UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ

FACULTAD DE INGENIERÍA

PROGRAMACIÓN DE UN COMPONENTE ESTEREOSCÓPICO PARA UN SISTEMA CAD

TRABAJO RECEPCIONAL

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN INFORMÁTICA

PRESENTA

MIGUEL ÁNGEL RÍOS LUCIO

SAN LUIS POTOSÍ, S. L. P. **MARZO DEL 2008**

ÍNDICE

INTRODUCCIÓN

Alcance

Otra técnica para representar escenas tridimensionales es la visión estereoscópica. La realización de este trabajo recepcional, consiste, en dar a conocer la utilidad y el alto impacto visual que ofrece la visión estereoscópica como una herramienta alternativa a la apariencia visual tradicional que se obtiene mediante monitores de computadoras para presentaciones de información visualmente más atractivas [1][2]. En este caso, mediante el desarrollo de un nuevo módulo estereoscópico para un sistema de diseño asistido por computadora, 3DS Studio MAX 5 para una computadora personal.

El motivo por el cual se eligió 3D Studio MAX 5, versión al de inicio de este trabajo en el año 2003, aparte de ser el programa más conocido y utilizado para crear aplicaciones 3D en la industria en todo el mundo y de que ya tenía experiencia utilizándolo, es que salió a la venta con casi el 70% del código fuente disponible mediante el SDK, para que programadores en todas las partes del mundo podamos desarrollar nuestros propios componentes. En la página *http :// www.tdt3d.com / articles_ viewer.php ?art_id = 99*; hay una comparación muy detallada de los programas más populares para 3D en el mercado a la fecha actual.

Figura A Visión Estéreo.

La visión estereoscópica es un componente de los sistemas de realidad virtual, donde los usuarios pueden entrar en una escena e interactuar con el entorno. Esta es una manera de obtener un sistema de realidad virtual a un menor costo [2].

Este componente está dirigido para cualquier artista 3D que este dentro de las siguientes industrias por mencionar las más destacadas:

- Publicidad: para previsualizar todo tipo de logos y anuncios publicitarios.
- Escultura: para previsualizar nuevas obras.
- Arquitectura: para previsualizar maquetas virtuales, diseños y decoración interior.
- Diseño de Productos: para previsualizar cualquier tipo de producto.
- Ingeniería: para previsualizar prototipos y modelos.

Antecedentes en Componentes Estereoscópicos

A la fecha de inicio de este trabajo en el año 2003, se encontró un componente llamado "StereoVue-3ds" desarrollado por la firma StereoGraphics en el año 2001, soportando visualización estereoscópica, este componente trabajaba para 3D Studio MAX 4, R3 y VIZ R3. y no se encontró una actualización de ese mismo componente para la versión 5. La página para el componente "StereoVue-3ds" actualmente no existe y solo queda una reseña en la siguiente dirección: *http: // findarticles.com / p /articles/mi_m0EIN /is_2001_June_27/ai_75947410*. La firma StereoGraphics empezó a fusionarse con la firma REAL D en el año 2001 y fue a partir del año 2006 que fue adquirida completamente por la firma REAL D (*http://www.reald-corporate.com*). Actualmente existe un componente que soporta visión estereoscópica llamado: "VR4MAX Generador" desarrollado por Tree C Technology B.V (Países Bajos). Este componente pertenece a la categoría: rendering y trabaja con: 3D Studio MAX: 8, 9; 3D Studio MAX VIZ: 2006, 2007, 2008. El lenguaje del componente está en inglés y su precio es de 850 Euros. Para más información acerca de este componente verificar la página*: http://www.vr4max.com*.

En que consiste

Con el fin de obtener una visión estereoscópica se necesita obtener dos imágenes de una escena tridimensional generadas en computadora desde una dirección de vista correspondiente a nuestros ojos. Las imágenes se ven a través de unos anteojos de cristal líquido **(Figura B)**, con cada lente diseñado para actuar como un obturador que alterna con rapidez (60 a 72 veces por segundo para cada ojo) para presentar las imágenes, los lentes están sincronizados con una señal infrarroja enviada por un emisor infrarrojo conectado a la tarjeta gráfica de la computadora para bloquear una de las imágenes mientras se observa la otra. Cuando vemos de modo simultaneo la imagen izquierda con el ojo izquierdo y la imagen derecha con el ojo derecho, las dos imágenes se combinan en una sola percibiendo una escena con profundidad estereoscópica **(Figura A)** [1][2].

Figura B Emisor Infrarrojo y Lentes para Visión Estéreo.

Conceptos Fundamentales Estereoscópicos

En nuestro sistema de visión fisiológico existen varios procesos para lograr una visión a partir de las imágenes que se forman en cada una de nuestras retinas, a continuación se discuten tales procesos.

Perspectivas Binoculares

Cada uno de nuestros ojos percibe imágenes diferentes debido a que vemos desde dos diferentes perspectivas, una imagen para lo que ve el ojo izquierdo y otra para lo que ve el ojo derecho. Desde estas dos imágenes, los ojos y el cerebro fusionan una sola imagen con profundidad estereoscópica fisiológica, de esta manera, visualizamos una única imagen, en vez de una doble imagen **(Figura C)** [1].

Figura C Perspectivas Binoculares.

Enfoque y Convergencia

Cada vez que visualizamos un objeto, los ojos enfocan (los lentes de los ojos se ajustan cambiando su forma, lo que permite que se vea claramente) y convergen (rotan con respecto a, lo que permite ver un único objeto) al objeto.

Figura D Convergencia Fisiológica.

Este proceso entre enfoque y convergencia se puede entender mejor mediante el siguiente experimento: mantén tu dedo pulgar enfrente de tu cabeza, cuando mires hacia el dedo, tus ojos estarán convergiendo hacia el dedo, de manera que los ejes ópticos de cada ojo están en dirección al dedo. Los músculos se acomodan, moviendo los ojos para efectuar esta tarea, colocando las imágenes del dedo en la porción central de cada retina. Ahora, si continuas convergiendo los ojos hacia el dedo, poniendo atención en lo que hay en el fondo, te darás cuenta que el fondo aparece doble **(Figura E)**. De otra manera, si miramos el fondo, los ojos convergerán a él, y tu dedo aparecerá dos veces **(Figura F)** [1].

Figura E Usando un Pulgar.

Figura F Dos dedos.

El caso en el que el dedo y el fondo aparecen doble se comprende al leer el siguiente proceso: *Disparidad Retinal*.

Disparidad Retinal

Si pudiéramos tomar las imágenes que están en nuestras retinas **(Figura G)**, y de algún modo pudiéramos sobreponerlas, veríamos dos imágenes con diferentes perspectivas **(Figura H)**, de manera que, la distancia en dirección horizontal entre un punto de una imagen retinal sobrepuesta izquierda y un punto de una imagen retinal sobrepuesta derecha es a lo que se llama *Disparidad Retinal*; ver los puntos extremos de la línea verde en la parte superior izquierda de la **Figura I** indicados por dos puntos blancos.

Figura G Imágenes que se forman en cada retina de una escena 3D.

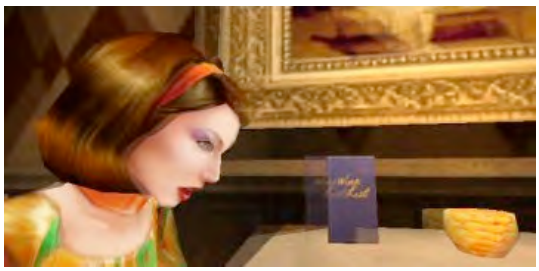

Figura H Imágenes Sobrepuestas.

Figura I Disparidad Retinal.

La disparidad retinal es causada por el hecho de que cada uno de nuestros ojos distan 2 ½ pulgadas o 64 mm (en promedio en personas adultas). La disparidad es procesada por los ojos y el cerebro fusionándolas en una simple imagen y el resultado es una sensación de profundidad llamada *Estereopsis* [1].

Sistema de Visión Estéreo en Computadoras

CAPÍTULO

1.1 Modelo Geométrico

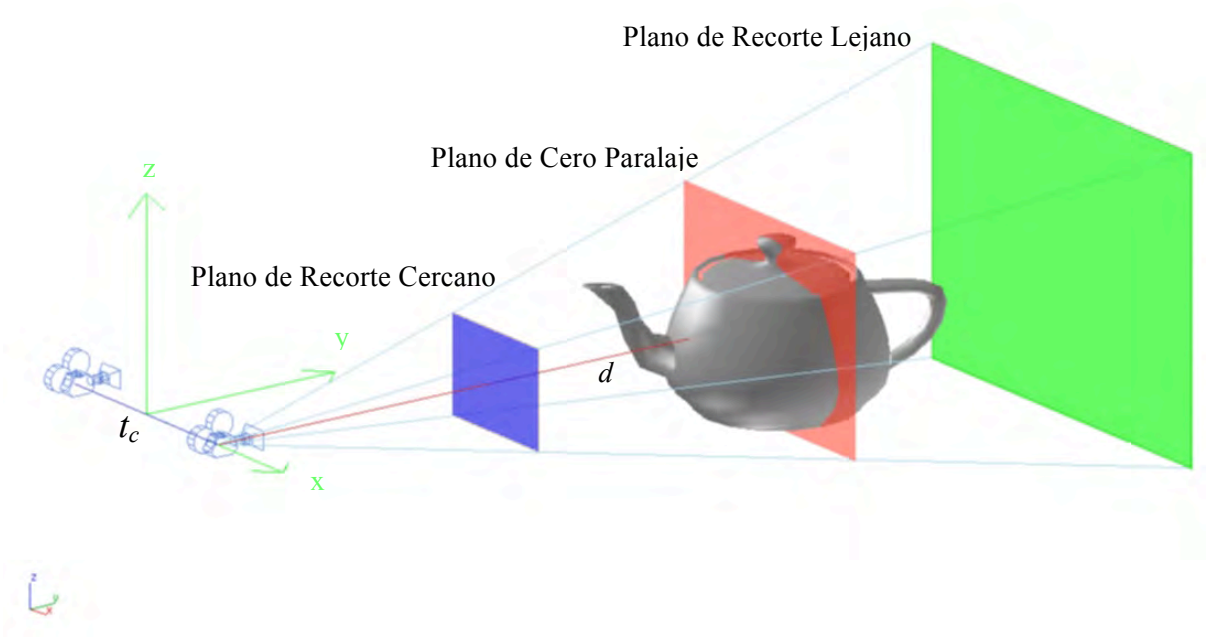

Figura 1.1 Geometría para Visión Estéreo en Computadoras.

