratarlas externamente en nuestro sistema.

Esto nos lo permite Cal3D[19]. Cal3D es una librería escrita en C++ que nos permite manejar animaciones de personajes tridimensionales basadas en esqueletos, es decir que nuestro personaje puede tener un esqueleto que le permite determinar los movimientos que puede hacer, regido por leyes similares a las físicas en el mundo real.

Cal3D se divide en dos partes: la parte de código y la parte de plugin. Como plugin nos permite exportar los datos de nuestro agente hecho con software de modelado y animación como 3DS Max a un formato que la librería sabe interpretar. A nivel de código nos permite trabajar con esta información para manipular los datos de nuestras animaciones de la forma que lo necesitemos.

Por cuestiones de practicidad en nuestro ejemplo utilizaremos uno de los modelos que trae Cal3D consigo cuando se obtiene desde su sitio en Internet. Esto sobre todo porque si bien esta librería es muy útil, requiere de mucho tiempo para poder entender y manejar los pasos necesarios para crear correctamente nuestro personaje y exportarlo de forma adecuada. Baste con decir que la información obtenida mediante el plugin es almacenada en archivos donde se encuentra toda la información relacionada con nuestro personaje: geometría, texturas y animaciones.

Teniendo ya todos los elementos necesarios, podemos continuar con la incorporación de todos ellos a nuestro entorno virtual.

4.4 Entorno de Realidad Virtual

Es tiempo de unir todos los elementos ya revisados. Para el caso de nuestro agente utilizaremos a Cally, un personaje de ejemplo que trae Cal3D y que posee una animación de caminado. La figura 4.5 nos muestra nuestra primer imagen del entorno virtual.

Existen dos paredes con texturas de ladrillo para evitar que la escena se pierda con el fondo. Los obstáculos están representados por cubos con texturas de madera y se puede observar a Cally de lado derecho.

En este estado inicial la posición de nuestro agente (Cally) es aleatoria, escogida de entre una lista de objetos como la vista en la tabla 4.2. La animación se encuentra en estado de pausa, mismo que cambia en cuanto nuestro agente comience a moverse hacia un destino requerido. Para decirle a nuestro agente hacia donde queremos que se mueva utilizamos un menú donde se encuentran los posibles destinos. Como trabajo futuro se incorporará la parte de reconocimiento de voz para interactuar con el agente, misma que está desarrollando el Ing. Ramón Ramírez Guzmán para su proyecto de Tesis de Maestría bajo el nombre de *"Desarrollo de interfaces verbales con Agentes Conversacionales"* y que forma parte del mismo proyecto final.

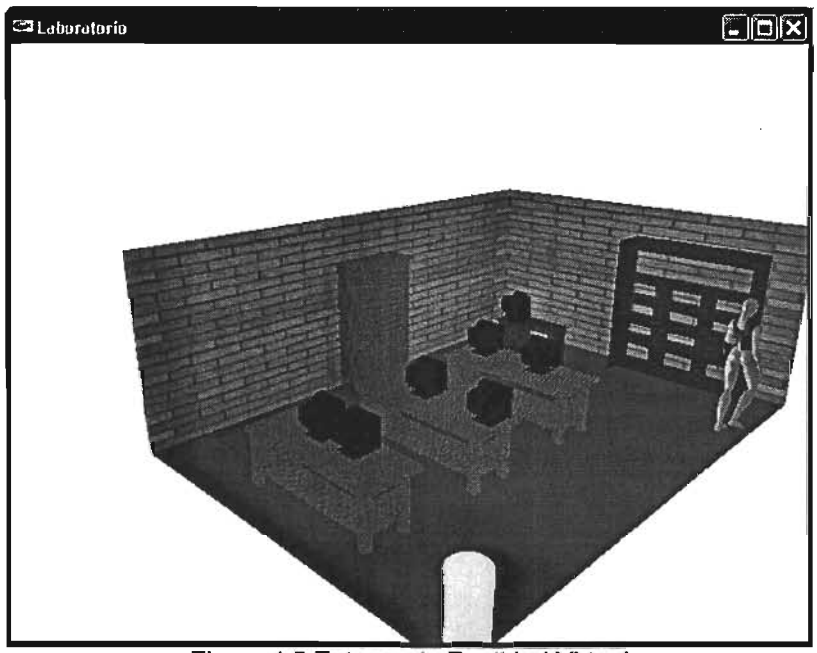


Figura 4.5 Entorno de Realidad Virtual

La figura 4.6 muestra la forma de indicarle al agente el destino y la figura 4.7 algunos puntos de la ruta que tomó al ir de pc6(a) hacia escritorio (f).

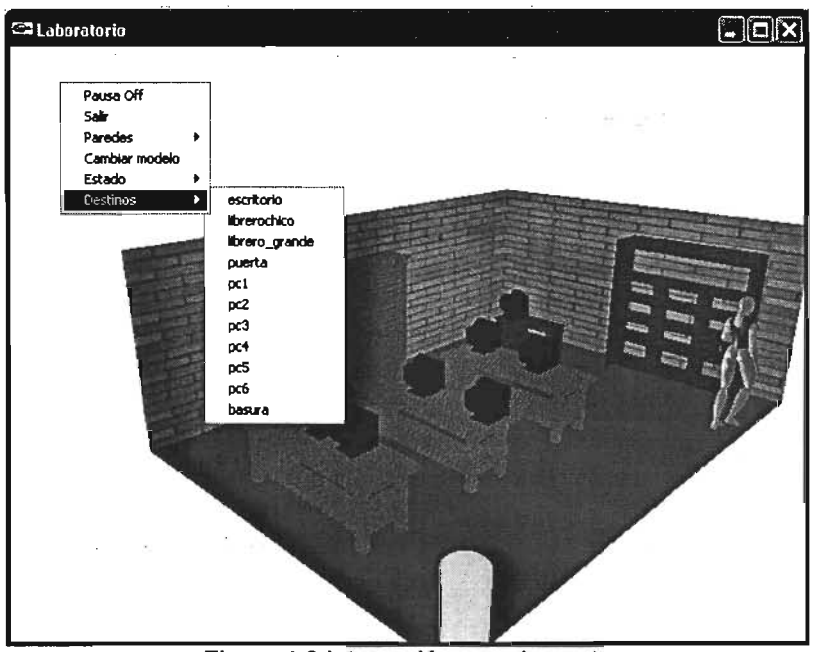
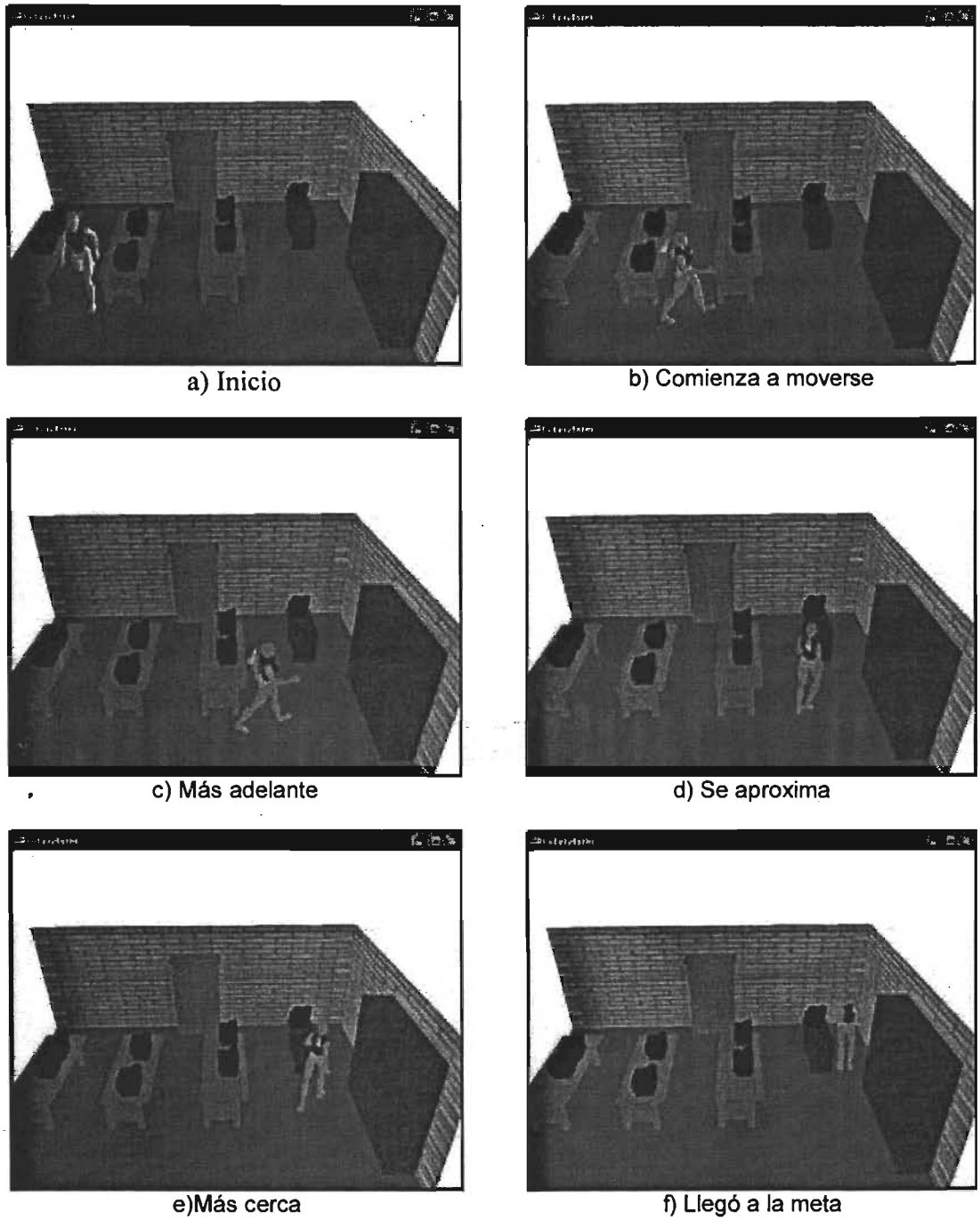