En un sistema de visión estéreo en computadora deben existir igualmente que en un sistema de visión fisiológico dos imágenes, una para lo que ve el ojo izquierdo y otra para lo que ve el ojo derecho a partir de dos posiciones de cámara.

La geometría básica para producir imágenes estereoscópicas se ilustra en la **Figura 1.1**, presentando una cámara con el eje de su lente paralelo a partir de dos posiciones (izquierda y derecha) en el espacio de la escena 3D. La distancia de la cámara hasta el plano de *cero paralaje* en la escena está dada por *d*. La cámara producirá dos imágenes con perspectivas diferentes, debido a que se ubica en dos posiciones separadas horizontalmente una *distancia interaxial* (t_c) [1].

Las variables básicas que intervienen para un despliegue estereoscópico se describen a continuación:

1.1.1 Paralaje

La disparidad y la paralaje son entidades similares. La disparidad es medida en la retina y la paralaje es medida en un despliegue en la pantalla de una computadora, de manera que, la distancia entre los puntos correspondientes de una imagen derecha y una imagen izquierda en una computadora es a lo que se denomina *Paralaje* y puede ser medido en pulgadas o milímetros. Cuando se usan lentes para visión estéreo, la paralaje comienza la disparidad retinal y la disparidad produce estereopsis. La paralaje también puede estar dada en términos de medida angular, tomando la distancia desde el observador hasta la pantalla [1].

1.1.2 Casos de Paralaje

Existen cuatro casos de paralaje [1]:

1. Cero Paralaje.- Es cuando los puntos de las imágenes son exactamente correspondientes o un punto se encuentra encima del otro. Cuando los ojos del observador alejados una distancia *t* ven un despliegue en pantalla con imágenes con *Cero Paralaje*, los ojos convergerán al plano de la pantalla. Este tipo de paralaje también es llamado *Configuración de Cero Paralaje* (*ZPS*, siglas en inglés).

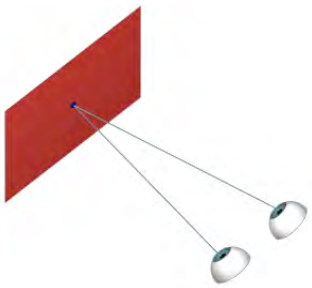

Figura 1.2 Cero Paralaje.

2. Paralaje Positivo.- Cuando la distancia entre los ojos *t* es igual a la paralaje, los ejes de los ojos estarán paralelos. Cuando se tienen valores de paralaje positivo se pueden generar imágenes aparentando estar dentro de la pantalla o en el *Espacio CRT*.

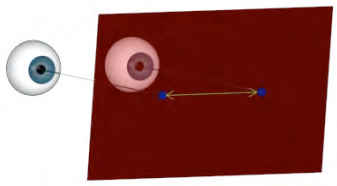

Figura 1.3 Paralaje Positivo.

3. Paralaje Negativo.- Es cuando los ejes de los ojos y los puntos de paralaje están cruzados o negativos. Cuando se tienen valores de paralaje negativo se pueden generar imágenes aparentando estar entre la pantalla y el observador o en el *Espacio del Observador*.

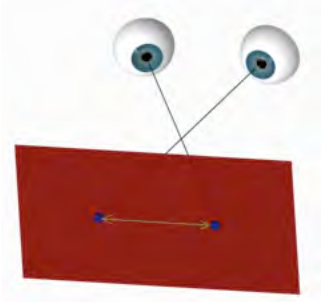

Figura 1.4 Paralaje Negativo.

4. Paralaje Divergente.- Cuando la paralaje es mayor que *t*, los ejes de los ojos se dice que están divergentes, y debido a tal hecho, este tipo de paralaje no se usa en despliegues estereoscópicos ya que esta divergencia no sucede en nuestro sistema visual fisiológico.

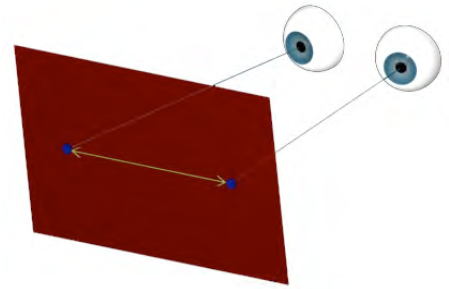

Figura 1.5 Paralaje Divergente.

1.1.3 Campo de Visión (FOV) y Objetivo (Lens)

Ambos parámetros representan una misma propiedad de la cámara por lo que la variación de una repercute en la otra. Se usan para encuadrar la vista de cámara.

El *Campo de Visión* representa el ángulo entre el lado izquierdo y derecho del cono de visión que abarca todo lo que se puede ver a través de una cámara. A mayor ángulo, mayor área de visualización, a costa de mayor distorsión de la imagen.

El *Objetivo* representa la distancia focal de la cámara, la cual se refiere al tamaño de la lente en milímetros. Con valores pequeños, el campo de visión aumenta y la cámara parece estar más alejada de la escena distorsionando la imagen; con valores altos sucede a la inversa y la distorsión disminuye.

Los objetivos menores a 50mm se llaman objetivos de gran angular; en la parte superior de la **Figura 1.6** se aprecia una imagen con un lente gran angular permitiendo ver mayor parte de la escena, y los mayores se llaman teleobjetivos, ver **Figura 1.6** parte inferior [5].

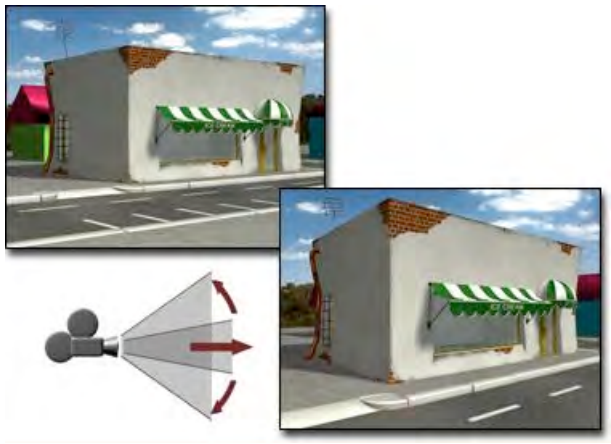

Figura 1.6 Campo de Visión.

Se recomienda realizar proyecciones de gran angular con un *campo de visión* entre los 40 y 50 grados, esto es debido a que si se usan valores diferentes a este rango se producirán imágenes con distorsión [1].

La longitud focal de la cámara debe ser igual para las dos proyecciones.

Referencia 1.1 Pág. 38

1.1.4 Distancia Interaxial

La distancia entre las dos posiciones de cámara usadas para obtener las imágenes para un despliegue estereoscópico es llamada *Distancia Interaxial* (t_c).

En la **Figura 1.7**, cuando la distancia interaxial tiende a aumentar se obtiene un incremento de paralaje y se produce un mayor efecto estereoscópico, de manera que, cuando la distancia tienda a disminuir, el efecto estereoscópico también disminuirá.

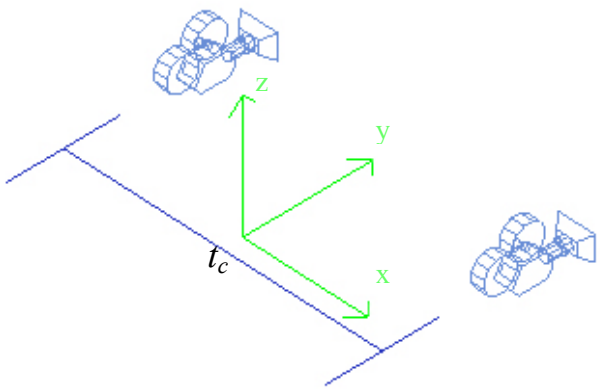

Figura 1.7 Distancia Interaxial (t_c).

El siguiente paso consiste en calcular el valor de la distancia interaxial, que permita obtener un efecto estereoscópico lo suficientemente fuerte para ofrecer una sensación de profundidad efectiva [1].

La tolerancia relativa a una persona para diferentes magnitudes de paralaje en la pantalla de una computadora, son más consistentes cuando la paralaje es expresada como un porcentaje del rango horizontal de la escena que cuando es calculada como una medida angular basada en la disparidad retinal.

Un buen punto de partida para el valor de la distancia interaxial es un 7% del rango horizontal de la escena. El rango horizontal de la escena debe ser medido a lo largo del plano de *cero paralaje* o multiplicando la distancia de la cámara al plano de *cero paralaje* por dos veces la tangente de la mitad del ángulo visual usado.

Referencia 1.2 Pág. 45, **Referencia 1.3 Pág. 45**

Otro factor importante a considerar en el cálculo para el valor de la distancia interaxial, es permitir al usuario ajustar la magnitud deseada de efecto estereoscópico, desde un efecto nulo hasta un máximo efecto. Se recomienda usar un valor por defecto de 1.0, donde el usuario pueda ser capaz de ajustar ese valor descendiendo hasta 0.0 (sin efecto estéreo) ó ascendiendo hasta 2.0 (este valor es usualmente un buen valor superior).

Referencia 1.4 Pág. 45

Una vez obtenido el valor para la distancia interaxial, el siguiente paso será desplazar la cámara la mitad de la distancia hacia la izquierda o en dirección negativa para obtener la imagen izquierda y viceversa para la imagen derecha a partir de su posición original [4].

Referencia 1.5 Pág. 45

1.1.5 Proyecciones Asimétricas

El término *Proyección Asimétrica* esta relacionado con el hecho de que se muestra más detalle de la parte izquierda de la escena que la parte derecha para la proyección izquierda y viceversa. Esto se puede realizar utilizando un *campo de visión* un 20% más del que se pretende usar, antes de realizar ambas proyecciones [3].

Referencia 1.6 Pág. 45

Al realizar las proyecciones, como se menciono en el subtema 1.1.4 *Distancia Interaxial*, todos los posibles puntos se proyectarán con paralaje negativo, lo que causará una visión incomoda. Esto se puede reparar simplemente desplazando los puntos proyectados a la izquierda para la imagen izquierda (aproximadamente un 15% del ancho de la imagen) y a la derecha para la imagen derecha (aproximadamente un 15% del ancho de la imagen) para lograr que una parte de la escena quede en *Cero Paralaje*. De esta manera elementos de la escena originalmente colocados en frente del plano de proyección se proyectarán con paralaje negativo y elementos colocados detrás del plano de proyección se proyectarán con paralaje positivo [4].