Figura 4.6 Interacción con el agente



a) Inicio

b) Comienza a moverse

c) Más adelante

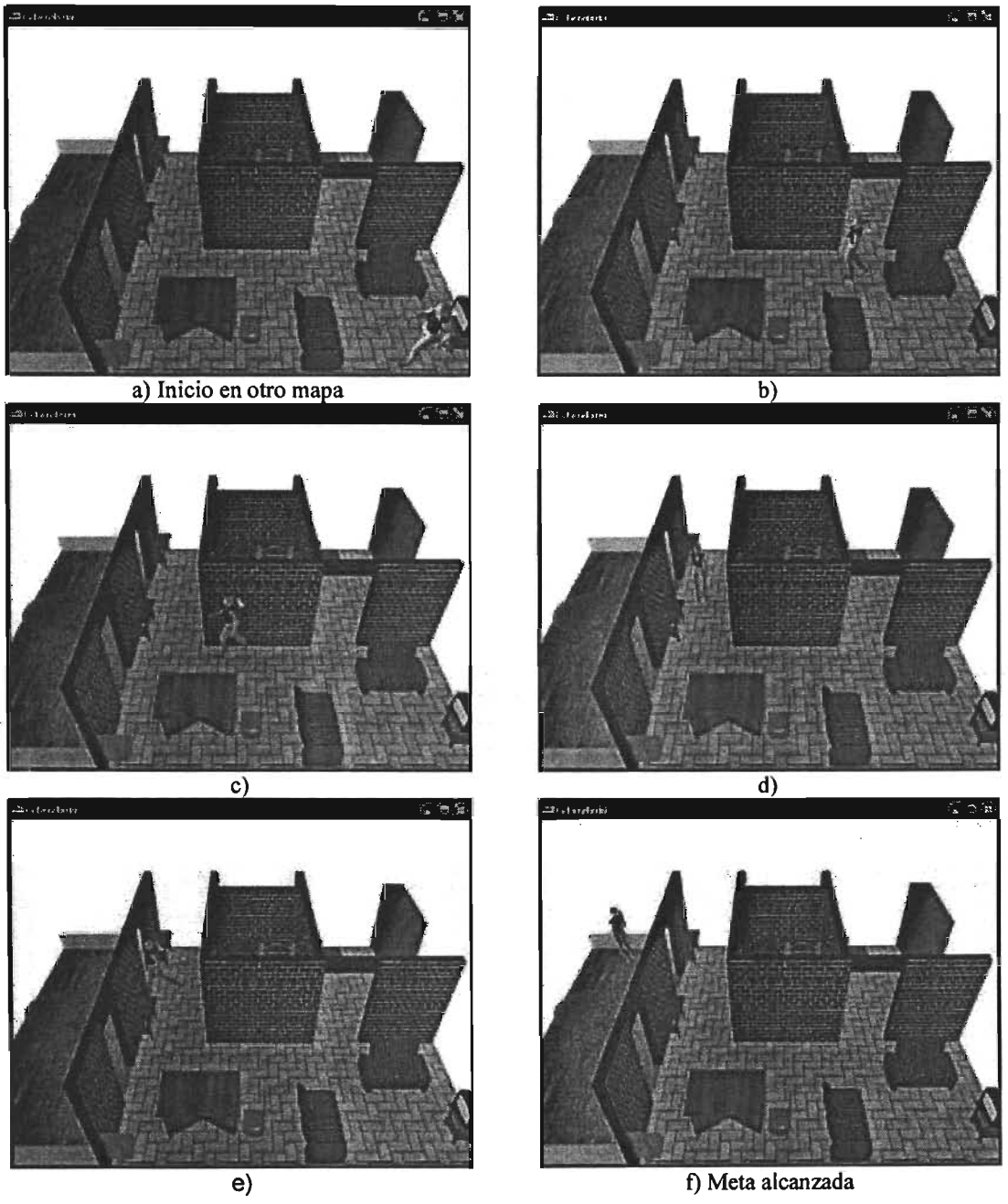
d) Se aproxima

e)Más cerca

f) Llegó a la meta

Figura 4.7 El agente sigue el camino encontrado

Así mismo ocurre cuando el entorno es un poco más complejo, como lo muestra la figura 4.8 para un cuarto de 10 mts x 7 mts.



a) Inicio en otro mapa b)

c) d)

e) f) Meta alcanzada

Figura 4.8 Otro entorno virtual

Se puede apreciar que el agente cambia su orientación mientras recorre el camino encontrado.

La forma para determinar el ángulo necesario para que el agente se oriente hacia su destino está dada por lo siguiente:

Recordemos del capítulo 3 que nuestro agente tiene permitidos tanto movimientos horizontales como diagonales, con sus restricciones pertinentes para este último caso.

Entonces, si tomamos como convención que nuestro agente siempre se encuentra observando hacia el Sur, y quisiera girar hacia el Este (lo que representaría un movimiento horizontal) el ángulo requerido θ sería de 90°. Si del Sur intentara observar al Norte θ=180° (movimiento vertical), y si observara al Sureste sería θ=45° (movimiento en diagonal). Para ver hacia el Noroeste, el ángulo sería de 135°. La figura 4.9 muestra estas reglas de orientación.

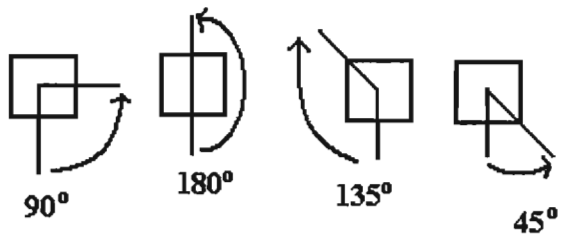


Figura 4.9 Reglas de orientación para el agente virtual

Como hemos visto todas las partes se amoldan perfectamente para trabajar en conjunto en nuestro entorno virtual, sin importar la complejidad del mismo. La parte final del trabajo aborda la problemática que surge al llevar esto a un entorno de realidad aumentada.

5. EL PASO A LO AUMENTADO

Introducción

En el capítulo anterior levantamos un entorno de realidad virtual en donde pudimos probar nuestro sistema.

Pensar en llevar eso a la realidad aumentada nos regresa a los puntos analizados en el capítulo 2. El más preocupante sin duda es el que se refiere a la alineación del mundo real con el virtual, lo cuál no se da de manera directa.

Otro punto importante fue que al utilizar el ARToolkit[15] en esta parte del proyecto, Cal3D[19] presentó problemas para incorporarlo, a diferencia del entorno virtual, por lo que utilizamos un método distinto con sus respectivas adecuaciones.

En este capítulo revisaremos entonces las situaciones que se presentaron al tratar de portar el sistema virtual al aumentado.

5.1 Cambio de agente

Al comenzar a portar las partes del sistema virtual a este ambiente aumentado lo que encontramos fue que habían conflictos entre las librerías de Cal3D[19] y ARToolkit[15]. Como ya vimos, Cal3D nos ayudó a la incorporación de nuestro agente en el entorno virtual. ARToolkit es necesario para trabajar sobre un sistema de realidad aumentada. Las diferencias entre estas herramientas se software parecían tan inconciliables que optamos por utilizar la característica que tiene ARToolkit para manejar animaciones en archivos de VRML. No obstante, el problema no se resuelve del todo con eso, ya que si bien podemos tener animaciones para nuestro agente, ARToolkit a diferencia de Cal3D no permite la manipulación de las animaciones, repitiéndolas infinitamente. Quizá las próximas versiones de Cal3D o del mismo ARToolkit resuelvan el conflicto, y por el momento la solución es la presentada arriba.

Para trabajar en ARToolkit es necesario, como vimos en el capítulo 2, utilizar un patrón que le indique al sistema dónde hacer las transformaciones de la cámara real (cámara web) para ajustarlas a la cámara virtual (la cámara de nuestro sistema). El patrón empleado es el que se ve en la figura 5.1, mientras que la figura 5.2 nos muestra este patrón visto desde la cámara web.

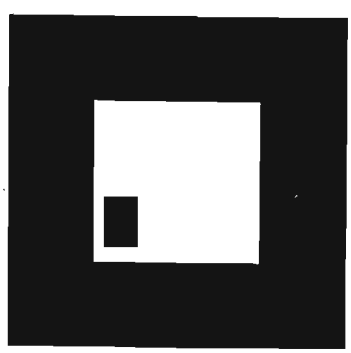


Figura 5.1 Patrón empleado

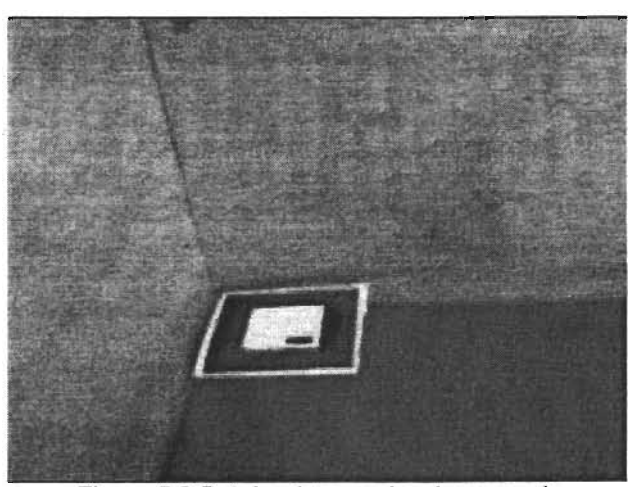


Figura 5.2 Patrón visto por la cámara web

A diferencia de nuestro entorno virtual, para el aumentado no trabajamos con archivos .3ds para representar nuestra escena; en contraste utilizamos la parte de ARToolkit que carga archivos .wrl y que podemos generar también con 3DS Max. La figura 5.3 muestra lo descrito.