Referencia 1.7 Pág. 46

1.1.6 Traslación Horizontal de Imagen

La Traslación Horizontal de la Imagen (*HIT*, siglas en inglés) consiste en un desplazamiento horizontal de las dos imágenes resultantes en direcciones contrarias, con el objetivo de permitir al usuario colocar la escena, ya sea en el *Espacio CRT, Espacio del Observador* o en el plano de la pantalla (*ZPS*) según su preferencia.

Si se trasladan horizontalmente las dos imágenes resultantes relativamente una sobre la otra, estas quedarán, hasta que una porción de las dos imágenes estén sobrepuestas. Donde quiera que las imágenes estén sobrepuestas la paralaje será igual a cero (*ZPS*), y esa porción del objeto aparecerá en el plano de la pantalla (monitor), en este caso, las porciones del objeto detrás del *ZPS* tendrán paralaje positivo, y las que están de frente tendrán paralaje negativo. En la **Figura 1.8**, se muestra este proceso con control de paralaje [1].

Referencia 1.8 Pág. 43

Figura 1.8 HIT.

1.1.7 Control de Distorsión

Existen varias causas en un despliegue estereoscópico que alteran o distorsionan las imágenes causando vistas no deseadas. A continuación, se discuten tales causas:

1.1.7.1 Interferencia

El término *Fantasmas* es usado para describir interferencias. La percepción de fantasmas varía de acuerdo a los valores en la composición de las imágenes: brillo, geometría, color, contraste y paralaje de una imagen. El valor de estos atributos para ambas imágenes deberá ser el mismo o estar dentro de un mínimo nivel de tolerancia. Imágenes con altos valores en los atributos mencionados tendrán más fantasmas que imágenes con valores bajos. Imágenes con altos contrastes, parecidos a líneas negras o fondos blancos presentarán muchos fantasmas. Dados los monitores y del resplandor fosfórico de sus tubos de rayos catódicos, el fósforo verde presenta el nivel mas alto de resplandor y produce demasiados fantasmas, de manera que, habrá que reducir el canal verde de una imagen [1].

Referencia 1.9 Pág. 37

1.1.7.2 Enfoque y Convergencia

.

Cuando se observa una presentación estereoscópica, los ojos enfocan al plano de la pantalla, pero convergen basados en la paralaje. Así, hay una distorsión en la unión de los dos procesos. Este hecho puede ser incomodo de observar para algunas personas.

El problema se hace más grande al ver presentaciones estereoscópicas en monitores pequeños a distancias cortas (de 45cm a 150cm) que son características de despliegues estereoscópicos en computadoras de escritorio. Despliegues en pantallas grandes, vistos a distancias grandes, podrán ser percibidos con menor esfuerzo.

Para evitar este tipo de distorsión causada por estos procesos, se recomienda configurar la escena en el caso de *Cero Paralaje* antes mencionado, en el cual, los ojos enfocan y convergen en el plano de despliegue en la pantalla (*ZPS*).

Referencia 1.10 Pág. 43

En el proceso de convergencia fisiológica mencionado en el capítulo anterior, donde los ojos rotan para fusionar las imágenes de un objeto para diferentes distancias. El uso de este término en este contexto es diferente, ya que la rotación no es empleada. El uso de la rotación para generar las imágenes estereoscópicas provocará una desalineación vertical para los puntos correspondientes de las imágenes derecha e izquierda. Esta desalineación vertical causa a los músculos de los ojos trabajar de forma forzada, lo cual puede ser para mucha gente una experiencia desagradable.

El rectángulo ABCD en la parte izquierda de la **Figura 1.9**, tiene su eje vertical indicado por la línea azul. Cuando se rota alrededor de este eje, como se ve en la parte derecha de la **Figura 1.9**, los puntos A´, B´, C´ y D´ quedan más altos ó más bajos que los puntos correspondientes A, B, C y D. De otra manera, al uso de la rotación, será mejor generar las dos imágenes mediante un desplazamiento horizontal de las imágenes resultantes (*HIT*), entonces una parte de la imagen es dicha ser convergida, cuando los puntos correspondientes de ambas imágenes coinciden [1].

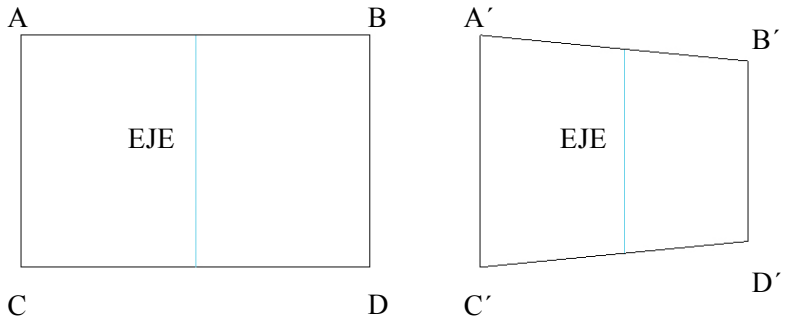

Figura 1.9 Distorsión de una Imagen por el efecto de la Rotación.

1.1.7.3 Control de Paralaje

El objetivo cuando se crean presentaciones estereoscópicas consiste en dar el mejor efecto con el más mínimo valor de paralaje, esto es efectuado en parte al reducir la distancia interaxial y también configurando el campo visual de la cámara con un lente gran angular con una abertura mínima de 40 grados y escalando el objeto para llenar la pantalla. Si la composición lo permite es mejor colocar los objetos principales de la escena cerca del plano de la pantalla (*ZPS*) ó dividir la diferencia en paralaje fijando el plano en medio del objeto, entonces la mitad de los valores de paralaje serán positivos y la otra mitad serán negativos (ZPS).

Referencia 1.11 Pág. 38, **Referencia 1.12 Pág. 43**

Dada la regla: que los valores de paralaje no excedan más de 1.5° que equivale a 1/2 pulgada o 12 mm de paralaje positivo o negativo, serán vistas en una computadora de escritorio a una distancia de 45 cm.

Expresando la paralaje en términos de medida angular, está directamente relacionada con la disparidad retinal. Por Ejemplo: 1/3 de pulgada de paralaje produce el mismo valor de la disparidad retinal a 90 cm de distancia como 2/3 de pulgada de paralaje a 180 cm. Habrá que tomar en cuenta la distancia desde el observador hasta la pantalla.

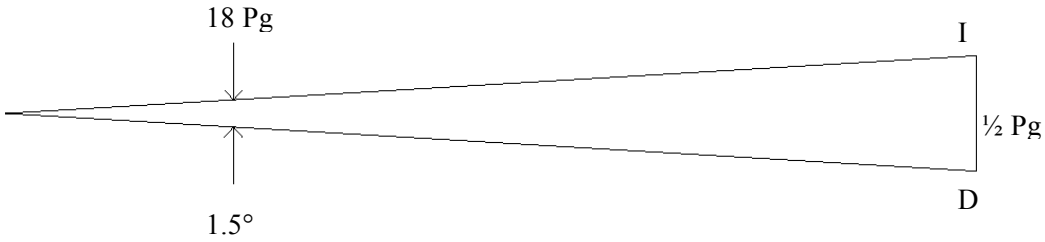

Figura 1.10 Paralaje Máximo.

En la **Figura 1.10**, los puntos de las dos imágenes están separados media pulgada, y vistos a una distancia de 18 pulgadas. Mientras más separado se esté de la pantalla, más distancia será aceptable entre los puntos de las imágenes.

Cuando la paralaje es expresada como un porcentaje del ancho de la pantalla en vez de una medida angular basada en la disparidad retinal resulta más conveniente desde que se elimina la necesidad de suponer el ancho del monitor y la distancia del usuario a la computadora, ver Referencia 1.2 [1][4].

1.1.7.4 Límites de la Ventana Estéreo

El plano de despliegue en la pantalla (monitor) es llamado: *Ventana Estéreo*. Los márgenes verticales y horizontales de la ventana estéreo conforman sus límites (Screen *Surround)*. Como se había comentado una escena estéreo puede estar en varias regiones: en el *Espacio CRT*, entre el *Espacio CRT* y el *Espacio del Observador*, o en el *Espacio del Observador*. Un despliegue estéreo lleva a tomar una decisión básica: donde colocar la escena en la ventana.

Los objetos pueden estar divididos por los límites horizontales del puerto de visión y proporcionar una vista correcta pero no sucede lo mismo si los límites verticales dividen el objeto en el espacio del observador. Si esto sucede, se puede llegar a mal interpretar la posición del objeto, el cual es un inconveniente debido a que esta anomalía no ocurre en nuestro mundo real, y se percibirá una imagen “Fuera de Foco” [1].

Figura 1.12 Escenas con Paralaje Negativo: en Foco (izquierda) y Fuera de Foco (derecha).

En la **Figura 1.12**:

 La imagen de la izquierda ofrece una vista correcta, presenta el globo, en el espacio del observador, centrado dentro los límites de la ventana estéreo.

 La imagen de la derecha ofrece una vista incorrecta debido a que el globo está dividido por el límite vertical izquierdo, en este caso, se presenta un problema de interpretación de la posición del globo, debido a que nuestro sistema de visión interpreta que el globo está detrás de la ventana estéreo, pero el nivel de estereopsis indica que el globo está en frente de la ventana estéreo.

Conceptos Fundamentales para Desarrollar Componentes para 3DS MAX Versión 5 □

CAPÍTULO

2.1 ¿Qué es 3D Studio MAX?

3D Studio Max es un programa que permite generar escenas tridimensionales dándonos la posibilidad de animarlas y representarlas en forma fotorrealista; en otras palabras nos permite construir un mundo virtual, darle vida, fotografiarlo y filmarlo como si estuviéramos en él.

3D Studio Max provee el SDK (Entorno de Desarrollo de Software), que es un conjunto de librerías de programación orientadas a objetos (conjunto de clases) para facilitar la creación de componentes (plugins) mediante los cuales opera.