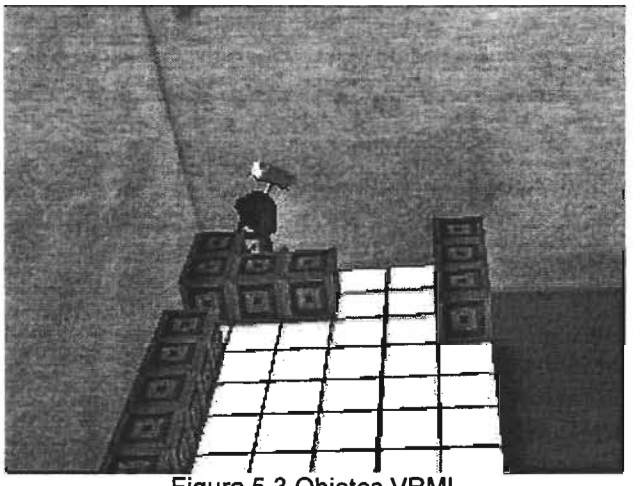


Figura 5.3 Objetos VRML

Podemos observar en la figura de arriba que nuestro personaje ya no es Cally, sino Marvin. Esto debido a los conflictos mencionados; este modelo es de uso libre en internet y lo utilizamos ya que Cally no se puede exportar puesto que su formato es el nativo de Cal3D y no de 3DS Max.

Una vez que nuestro sistema encuentra el patrón, podemos almacenar la información de la cámara y prescindir del patrón y seguir trabajando con nuestra escena, como se ve en las figuras 5.4 y 5.5.

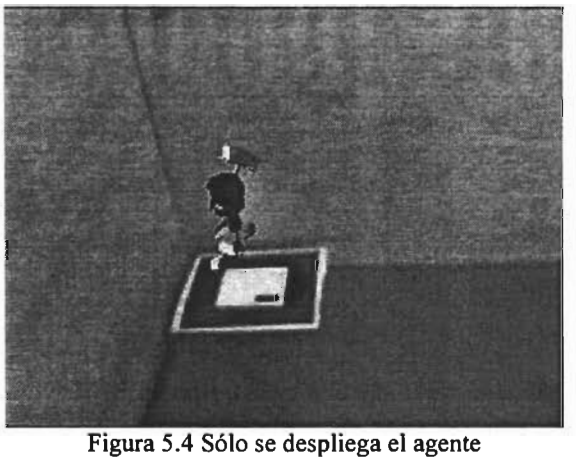
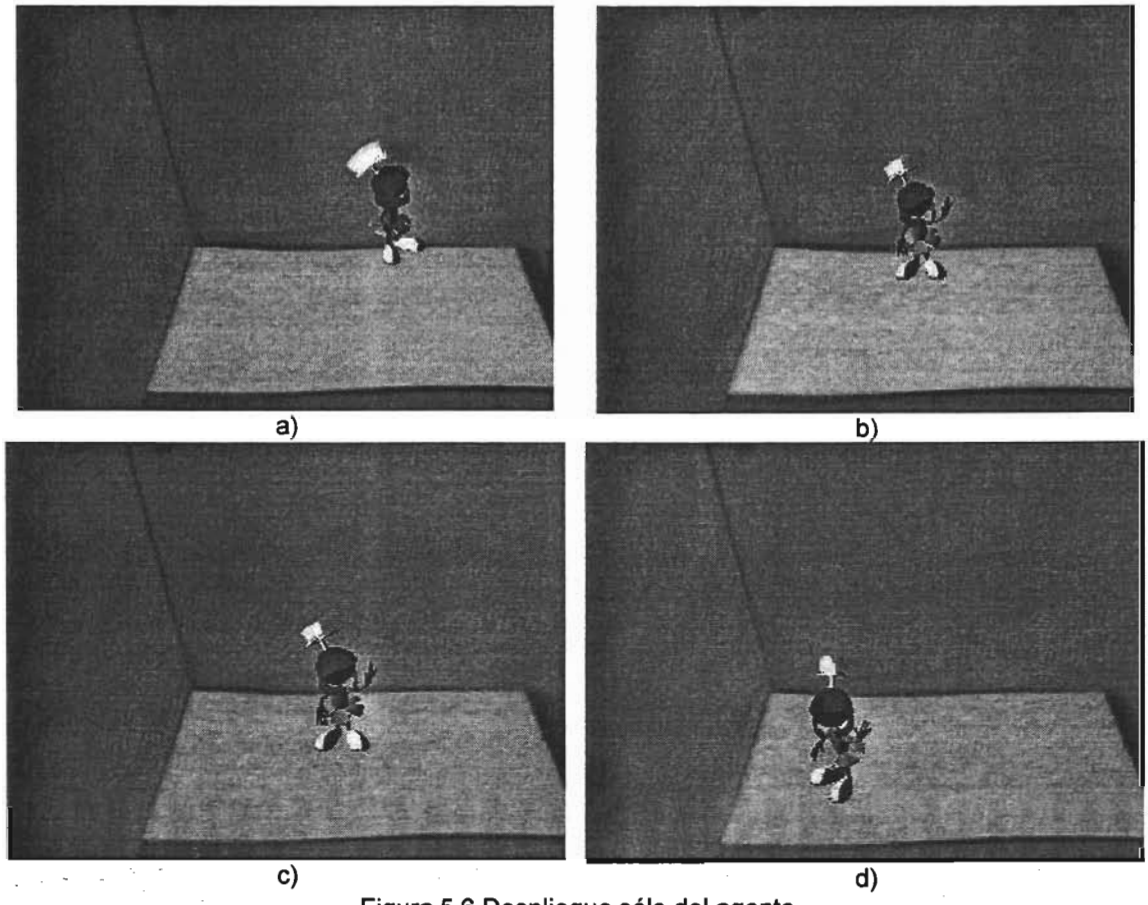


Figura 5.4 Sólo se despliega el agente

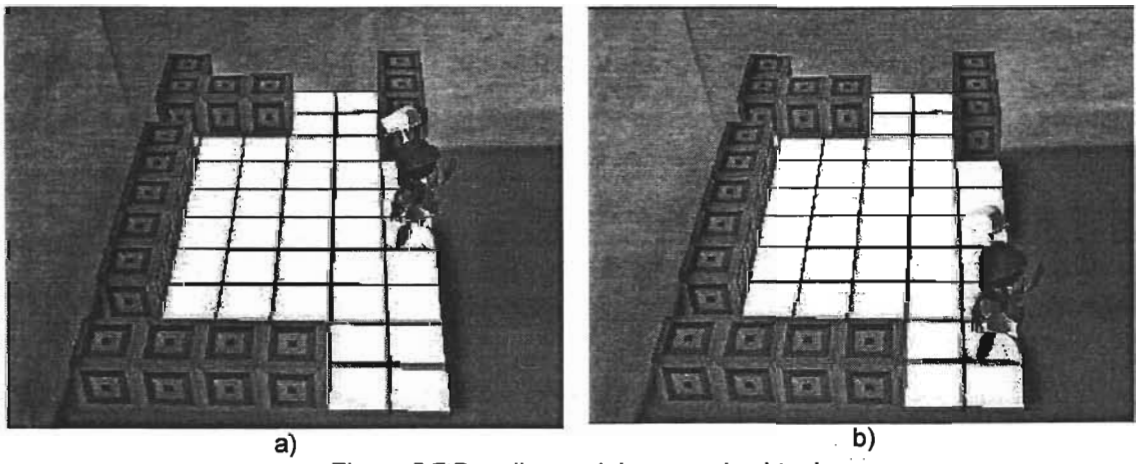
Figura 5.5 El patrón es removido

El poder desplegar solamente a nuestro agente es importante puesto que es lo único que debe ser generado por computadora, el resto de la escena debe estar formado por objetos reales. La figura 5.6 muestra al agente siguiendo un camino sin desplegar la escena virtual, y la figura 5.7 muestra cómo sería si se desplegara esa parte.



a) b) c) d)

Figura 5.6 Despliegue sólo del agente



a) b)

Figura 5.7 Despliegue del escenario virtual

5.2 Aspectos de alineación

Si observamos con atención notaremos en las figuras anteriores un problema de perspectiva. La figura 5.8 muestra al agente junto a un vaso de plástico real.

Figura 5.8 Vaso real y agente virtual

Dejando de lado el aspecto de la relación del tamaño del agente con el vaso real, puesto que estamos trabajando a escala, podemos notar que hay un problema de alineación entre la parte de la escena generada por computadora y la parte que se obtiene del mundo real. Esto ocurre por el efecto de la cámara física, la cámara de videoconferencia, que al no tener un campo de visión aproximado al del ojo humano, produce ese efecto dispar. El trabajar a escala nos permite prever los problemas que se nos presentan al intentar alinear la escena virtual con la real.

Como vimos en el capítulo 2, la alineación del mundo real con el virtual involucra aspectos tanto de hardware como de software. Por un lado están los dispositivos empleados para adquirir las imágenes, tales como los HMD o una cámara web (de videoconferencia) como fue nuestro caso, y los cuáles impactan en el área que podemos percibir mediante ellos del mundo real. Por otro lado, la alineación por software de las cámaras real y virtual puede cubrirse con el ARToolkit, pero desafortunadamente sólo resuelve parte del problema. Por todo esto, está claro que para poder avanzar más requerimos de equipo más especializado que nos permita solventar las dificultades con las que nos encontramos hasta este punto.