Mediante el SDK se puede crear una gran variedad de componentes, a continuación, se mencionan algunos de los tipos para los que se puede desarrollar: Objetos Geométricos, Partículas, Objetos de Composición, NURBS, Luces, Cámaras, Modificadores de Objetos, Efectos Especiales, Controladores, Importación y Exportación de Archivos, Efectos Atmosféricos, Materiales, Texturas, Procesamiento de Imágenes, Efectos de Representación, Abrir y Guardar Imágenes, Utilería, Renderers, Filtros Anti-Aliasing, Generadores de Sombras, Dispositivos de Captura de Movimiento, Visión de Imágenes [5][6].

2.2 Secciones Básicas para el Desarrollo de Componentes

Las siguientes secciones son fundamentales al escribir componentes para 3dsmax [6]:

2.2.1 Función DllMain.
2.2.2 Archivo de Definición de Módulo (DEF).
2.2.3 Clase Descriptiva.
2.2.4 Interfase de Usuario.
2.2.5 Bloques de Parámetros.
2.2.6 Mapas de Parámetros.
2.2.7 Nodos.
2.2.8 Matrices de Transformación.
2.2.9 Referencias.

A continuación se describen cada una de estas secciones:

2.2.1 Función DllMain

Todos los componentes para 3dsmax están implementados como DLLs (Dynamic Link Library). Las librerías de enlace dinámico son librerías de código objeto que permiten a múltiples programas compartir código, datos y recursos.

El ligado dinámico permite a un componente ejecutable incluir solo la información requerida en tiempo de ejecución, poniendo el código ejecutable en una función DLL. Este tipo de enlace a diferencia del enlace estático, requiere una copia del código ejecutable de una librería de funciones en el componente ejecutable de cada aplicación que usa este tipo de enlace.

Cuando se compila y enlaza el código del componente, la salida es una DLL. Cualquier componente necesita proveer el código para la función DllMain y cuatro funciones LIB (ver siguiente subtema). La función DllMain es llamada por Windows para inicializar la DLL. Todos los componentes de 3dsmax la usan para inicializar la variable hInstance (contiene la dirección de la DLL) y para inicializar la librería de controles comunes y controles personalizados de 3dsmax. Esta función también puede ser llamada muchas veces durante operaciones de tiempo crítico como una representación. Esta función debe retornar TRUE.

Referencia 2.1 Pág. 31

2.2.2 Archivo de Definición de Módulo (DEF)

Un archivo de definición de módulo es un archivo de texto conteniendo una o más estructuras que describen varias propiedades de una DLL. Una DLL requiere un archivo DEF para crear un archivo LIB (Librería de importación) y un archivo EXP (Librería de exportación). El enlazador usa la librería de importación para construir un módulo ejecutable usado por la DLL y usa la librería de exportación para construir el archivo DLL. La estructura de un archivo DEF consta de las secciones: LIBRARY, donde se especifica el nombre para la librería; EXPORTS, donde se especifica únicamente los nombres de las funciones de exportación (las funciones son implementadas en un archivo de código fuente CPP); SECTIONS donde se especifican atributos de lectura y escritura de datos. La siguiente referencia muestra el contenido del archivo DEF.

Referencia 2.2 Pág. 30

2.2.2.1 Funciones LIB

A continuación, se describen las cuatro funciones LIB usadas para describir las propiedades de un componente contenidas en la sección EXPORTS de un archivo DEF. Las siguientes referencias muestran la implementación de las funciones en código fuente CPP.

- **LibDescription().** Cuando un archivo es cargado conteniendo una entidad (objeto, modificador, etc.) que el sistema no puede acceder (por ejemplo, la DLL no está disponible) esta función regresa una cadena de texto informar al usuario si la DLL no esta disponible.

Referencia 2.3 Pág. 31

- **LibNumberClasses().** Cuando se ejecuta 3dsmax, este verifica para todas las DLLs a cargar. Cuando encuentra una, 3dsmax necesita una manera para determinar el número de las clases contenidas.

Referencia 2.4 Pág. 31

- **LibVersion().** Esta función retorna una constante predefinida denominada **VERSION_3DSMAX** indicando la versión de 3dsmax y del SDK bajo la cual la DLL fue compilada.

Referencia 2.5 Pág. 31

La palabra alta de **VERSION_3DSMAX** contiene el número de versión de 3dsmax multiplicado por 1000, y la palabra baja la conforman el número de la API y la revisión del SDK. 3dsmax solamente cargará DLLs con el número de API actual.

- **LibClassDesc().** Esta función regresa un puntero a la i-gésima clase descriptiva. Una DLL puede tener varias clases descriptivas una por cada categoría de componente. En el siguiente subtema se discute dicha clase.

Referencia 2.6 Pág. 31

2.2.3 Clase Descriptiva

Esta clase provee al sistema información para identificar la categoría de la DLL e información acerca de las clases contenidas.

Para hacer esto, se crea una clase derivada de la clase **ClassDesc**.

Referencia 2.7 Pág. 34

A continuación se describen los métodos que deben ser implementados para esta clase:

IsPublic()

Este método regresa un valor booleano. Si un componente es seleccionado por un usuario, regresa TRUE. Ciertos componentes pueden usarse de forma privada por otros componentes implementados en la misma DLL y no deben aparecer en la lista para ser escogidos por el usuario. Estos componentes retornarán FALSE.

Create(BOOL loading=FALSE)

3dsmax llama a este método cuando necesita crear una instancia de la clase contenida en el componente. El parámetro loading es una bandera indicando si la clase que está siendo creada va a hacer cargada desde un archivo en disco.

ClassName()

Este método regresa el nombre de la clase. Este nombre aparece en el botón para el componente en la interfase de usuario de 3dsmax.

SuperClassID()

Este método regresa una constante definida por el sistema describiendo la categoría de la cual deriva el componente. Por ejemplo, este componente regresa **UTILITY_CLASS_ID**.

ClassID()

Este método retorna el **ID único** del componente. Hay un programa especial en el SDK para generar este ID. Un ID de clase consta de dos cantidades unsigned de 32 bits. El constructor asigna un valor para cada una de estas; por ejemplo, este componente usa el siguiente Class_ID**, Class_ID(0x5048a800, 0x59369337)**.

Category()

Este método regresa una cadena de texto indicando la categoría a la que pertenece un componente en la sección crear en el panel de comandos. Si la cadena de texto es una categoría existente ("Standard Primitives", "Particle Systems", etc.), entonces el componente aparecerá en esa categoría.

3dsmax permite cargar un límite de 12 componentes por categoría. Para prevenir un problema al exceder el límite, se recomienda crear siempre una nueva categoría. Si un componente no necesita aparecer en una categoría específica, el método debe retornar una cadena vacía _T("").

2.2.4 Interfase de Usuario

La interfase de usuario en 3dsmax esta compuesta por una gran cantidad de cuadros de diálogo, persianas, barras de herramientas, barras de estado, visores 3D, el ratón y el teclado permitiendo interactuar a un usuario con 3dsmax.

La interfase de usuario utilizada en el componente para este trabajo está compuesta de los siguientes elementos:

2.2.4.1Persiana

Una persiana permite agregar un área limitada en un espacio de trabajo, con el fin de poder incorporar controles comunes o personalizados de 3dsmax para un fin específico. Un dato importante a considerar el diseño de una persiana es su ancho. El ancho de una persiana dependerá del lugar donde se ubique, en este caso, la persiana está ubicada en el panel de comandos, por lo tanto, el ancho será de 108 unidades.

Figura 3.1 Persiana Estereoscopía.

Figura 3.2 Persiana Parámetros.

Las **Figuras 3.1 y 3.2** muestran las dos persianas utilizadas en el plugin, la persiana Estereoscopia y la persiana Parámetros.

La persiana Estereoscopía esta conformada por una serie de controles comunes, tales como, controles botón, caja de selección y texto; La persiana Parámetros esta conformada por controles comunes como texto y edición (Separación interocular) y por controles personalizados de 3dsmax llamados contadores (Distancia Plano Visión, Ajustar Magnitud Estéreo, Ajustar FOV, Traslación Horizontal Imagen).

El siguiente subtema explica como crear controles comunes y personalizados para una persiana.

2.2.4.2 Como Crear Controles en una Persiana

La herramienta para crear controles comunes y personalizados en persianas para 3dsmax es el editor de diálogos de Visual C++. Los controles se arrastran desde el cuadro de diálogo "controles" de visual C++ y se posicionan en la persiana **(Figura 3.3)**.

Figura 3.3 Diálogo Controles.

Cuando se crea un control personalizado en una persiana, por ejemplo los controles contadores en la persiana Parámetros, se deberá agregar el nombre de la clase que le corresponde en el campo "Class" y un número en sistema hexadecimal indicando el tipo de estilo en el campo "Style", en el cuadro de diálogo "propiedades de control personalizado" **(Figura 3.4)**.

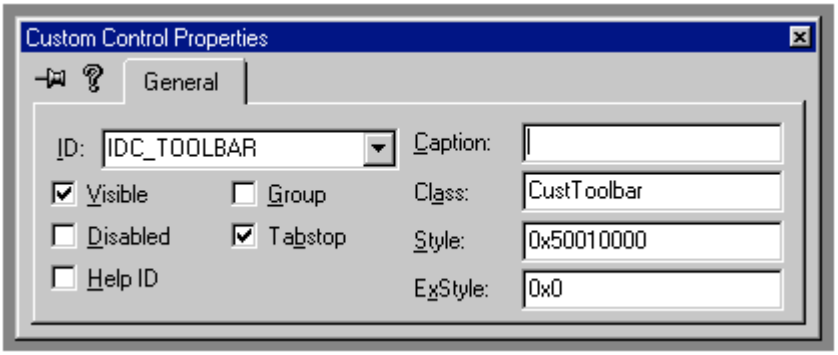

Figura 3.4 Diálogo Propiedades de Controles Personalizados.

Los valores para los campos “Class” y “Style” para los controles contadores (formados por un control edición y uno de ajuste), son los siguientes:

- Control de Edición: Class = **CustEdit,** Style = **0x50010000**
- Control de Ajuste: Class = **SpinnerControl,** Style = **0x50000000**

2.2.4.3 Como Crear y Eliminar una Persiana

Una vez creada una persiana, el siguiente paso será determinar dos consideraciones: el método a utilizar para agregarla al panel de comandos y la técnica a implementar para el procesamiento de mensajes. La persiana Estereoscopía usa técnicas diferentes a la persiana Parámetros en cuanto a las dos consideraciones. El resto de este subtema habla de las técnicas usadas para la persiana Estereoscopía, se continuará hablando de las técnicas usadas para la persiana Parámetros en el subtema 2.2.6 *Mapas de Parámetros*.

El método de la clase Interface **AddRollupPage()** permite agregar una persiana al panel de comandos. Este método es implementado desde el método **BeginEditParams()** de la clase que deriva de la clase UtilityObj.

Referencia 2.8 Pág. 38

A su vez, el método **DeleteRollupPage()** permite eliminar una persiana del panel de comandos. Este método es implementado desde el método **EndEditParams()** heredado de la clase UtilityObj.

Referencia 2.9 Pág. 39

2.2.4.4 Como Procesar Entradas de Usuario

Cualquier persiana usa un "Procedimiento de Diálogo" para procesar las acciones de los usuarios en los controles. Cada vez que el usuario manipula un control, se envía un mensaje con información especificando la acción del usuario, el ID del control, etc. al procedimiento de diálogo especificado, de manera que, habrá que responder a estos mensajes implementando el código para procesar las acciones de usuario. La estructura básica del procedimiento de diálogo es el siguiente:

```
BOOL CALLBACK DialogProc(HWND hDlg, UINT message, WPARAM wParam, LPARAMlParam)
    {
       switch (message) {
        case WM_INITDIALOG:
                break;
        case WM_DESTROY:
                break;
        case WM_COMMAND:
               break;
        // Otros casos...
        default: return FALSE;

      }
      return TRUE;
    }
```

Los cuatro parámetros para un procedimiento de diálogo son: el handle de la caja de diálogo, el mensaje, y dos parámetros los cuales contienen información específica del mensaje. El procedimiento de diálogo debe retornar TRUE si se procesó el mensaje, y FALSE si no lo hizo.

El mensaje **WM_INITDIALOG** es enviado cuando se inicializa la persiana. Este mensaje es usado para inicializar variables, controles, etc.

Cuando un usuario cierra un componente el mensaje **WM_DESTROY** es enviado. Este mensaje es usado para liberar memoria usada por variables, controles, etc.

El mensaje **WM_COMMAND** es enviado cuando un usuario ha ejecutado alguna acción en alguno de los controles. La variable **wParam** contiene información acerca de la naturaleza de la acción y el ID del control, por lo que se deberá analizar la palabra baja y alta de dicha variable para extraer la información.

Referencia 2.10 Pág. 35 (2)

2.2.4.5 Cuadro de diálogo

Un cuadro de diálogo permite agregar un área limitada en un espacio de trabajo, con el fin de poder incorporar controles comunes o personalizados de 3dsmax para un fin específico.

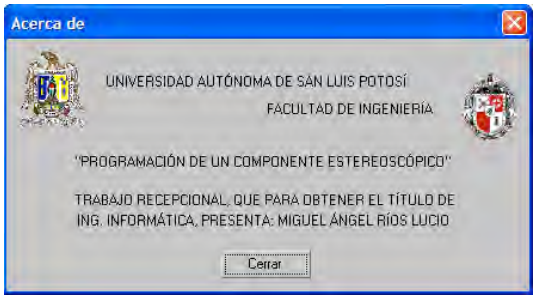

Figura 3.5 Cuadro de Diálogo Acerca de. **Figura 3.6** Cuadro de Diálogo GL_FRONT_BUFFER.

Las **Figuras 3.5 y 3.6** muestran los dos cuadros de diálogo utilizados en el componente.

El cuadro de diálogo "Acerca de" consta de un control texto y un control botón. El cuadro de diálogo GL_FRONT_BUFFER no tiene controles simplemente se usa como una ventana gráfica.

La creación de controles comunes y personalizados para un cuadro de dialogo es de la misma manera como se explico en el subtema 2.2.4.2 *Como Crear Controles en una Persiana*.

Una vez creado un cuadro de diálogo, el siguiente paso será determinar dos consideraciones: el método a utilizar para mostrarlo mediante una acción del usuario en el componente y la técnica a implementar para el procesamiento de mensajes.

La función DialogBox**()** permite mostrar un cuadro de diálogo modal y la función EndDialog() elimina el cuadro de diálogo modal.

Referencia 2.11 Pág. 36, Referencia 2.12 Pág. 35

A su vez, la función **CreateDialogParam()** permite mostrar un cuadro de diálogo no modal y la función **DestroyWindow()** elimina el cuadro de diálogo no modal.

Referencia 2.13 Pág. 39, Referencia 2.14 Pág. 39

Al igual que en una persiana cualquier cuadro de diálogo usa un "Procedimiento de Diálogo" para procesar las acciones de los usuarios en los controles, ver *Como Procesar Entradas de Usuario* para una persiana en el subtema 2.2.4.4.

Referencia 2.15 Pág. 36

2.2.5 Bloques de Parámetros

Un bloque de parámetros provee el mecanismo para almacenar valores constantes o animados de los parámetros (controles) de una interfase de usuario.

2.2.5.1 La Clase ParamBlockDesc

Antes de crear un bloque de parámetros, se debe crear un arreglo especificando el número de parámetros, el tipo de cada parámetro y si estos están o no animados.

Prototipo:

```
class ParamBlockDesc {
 public:
  ParamType type;
  UserType *user;
  BOOL animatable;
};
```

Los argumentos para la clase ParamBlockDesc son:

ParamType type

Los siguientes tipos pueden ser usados:

TYPE_INT – Valores enteros.

TYPE_FLOAT – Valor de punto flotante.

TYPE_POINT3 – Valor point.

TYPE_RGBA – Valores de colores: Rojo, Verde, Azul y Alfa.

TYPE_BOOL – Valores booleanos.

UserType *user

El siguiente valor no es usado debe ser siempre NULL.

BOOL animatable

Esta es una bandera indicando si el parámetro está animado (TRUE) o no (FALSE).

Por ejemplo:

```
// Índices de los parámetros en el bloque de parámetros
#define PB_INDICE_ PARAMETRO_1  0
#define PB_INDICE_ PARAMETRO_2  1
#define PB_INDICE_ PARAMETRO_n  2

ParamBlockDesc pdesc[] = {
    { TYPE_FLOAT, NULL, TRUE }, // Parámetro 1
    { TYPE_FLOAT, NULL, TRUE }, // Parámetro 1
    { TYPE_INT, NULL, FALSE }      // Parámetro n
};
```

2.2.5.2 Creando un Bloque de Parámetros

Un bloque de parámetros es creado mediante el método **CreateParameterBlock()**. Este método requiere dos valores, un puntero al arreglo ParamBlockDesc y un contador del número de parámetros.
Prototipo:

```
IParamBlock *CreateParameterBlock(ParamBlockDesc *pdesc, int count);
```

```
IParamBlock *pblk = CreateParameterBlock(pdesc, 3);
```

La función regresa un puntero al bloque de parámetros creado.

2.2.5.3 Recuperando valores de un Bloque de Parámetros

Cuando se necesita recuperar un valor del bloque de parámetros, se usa el método **GetValue()**. Hay funciones sobrecargadas para cada tipo de valor a recuperar (int, float, Point3, y RGBA). Cada método tiene cuatro parámetros. Abajo se muestra la versión flotante.

```
BOOL GetValue( int i, TimeValue t, float &v, Interval &ivalid );
```

El parámetro **i** es el índice entero del parámetro a recuperar, este es el índice adentro del arreglo ParamBlockDesc. Si el parámetro está animado, este estará variando sobre el tiempo.
El parámetro t especifica a que tiempo se quiere recuperar el valor.
El parámetro **v** es una referencia a un valor flotante. El valor es regresado a través de **v**.
El parámetro **ivalid** es una referencia a un Intervalo.
El valor regresado es TRUE si un valor fue recuperado. En otro caso es FALSE.

Si el valor no ha sido animado, esto es, **SetValue()** no ha sido llamado con un tiempo diferente de cero, entonces un valor constante es almacenado. Cuando **SetValue()** es llamado con el tiempo diferente de cero y el botón “Animate” esta activado una nueva instancia del controlador para el tipo de parámetro es puesto en el bloque de parámetros e inicializado con el valor del parámetro.

El método **GetValue()** se usa para obtener cada parte del intervalo. Un intervalo es una clase que representa un periodo de tiempo, este tiene dos datos miembro privados **start** y **end**, ambos son del tipo de datos **TimeValue**. Un **TimeValue** es un simple instante en el tiempo. Los intervalos son usados en cada parte de 3dsmax describiendo un rango de tiempo y estos son un concepto clave para el entendimiento de animaciones en 3dsmax.

Un componente debe proporcionar información cuando están cambiando y cuando no lo hace. Si el parámetro nunca cambia el parámetro será indicado FOREVER.

2.2.5.4 Colocando Valores en un Bloque de Parámetros

Cuando se necesita guardar un valor en el bloque de parámetros, se usa el método **SetValue()**. Hay funciones sobrecargadas para cada tipo de valor a guardar (int, float, Point3, y RGBA.) Cada método tiene tres parámetros. A continuación se muestra la versión para valores de tipo flotante:

BOOL SetValue(int i, TimeValue t, float v);

El parametro **i** es el índice entero del parámetro a poner. Este es el índice adentro del arreglo ParamBlockDesc del parámetro.
El parametro **t** especifica en que tiempo poner el valor.
El valor a guardar es pasado en **v**.
El valor regresado es TRUE si el valor fue puesto. En otro caso es FALSE.

Referencia 2.16 Pág. 41, **Referencia 2.17 Pág. 41**, **Referencia 2.18 Pág. 41**

2.2.6 Mapas de Parámetros

En el subtema 2.2.4 *Interfase de Usuario* se mencionó como una persiana o cuadro de diálogo usan un "Procedimiento de Diálogo" para procesar las acciones de los usuarios en los controles.

Los mapas de parámetros son otra alternativa para procesar las acciones de los usuarios en los controles en la interfase de usuario minimizando el esfuerzo de programación en comparación con un procedimiento de diálogo.

Cuando un usuario opera los controles en un componente, el mapa de parámetros procesa los mensajes enviados por los controles y guarda automáticamente los valores en un bloque de parámetros. Se necesita implementar los métodos **SetValue()** y **GetValue()** para que el mapa de parámetros pueda obtener y poner el i-gésimo elemento en el arreglo virtual asignando un índice entero a cada parámetro.