CONCLUSIONES Y TRABAJO FUTURO

En este trabajo hemos revisado y trabajado los aspectos necesarios para permitir que un agente virtual se mueva en diferentes entornos. Comenzamos por establecer las diferencias entre un sistema de realidad virtual y un sistema de realidad aumentada en los primeros capítulos. Posteriormente analizamos el problema de la navegación y desarrollamos el algoritmo que nos permitió encontrar la ruta más corta de un punto A hasta un destino B, siendo este un camino libre de obstáculos. Para ello utilizamos una representación bidimensional de nuestro entrono por medio de una matriz que contenía la información de nuestro mapa que describía el ambiente donde el agente se movía.

Esta representación bidimensional es muy importante, ya que al no hacer ningún tipo de despliegue gráfico para analizar el entorno por donde se moverá el agente, nos vemos beneficiados en tiempo y costo computacional, lo que dota al algoritmo de una gran velocidad de respuesta en tiempo real. Hay que recordar que este proceso es sólo una parte del todo final, es decir, existen más procesos en el sistema (como el de la captura del video de la escena real, el despliegue gráfico de los objetos virtuales, etc.) y por lo tanto la velocidad con la que se realicen cada uno de ellos oportunamente afectará positivamente el desempeño de nuestro sistema.

También realizamos una representación tridimensional básica de nuestro entorno, al cuál le aplicamos nuestro algoritmo y determinamos los criterios de orientación de nuestro agente mientras va siguiendo el camino encontrado por el planeador.

Llegamos a la parte de realidad aumentada. El primer problema al que nos enfrentamos fue la imposibilidad de incorporar el agente de la misma manera que en el entorno virtual, es decir no pudimos utilizar Cal3D como en el primer escenario, así que tuvimos que cambiarlo por la característica que nos brinda ARToolkit de utilizar objetos en VRML. Esto nos afectó en la forma en que manipulamos las animaciones, no obstante el algoritmo al ser independiente se adapta a esta nueva forma de trabajo.

Finalmente nos encontramos con el gran problema de la alineación de los mundos, el cual sigue siendo un reto a vencer. Basándonos en todo lo investigado al respecto, la solución más factible hasta el momento parece ser la utilización de patrones junto con el ARToolkit para encontrar una especie de unidades de medida que nos permitan hacer una correspondencia entre las dimensiones de los objetos del mundo real y los objetos generados por computadora, empatando así ambos mundos lo mejor posible, ya que siempre existe el factor de ruido al sobreponer las escenas.

No obstante las limitantes, las aportaciones hechas por este trabajo son muy importantes. Por un lado en el entorno virtual funcionó perfectamente, y con un poco más de trabajo se pueden desarrollar laboratorios virtuales donde nuestro agente pueda interactuar con el usuario. Incluso podría ser posible que existan más de un agente en nuestro entorno, interactuando entre ellos y con el usuario, para lo cual se tendrían que definir e incorporar las características de cada uno de ellos que los hacen diferentes entre si, dándole variedad al usuario.

Con las adecuaciones pertinentes, el algoritmo de búsqueda de caminos aquí desarrollado puede emplearse en robots móviles que se encuentren en un escenario tal como un laboratorio real, aunque no se limita a esa posibilidad.

Por tanto, los objetivos planteados inicialmente fueron cumplidos. Como trabajo futuro queda entonces el resolver el problema de la alineación de los mundos, así como considerar obstáculos móviles como puede ser el mismo usuario al estar interactuando directamente con el agente virtual en el laboratorio real.

GLOSARIO

Avatar: Representación tridimensional de una persona real en un ambiente virtual o aumentado.

Engine: Parte principal de un sistema de videojuegos que se encarga de unir y controlar todos los elementos que intervienen en él.

FPF: (*Frames per Second*). Son los cuadros por segundo que despliega una computadora, por ejemplo para un videojuego o una escena tridimensional.

GPL: (*General Public License*) Licencia que permite que el código fuente de software sea libre y disponible para quien lo desee.

GPS: (*Global Positioning System*). El Sistema de Posicionamiento Global está formado por una red de 24 satélites denominada NAVSTAR, situados en una órbita a unos 20.200 km. de la Tierra, y unos receptores GPS, que permiten determinar nuestra posición en cualquier lugar del planeta a cualquier hora y en cualquier condición meteorológica.

Head/Pitch/Roll: Define la orientación del usuario en su espacio local.

HMD: (*Head Mounted Display*) Dispositivo de despliegue a manera de casco o lentes donde se proyecta la salida en video de una escena generada por computadora. Generalmente estos cascos son cerrados y de color opaco, evitando que se vea algún elemento del exterior con la finalidad de sumergir completamente al usuario en el ambiente virtual donde se utilicen.

HMD see-through: Casco HMD que son parcialmente transparentes. Esto permite al usuario percibir tanto el mundo real como la parte generada por computadora.