El uso de los mapas de parámetros tiene las siguientes ventajas:

- No se necesita escribir código para controlar el procesamiento de mensajes para los eventos manejados por el usuario. En caso de no usar mapas de parámetros se deberá implementar un procedimiento de diálogo para procesar los mensajes.

- Las operaciones Hacer y Deshacer son manejadas automáticamente.

- Las operaciones para Cargar y Guardar parámetros hacia y desde el disco son manejadas automáticamente.

A continuación se mencionan las clases que se relacionan con los mapas de parámetros:

- Clase ParamUIDesc – Esta clase define las propiedades de los controles en una interfase de usuario según su tipo (casillas de verificación, ajustadores, etc.), su ID de recurso, su índice en el arreglo virtual y el rango de valores permitidos.

- Clase ParamBlockDescID – Esta clase describe cada parámetro según su tipo de dato (entero, flotante, punto, color), si son animados o constantes, donde se puede usar un ID para el control de versiones futuras.

- Clase IParamArray – Esta clase representa un arreglo virtual donde se almacenan los valores de los controles o parámetros de una interfase de usuario. Los mapas de parámetros y los bloques de parámetros derivan de esta clase.

- Clase IParamMap - Esta clase provee métodos para trabajar con mapas de parámetros.

En la **Figura 3.7** se muestra la persiana con los controles que conforman el mapa de parámetros para el componente.

Figura 3.7 Interfase de usuario.

2.2.6.1 Declarar las Variables de la Interfase de Usuario

Cada mapa de parámetros administra una única persiana en el panel de comandos. Se deberá declarar una o varias variables dependiendo de la cantidad de persianas a utilizar. Cada variable será un puntero al objeto **IparamMap.**

Referencia 2.19 Pág. 32

También se deben declarar variables relacionadas con cada control (parámetro) en la interfase de usuario.

Referencia 2.20 Pág. 32

2.2.6.2 Arreglo de Parámetros

Se debe definir un arreglo de tipo **ParamUIDesc** que como ya se menciono sirve para establecer las propiedades de los controles que posteriormente se utilizará para crear el mapa de parámetros
.

Referencia 2.21 Pág. 34

2.2.6.3 Agregar la Persiana al Panel de Comandos

El método **CreateCPParamMap()** permite agregar una persiana al panel de comandos y también crea un mapa de parámetros.

Este método tiene varios argumentos. El primero es el arreglo **ParamUIDesc**. El segundo es el número de elementos en el arreglo. El tercero es un puntero al arreglo virtual de parámetros. El puntero **this** significa que el objeto UtilTest es el puntero a **IparamArray** (el objeto UtilTest deriva de la clase **IparamArray**). De está forma, el mapa de parámetros puede acceder a las variables del objeto UtilTest relacionadas con los controles o parámetros de la interfase de usuario para el arreglo virtual. El cuarto parámetro es el puntero a la clase interfase pasado en el método **BeginEditParams().** El quinto parámetro es la instancia DLL del componente. El siguiente parámetro es el diálogo para la persiana. El siguiente parámetro es el título desplegado en la barra de título de la persiana. El parámetro final es un conjunto de banderas para controlar la configuración de la persiana.

Referencia 2.22 Pág. 38

2.2.6.4 Permitir al Mapa de Parámetros Acceder a las Variables del Plugin

Los métodos **GetValue() y SetValue()** de la clase **IParamArray** permiten a un mapa de parámetros trabajar con las variables relacionadas**.** El arreglo virtual trabaja usando un índice para especificar el parámetro que se debe recuperar o poner.

La siguiente referencia muestra la implementación de estos métodos. Cada método usa un selector para verificar el índice en el arreglo virtual y recuperar o poner el valor de la variable apropiada.

Referencia 2.23 Pág. 40 (2)

2.2.6.5 Bloques de Parámetros

Un mapa de parámetros maneja internamente un bloque de parámetros, La clase IParamaMap provee algunos métodos para trabajar con el bloque de parámetros. Ver la sección 2.2.5 *Bloques de Parámetros* para más detalles**.**

Referencia 2.24 Pág. 35, Referencia 2.25 Pág. 44

2.2.6.6 Eliminando la Persiana del Panel de Comandos

El método **DestroyCPParamMap()** elimina una persiana del panel de comandos y también libera los controles asociados al mapa de parámetros. Este método se implementa desde el método **EndEditParams()** de la clase que deriva de la clase UtilityObj.

Referencia 2.26 Pág. 39

2.2.7 Nodos

Un nodo es el elemento que contiene la información necesitada para permitir al objeto asociado existir en la escena, tal como, el controlador de transformación, el material usado, información de grupo, estado de visibilidad o invisibilidad, información de jerarquía, etc. Cada objeto en una escena 3D está asociado a un nodo. Hay una correspondencia uno a uno entre cada elemento de la escena y un nodo.

La clase **INode** dispone de los métodos para trabajar con los nodos**.** Tales métodos proveen funciones tales como, acceso al panel de geometría, poner y obtener el nombre de un nodo, definir las jerarquías entre nodos padres/hijos, acceso a los atributos de un nodo, acceso a los controladores, etc.

Referencia 2.27 Pág. 32, Referencia 2.28 Pág. 38, Referencia 2.29 Pág. 38, Referencia 2.30 Pág. 40, Referencia 2.31 Pág. 41 (3), Referencia 2.32 Pág. 42 (6), Referencia 2.33 Pág. 42 (5), Referencia 2.34 Pág. 45 (3), Referencia 2.35 Pág. 48

2.2.8 Matrices de Transformación

Una matriz es un arreglo bidimensional de números de 4 renglones x 3 columnas, habiendo un total de 12 elementos.

Las transformaciones proporcionadas por estas matrices son: traslación, escalación, y rotación. La clase **Matrix3** provee métodos para crear las matrices y realizar operaciones aritméticas. Estas matrices pueden ser usadas, por ejemplo, para posicionar nodos en la escena.

La representación de los elementos de una matriz se muestra en la **Figura 3.8**:

m[0][0]	m[0][1]	m[0][2]
m[1][0]	m[1][1]	m[1][2]
m[2][0]	m[2][1]	m[2][2]
m[3][0]	m[3][1]	m[3][2]

Figura 3.8 Matrix3.

Referencia 2.36 Pág. 42 (14), Referencia 2.37 Pág. 45 (9)

2.2.9 Referencias

Cuando un usuario opera una entidad sea cual fuere su naturaleza, por ejemplo una cámara, al cambiar un valor para alguno de sus parámetros, se deberá informar al sistema acerca de cualquier cambio que se realice, ya que un cambio en una entidad puede repercutir en algunas acciones a realizar por otras entidades que dependan o estén relacionadas, de otra manera, una entidad también puede ser informada cuando se realice algún cambio en las entidades dependientes. 3dsmax permite manejar estas relaciones de dependencia entre entidades mediante un esquema denominado *referencias*.

Hay dos clases relacionadas en un esquema de referencias, estas clases son: **ReferenceMaker** y **ReferenceTarget.**

Cualquier componente que crea una referencia deriva de la clase **ReferenceMaker**. En el componente para este trabajo se crea una referencia a la cámara que el usuario define para obtener las perspectivas estéreo. 3dsmax usa un sistema de mensajes para notificar a las entidades dependientes acerca de algún cambio.

La clase **ReferenceTarget** es usada en un componente para permitir a las entidades que tienen referencias a nuestra entidad (en el componente para este trabajo ninguna entidad dependerá, por lo tanto los métodos para esta clase no se implementan) enviarles mensajes de notificacion de cambios.

La clase ReferenceTarget deriva de la clase ReferenceMaker. En un componente se deberá crear una clase que derive de la clase ReferenceTarget para permitir al componente hacer uso del esquema de referencias.

Referencia 2.38 Pág. 33

Hay ocho métodos clave que deben ser implementados de la clase ReferenceMaker para trabajar en un esquema de referencias. Estos métodos resultan de la herencia al derivar nuestra clase de la clase RerenceTarget como se explico anteriormente. Estos métodos son **MakeRefByID()**, **NotifyRefChanged(), SetNotifyFunc(), NumRefs(), GetReference(), SetReference(), DeleteAllRefsFromMe() y DeleteReference()**.

El método **MakeRefByID()** permite a un componente crear una referencia a cualquier entidad en la escena y de esta manera permitir al componente recibir mensajes de notificación de cambios de las entidades a los que ha creado referencias.

Referencia 2.39 Págs.: 38, 47

Un componente debe implementar un método para recibir mensajes de notificación de cambios enviados por entidades en la escena. Estos se hace implementando el método **NotifyRefChanged()**.

Referencia 2.40 Pág. 47

Un componente debe implementar el método **NumRefs()** para retornar el numero de referencias que crea.

Referencia 2.41 Pág. 47

Un componente debe implementar el método **GetReference()** para retornar la i-gésima referencia. Un componente mantiene una lista de referencias usando un número entero para identificar a cada una. Cuando el sistema llama a este método el componente retorna la i-gésima referencia.

Referencia 2.42 Pág. 47

Un componente debe implementar el método **SetReference()** para guardar la i-gésima referencia. Un componente mantiene una lista de referencias usando un número entero para identificar a cada una. Cuando el sistema llama a este método el componente guarda la i-gésima referencia.

Referencia 2.43 Pág. 47

El método **DeleteAllRefsFromMe**() elimina todas las referencias para un componente**.**

Referencia 2.44 Pág. 47

El método **SetNotifyFunc()** es usado para indicar al sistema la función que servirá para procesar los mensajes de referencias enviados a un componente por parte de las entidades en la escena a los que ha creado una referencia o de los que depende.

Referencia 2.45 Págs.: 33, 38, 48

Un componente debe implementar un método para eliminar las referencias creadas. Estos se hace implementando el método **DeleteReference()**.

Referencia 2.46 Pág. 47

2.3 Componentes de Utilidades

Los componentes de utilería pertenecen a una clase de componentes que se diseñan con el fin de crear herramientas para tareas muy específicas. Este tipo de componentes se encuentran en el panel utilidades del panel de comandos. Los componentes de utilería no participan directamente del panel de geometría. Por esta razón este tipo de componentes no son aptos para trabajar con la geometría de los objetos. Cualquier componente en esta categoría derivará de la clase **UtilityObj** [6].

Referencia 2.47 Pág. 32

Los métodos de la clase **UtilityObj** se describen a continuación:

virtual void BeginEditParams(Interface *ip,IUtil *iu)=0;

Este método es implementado por el componente. Este método es llamado cuando el componente es elegido desde el panel utilidades en el panel de comandos. Este método es usado para agregar las persianas que lo conforman.

El parámetro ip es un puntero a la clase Interfase la cual proporciona muchos de los métodos para acceder a una gran cantidad de funciones de 3dsmax. El parámetro iu es un puntero a la clase IUtil la cual proporciona un único método usado para cerrar el componente del panel utilidades.

Referencia 2.48 Pág. 32, Referencia 2.49 Pág. 38

virtual void EndEditParams(Interface *ip,IUtil *iu)=0;

Este método es implementado por el componente. Este método es llamado cuando el usuario elige cerrar el componente o cuando decide cambiar a cualquier otro panel en el panel de comandos. En este método se implementa código para liberar la memoria usada, por ejemplo, por punteros, cuadros de diálogo no modales, persianas, mapas de parámetros, etc. Los parámetros de este método se describieron en el método anterior.

Referencia 2.50 Pág. 32, Referencia 2.51 Pág. 39

virtual void DeleteThis()=0;

Este método es implementado por el componente. Este método es usado para eliminar el objeto de utilería que fue instanciado en el método **ClassDesc::Create()**. Ver método **ClassDesc::Create()** en la sección 2.2.3 *Clase Descriptiva*, para más detalles**.**

Cuando el sistema necesita eliminar la instancia de la clase en un componente, este llama al método **Animatable::DeleteThis()**. Dado que se usa el operador **new** para asignar memoria, se deberá usar el operador **delete** para liberar la memoria como se describe: **void DeleteThis() { delete this; }**

Para aquellos componentes que no derivan de la clase Animatable, esta acción puede ser llevada acabo mediante un método DeleteThis() no heredado. Por ejemplo, la clase **UtilityObj** usada para crear componentes de utilería no deriva de la clase **Animatable** y tiene su propio método **DeleteThis()**. El método **DeleteThis()** proporciona control sobre la liberación de memoria. Por ejemplo, un objeto de un componente de utilería puede ser declarado estáticamente y no ser declarado en la pila. Un componente de utilería no implementa el método **DeleteThis().** Si no hay memoria en la pila para liberar el método queda así:
void DeleteThis() {}.

Referencia 2.52 Pág. 32

Programación de un Componente Estereoscópico para 3DS MAX Versión 5 □

CAPÍTULO

3.1 Introducción

Esta sección contiene el código fuente completo del componente hecho en Microsoft Visual C++ Versión 6.0.

3D Studio Max 5 está programado con Visual C++ 6.0 y fue desde la versión 3D Studio Max 6 cuando se utilizo la nueva plataforma de programación .NET Framework. Por lo tanto, si este componente desea ser utilizado en versiones posteriores tendrá que ser recompilado y configurado según las nuevas características de la nueva plataforma .NET Framework.

3.2 Archivo de Definición de Módulo

Nombre del Archivo: util.def
Código:

Referencia 2.2

```
LIBRARY estereo
EXPORTS
        LibDescription          @1
        LibNumberClasses        @2
        LibClassDesc            @3
        LibVersion              @4

SECTIONS
        .data READ WRITE
```

3.3 Archivos de Cabecera

Nombre del Archivo: **util.h**
Código:

```
#ifndef __UTIL__H

        #define __UTIL__H

        #include "Max.h"
        #include <gl\gl.h>
        #include <gl\glu.h>
        #include "resource.h"

        TCHAR *GetString(int id);
        extern ClassDesc* GetUtilTestDesc();
        extern HINSTANCE hInstance;

#endif
```

3.4 Archivos de Código Fuente

Nombre del Archivo: **util.cpp**
Codigo:

```
#include "util.h"

HINSTANCE hInstance;
int controlsInit = FALSE;

Referencia 2.1
BOOL WINAPI DllMain(HINSTANCE hinstDLL,ULONG fdwReason,LPVOID lpvReserved) {
        hInstance = hinstDLL;
        if (!controlsInit) {
            controlsInit = TRUE;
            InitCustomControls(hInstance);
            InitCommonControls();
        }
        return (TRUE);
}


Referencia 2.3
__declspec( dllexport ) const TCHAR * LibDescription()
{
        return GetString(IDS_PLUGIN_MANAGER);
}


Referencia 2.4
__declspec( dllexport ) int LibNumberClasses()
{
        return 1;
}


Referencia 2.5
__declspec( dllexport ) ULONG LibVersion()
{
        return VERSION_3DSMAX;
}


Referencia 2.6
__declspec( dllexport ) ClassDesc* LibClassDesc(int i)
{
        switch(i) {
        case 0:  return GetUtilTestDesc();
        default: return 0;
        }
}
```

```
TCHAR *GetString(int id)
{
      static TCHAR buf[256];

      if (hInstance)
        return LoadString(hInstance, id, buf, sizeof(buf)) ? buf : NULL;

     return NULL;
}
```

Nombre del Archivo: **utiltest.cpp**
Codigo:

```
#include "util.h"
#include "utilapi.h"
#include "istdplug.h"
#include "bmmlib.h"
#include "iparamm.h"

#define UTILTEST_CLASS_ID                Class_ID(0x5048a800, 0x59369337)
#define VIEW_WIDTH                       320
#define VIEW_HEIGHT                      240
#define STEREO_MAGNITUDE_CONSTANT        0.07f
#define ASIMMETRIC_WIDTH                 VIEW_WIDTH*0.05

HDC              hdc;
HGLRC            hglrc;

class UtilTest : public UtilityObj, public IParamArray{                    Referencia 2.47
public:
        // Métodos Heredados de UtilityObj
        void BeginEditParams (Interface *ip, IUtil *iu);                   Referencia 2.48
        void EndEditParams   (Interface *ip, IUtil *iu);                   Referencia 2.50
        void DeleteThis      (){}                                          Referencia 2.52

        // Métodos Heredados de IParamArray
        BOOL SetValue(int i, TimeValue t, float v);
        BOOL GetValue(int i, TimeValue t, float &v, Interval &ivalid);

        // Variables y Métodos Generales de UtilTest
        IUtil           *iu;
        Interface       *ip;
        HWND            m_hwndPanel,m_hwndStereoPanel;
        IParamMap       *m_pmapPARAMETERS;                                 Referencia 2.19
        ViewExp         *m_veActiveViewport;
        INode           *m_nodeZPS,*m_nodeCamUser;                         Referencia 2.27
        CameraObject    *m_coUser;
                                                                           Referencia 2.20
        float           m_fFOV,m_fHIT,m_fStereoMagnitudeAdj,m_fVisionPlaneDist;
        BOOL            m_bRenderFirstFrame;
        Bitmap          *m_bmIzq,*m_bmDer;
```

```
        void iniVariables           ();
        BOOL GetUserCamera    ();
        void ActivaControles        ();
        void CreaReferencia         ();
        void CrearPlanoZPS          ();
        void PlanoZPS_TM            ();
        void AjustarFOV             ();
        void AjustarHIT             ();
        void RenderFrame            ();
        void StereoProjection  (Bitmap *bm, double EyePosition);
        void EscribeMapaBits (Bitmap *bm, double EyePosition);
};
static UtilTest theUtilTest;
```

Referencia 2.45

```
RefResult ProcessRefMessage(Interval changeInt, RefTargetHandle hTarget,
        PartID& partID, RefMessage message, DWORD pData);
```

Referencia 2.38

```
class NotifyMgr : public ReferenceTarget {
private:

        RefResult (*pNotifyFunc)(Interval changeInt,RefTargetHandle hTarget,
                        PartID& partID,RefMessage message, DWORD pData);
        RefTargetHandle ref0;
        DWORD pPrivateData;
public:
        //Métodos Virtuales Heredados de Animatable
        void GetClassName(TSTR &s) { s = _T("(Reference)"); }

        //Métodos Virtuales Heredados de ReferenceMaker
        RefResult NotifyRefChanged (Interval changeInt,RefTargetHandle hTarget,
                                        PartID& partID,RefMessage message);
        int NumRefs();
        RefTargetHandle GetReference (int i);
        void SetReference (int i, RefTargetHandle rtarg);

        //Métodos de NotifyMgr
        NotifyMgr::NotifyMgr();
        NotifyMgr::~NotifyMgr();
        BOOL CreateReference(RefTargetHandle hTarget);
        BOOL RemoveReference();

        void SetNotifyFunc(RefResult (*func)(Interval changeInt,
                    RefTargetHandle hTarget, PartID& partID,
                    RefMessage message,DWORD pData), DWORD pData);
        void ResetNotifyFunc();
};

NotifyMgr notifyObject;
```

Referencia 2.7

```
class UtilTestClassDesc:public ClassDesc {
public:
        int                     IsPublic() {return 1;}
        void *                  Create(BOOL loading = FALSE) {return &theUtilTest;}
        const TCHAR *           ClassName() {return GetString(IDS_NOMBRE_PLUGIN);}
        SClass_ID               SuperClassID() {return UTILITY_CLASS_ID;}
        Class_ID                ClassID() {return UTILTEST_CLASS_ID;}
        const TCHAR*            Category() {return _T("");}
};

static UtilTestClassDesc utilTestDesc;
ClassDesc* GetUtilTestDesc() {return &utilTestDesc;}


/*============================================================================*\
                Mapas de Parámetros (Interfase de usuario) y Procedimientos de Diálogo
/*============================================================================*/

#define PM_DISTANCIA_PLANO      0
#define PM_MAGNITUD_ESTEREO  1
#define PM_AJUSTAR_FOV          2
#define PM_AJUSTAR_HIT          3


#define PARAMETERS_DESC_LENGTH 4
```

Referencia 2.21

```
static ParamUIDesc descParams[] = {
        ParamUIDesc(PM_DISTANCIA_PLANO,
                        EDITTYPE_UNIVERSE,
                        IDC_EDIT_PLANO, IDC_SPIN_PLANO,
                        13.0f, 4000.0f, 1.0f),

        ParamUIDesc(PM_MAGNITUD_ESTEREO,
                        EDITTYPE_UNIVERSE,
                        IDC_EDIT_CAMARA,IDC_SPIN_CAMARA,
                        0.0f, 2.0f, 0.1f),

        ParamUIDesc(PM_AJUSTAR_FOV,
                        EDITTYPE_UNIVERSE,
                        IDC_EDIT_FOV,IDC_SPIN_FOV,
                        40.0f, 100.0f, 10.0f),

        ParamUIDesc(PM_AJUSTAR_HIT,
                        EDITTYPE_UNIVERSE,
                        IDC_EDIT_HIT,IDC_SPIN_HIT,
                        0.0f, 350.0f, 1.0f)
};
```