Navegación dirigida: Se refiere a la forma en que se encuentra un camino desde un punto inicial hasta un punto final u objetivo, el cuál será seguido por un carácter animado.

Tracking: Se refiere a las herramientas empleadas para seguir la posición y orientación del usuario.

Virtualidad Aumentada: Se refiere a los sistemas donde se trabaja sobre el mundo virtual incorporando algunos elementos del mundo real.

VRML: (*Virtual Reality Modeling Languaje*). Es el lenguaje utilizado en Internet para crear desde simples objetos tridimensionales hasta escenarios más elaborados.

BIBLIOGRAFÍA

[1] Kuffner, James J. Jr., Goal-Directed Navigation for Animated Characters Using Real-Time Path Planning and Control. En *Lecture Notes In Computer Science*, Vol 1537, 1998, pág. 171-186.

[2] James J. Kuffner, Jr. Autonomus Agents for Real-Time Animation.2002. En su trabajo de Tesis, Diciembre de 1999, pág. 31-66.

[3] Strategy Game Programming with DirectX 9.0. Todd Barron, Wordware Publising, Inc. 2003. pág. 248-273

[4] Thomas, Bruce; Close, Ben; Donoghue, John; Squires, John; De Bondi, Phillip; Morris, Michael; Piekarski, Wayne. ARQuake: An Outdoor/Indoor Augmented Reality First Person Application. En *Proceedings of the 4th IEEE International Symposium on Wearable Computers*, 2000, pág. 139-146.

[5] Thomas, Bruce; Krul, Nicholas; Close, Benjamin; Piekarski, Wayne. Usability and Playability Issues for ARQuake. En *First International Workshop on Entertainment Computing*, 2002, pág. 447-454.

[6] Piekarski, Wayne; Thomas, Bruce. Bread Crumbs: A Technique for Modelling Large Outdoor Ground Features. En *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2002, pág. 321.

[7] Piekarski, Wayne; Thomas, Bruce. Augmented Reality with Wearable Computers Running Linux. En *2nd Australian Linux Conference*. Sydney, NSW, Enero de 2001.

[8] Billinghurst, Mark; Poupyrev, Ivan; Kato, Hirokazu; May, Richard. Mixing Realities in Shared Space: An Augmented Reality Interface for Collaborative Computing. En *IEEE International Conference on Multimedia and Expo (III)*, 2000, pág. 1641-1644.

[9] Billinghurst, Mark; Campbell, S.; Hendrickson, D.; Chinthammit, W. Magic Book: Exploring Transitions in Collaborative AR Interfaces. En *SIGGRAPH 2000 Emerging Technologies Proposal*, 2000.

[10] Malik, Shahzad; Roth, Gerhard; McDonald, Chris. Robust 2D Tracking for Real-Time Augmented Reality. En *The 15th International Conference on Vision Interface*, 2000, páginas 309-406.

[11] Piekarski, Wayne; Thomas, Bruce. Augmented Reality User Interfaces and Techniques for Outdoor Modelling. En *SI3D*, 2003, pág. 225-226.

[12] Billinghurst, Mark; Kato, Hirokazu. Collaborative Augmented Reality. En *Communications of the ACM*, 2002, pág. 64-70.

[13] Hirokazu Katoa, Mark Billinghurst, Ivan Poupyrevg. Manual para ARToolKit, versión 2.33, año 2000.

[14] Game Programming Gems. Mark DeLoura, Charles River Media, Inc., 2000. pág. 254-297

REFERENCIAS EN INTERNET

[I5] ARToolkit
http://www.hitl.washington.edu/research/shared_space/

[I6] MagicBook
http://www.hitl.washington.edu/magicbook/

[I7] ARQuake
http://wearables.unisa.edu.au/projects/ARQuake/www/

[I8] Tinmith
http://www.tinmith.net/

[I9] Cal3D
http://cal3d.sourceforge.net/

FUENTES DE IMÁGENES

[20] http://info.pue.udlap.mx/~tesis/lis/von_r_pa/capitulo1.pdf
[21] http://www.cs.brown.edu/courses/cs196-2/lectures/29Jan02.pdf
[22] http://www.hitl.washington.edu/research/shared_space/
[23] http://www.cs.duke.edu/courses/fall02/cps124/projects/room/,
http://www-vrl.umich.edu/sel_prj/EECS498/,https://java3d.dev.java.net/,
http://techpubs.sgi.com/,http://www.akathiotis.com/quicktime.html,http://www.crystalspace3d.org/
[24] http://www.stereoscopy.com/faq/virtualreality.html
[25]http://www.sv.vt.edu/future/vt-cave/whatis/,
http://www.ceade.cl/realidad_virtual.htm
[26] http://wearables.unisa.edu.au/projects/ARQuake/www/
[27] http://www.tinmith.net/
[28] http://www.hitl.washington.edu/magicbook/
[29] http://www.hitl.washington.edu/research/shared_space/
[30] http://www.gamasutra.com/features/19970801/pathfinding.htm