Referencia 2.10

```
static BOOL CALLBACK VerAcercaDe(
        HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam) {
        switch (msg) {
                case WM_INITDIALOG:
                        CenterWindow(hWnd, GetParent(hWnd));
                        break;
                case WM_COMMAND:
                        switch (LOWORD(wParam)) {
                                case IDOK:
                                        EndDialog(hWnd,1);                    Referencia 2.12
                                        break;
                        }
                        break;
                default:
                        return FALSE;
                }
        return TRUE;
}
```

Referencia 2.10

```
static INT_PTR CALLBACK MainUserDlgProc(
        HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam) {

        switch (msg) {
        case WM_INITDIALOG:
                //Deshabilitar controles IDD_MENU
                CheckDlgButton(hWnd,IDC_CHECK_FORMATO,BST_UNCHECKED);
                EnableWindow(GetDlgItem(hWnd, IDC_RENDER),     FALSE);
                EnableWindow(GetDlgItem(hWnd, IDC_VAL_DEFECTO),FALSE);
                EnableWindow(GetDlgItem(hWnd, IDC_SAVE_IMAGE), FALSE);
                return TRUE;

        case WM_DESTROY:
                 return TRUE;

        case WM_COMMAND:
                switch (LOWORD(wParam)) {
                case IDOK:
                        theUtilTest.iu->CloseUtility();
                        break;
                case IDC_RENDER:
                        theUtilTest.RenderFrame();
                        break;
                case IDC_VAL_DEFECTO:
                        IParamBlock *pb;                                    Referencia 2.24
                        pb= (IParamBlock*)theUtilTest.m_pmapPARAMETERS->GetParamBlock();

                        pb->SetValue(PM_DISTANCIA_PLANO,     theUtilTest.ip->GetTime(), 13.0f );
                        pb->SetValue(PM_MAGNITUD_ESTEREO, theUtilTest.ip->GetTime(),  0.0f  );
                        pb->SetValue(PM_AJUSTAR_FOV,              theUtilTest.ip->GetTime(), 40.0f );
                        pb->SetValue(PM_AJUSTAR_HIT,  theUtilTest.ip->GetTime(),  0.0f  );
```

```
                    theUtilTest.m_pmapPARAMETERS->SetParamBlock((IParamArray*)pb);
                    break;

            case IDC_ACERCADE:
                                                                    Referencia 2.11
                    DialogBox(hInstance, MAKEINTRESOURCE(IDD_ACERCADE),
                            GetCOREInterface()->GetMAXHWnd(), VerAcercaDe);
                    return TRUE;

            case IDC_AYUDA:
                    MessageBox(hWnd, GetString(IDS_AYUDA),
                            "Ayuda", MB_ICONEXCLAMATION|MB_OK);
                    break;
            }

            default: return FALSE;
            }
        return TRUE;
}
```

Referencia 2.15

```
static INT_PTR CALLBACK MainDlgProc(
        HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam) {

GLboolean bStereo = GL_FALSE;
PIXELFORMATDESCRIPTOR pfd;
int iPixelFormat,i;

        switch (msg) {
        case WM_INITDIALOG:

                hdc=GetDC(hWnd);
                memset(&pfd,0,sizeof(PIXELFORMATDESCRIPTOR));

                pfd.nSize       = sizeof(PIXELFORMATDESCRIPTOR);
                pfd.nVersion    = 1;
                pfd.dwFlags     = PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL |
                                  PFD_DOUBLEBUFFER  | PFD_STEREO;
                pfd.iPixelType  = PFD_TYPE_RGBA;
                pfd.cColorBits  = 32;
                pfd.cDepthBits  = 24;
                pfd.iLayerType  = PFD_MAIN_PLANE;

                iPixelFormat = ChoosePixelFormat(hdc, &pfd);
                i=SetPixelFormat(hdc, iPixelFormat, &pfd);
                hglrc = wglCreateContext(hdc);
                wglMakeCurrent(hdc, hglrc);

                 //glGetBooleanv(GL_STEREO,...); Es supuestamente la manera para
                //consultar disponibilidad estéreo, pero desafortunadamente, el
                //resultado de esta consulta no es confiable para algunas tarjetas
                //gráficas.
                glGetBooleanv(GL_STEREO, &bStereo);
```

```
            //Recomiendo esta aproximación para verificar disponibilidad estéreo.
            iPixelFormat = GetPixelFormat (hdc);
            DescribePixelFormat(hdc, iPixelFormat, sizeof(PIXELFORMATDESCRIPTOR),&pfd);
            if (pfd.dwFlags & PFD_STEREO) //Si modo estéreo es aceptado
              CheckDlgButton(theUtilTest.m_hwndPanel,
                             IDC_CHECK_FORMATO,
                            BST_CHECKED);

            //Fondo no completamente negro, sin fantasmas.                    Referencia 1.9
            glClearColor(0.2f,0.2f,0.2f,0.0f);
            //glClearColor(0,0,0,0);
            glDrawBuffer(GL_BACK);
            glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
            glEnable(GL_DEPTH_TEST);
            glDepthFunc(GL_LESS);
            glEnable(GL_CULL_FACE);
            glShadeModel(GL_FLAT);
            return TRUE;

        case WM_SIZE:
            glViewport(0,0, (GLsizei)LOWORD(lParam),(GLsizei)HIWORD(lParam));
            break;

        case WM_DESTROY:
            if(hdc) {
                if(theUtilTest.m_bRenderFirstFrame) {
                    glDeleteLists(1,1);
                    glDeleteLists(2,1);
                }
                wglMakeCurrent(hdc, NULL);
                wglDeleteContext(hglrc);
                ReleaseDC(hWnd, hdc);
            }
            return TRUE;

        case WM_ERASEBKGND:
            return 1;

        case WM_PAINT:
            if(theUtilTest.m_bRenderFirstFrame)theUtilTest.AjustarHIT();
            else {
                SwapBuffers(hdc);
                glFlush();
            }
            ValidateRect(hWnd, NULL);
            break;

        default: return FALSE;
        }
        return TRUE;
}
```

```
/*==========================================================================*\
                              Métodos de UtilTest
\*==========================================================================*/

void UtilTest::iniVariables() {
        iu                      = NULL;
        ip                      = NULL;
        m_hwndPanel             = NULL;
        m_hwndStereoPanel       = NULL;
        m_pmapPARAMETERS        = NULL;
        m_veActiveViewport      = NULL;
        m_nodeZPS = m_nodeCamUser = NULL;                          Referencia 2.28
        m_coUser                = NULL;
        m_fFOV                  = 40.0f;          Referencia 1.1 , Referencia 1.11
        m_fHIT                  = 0.0f;
        m_fStereoMagnitudeAdj   = 0.0f;
        m_fVisionPlaneDist      = 1.0f;
        hdc                     = NULL;
        hglrc                   = NULL;
        m_bRenderFirstFrame     = FALSE;
        m_bmIzq = m_bmDer       = NULL;
}

Referencia 2.49
void UtilTest::BeginEditParams(Interface *ip,IUtil *iu) {
        iniVariables();
        this->iu = iu;
        this->ip = ip;

        if(!m_hwndPanel)
        {
                                                                   Referencia 2.8
                m_hwndPanel = ip->AddRollupPage(hInstance, MAKEINTRESOURCE(IDD_MENU),
                                        MainUserDlgProc,_T("Estereoscopía"),(LPARAM)0, 0);
                                                                   Referencia 2.22
                m_pmapPARAMETERS = CreateCPParamMap(NULL,0,NULL,ip,hInstance,
                                MAKEINTRESOURCE(IDD_DIALOGO_NULO), _T("Parámetros"),
                                APPENDROLL_CLOSED);

                // Apartir de este método se instancian las demas variables
                if(GetUserCamera()) {
                        ActivaControles();
                                                                   Referencia 2.29

                        m_coUser = (CameraObject *) m_nodeCamUser->GetObjectRef();
                        AjustarFOV();
                        m_coUser->SetClipDist(ip->GetTime(), CAM_HITHER_CLIP,
                                                m_fVisionPlaneDist);
                                                                   Referencia 2.45
                        notifyObject.SetNotifyFunc(ProcessRefMessage, (DWORD_PTR)this);
                        CreaReferencia();                          Referencia 2.39
```

```
                        //Crear un diálogo no modal como ventana estereoscópica
                                                                        Referencia 2.13
                        m_hwndStereoPanel=CreateDialogParam(hInstance,
                                        MAKEINTRESOURCE(IDD_ESTEREO),
                                        ip->GetMAXHWnd(), MainDlgProc,
                                        (LPARAM)this);

                        if(m_hwndStereoPanel) {
                                RECT r;
                                ip->RegisterDlgWnd(m_hwndStereoPanel);

                                MoveWindow(m_hwndStereoPanel,0,0,
                                VIEW_WIDTH,VIEW_HEIGHT,TRUE);
                                GetClientRect(m_hwndStereoPanel,&r);
                                int nWidth  = VIEW_WIDTH  - (r.right-r.left);
                                int nHeight = VIEW_HEIGHT - (r.bottom-r.top);
                                MoveWindow(m_hwndStereoPanel,
                                                0, 0,
                                                VIEW_WIDTH+nWidth,
                                                VIEW_HEIGHT+nHeight, TRUE);
                        }
                }
        }
}
```

Referencia 2.51

```
void UtilTest::EndEditParams(Interface *ip,IUtil *iu) {
        //nodeCamUser,camUser no se eliminan porque
        //apuntan al nodo que creo el usuario

        if(m_hwndStereoPanel) {
                ip->UnRegisterDlgWnd(m_hwndStereoPanel);
                DestroyWindow(m_hwndStereoPanel);                       Referencia 2.14
                m_hwndStereoPanel=NULL;
        }

        if (m_bmIzq  ) m_bmIzq->DeleteThis();
        if (m_bmDer ) m_bmDer->DeleteThis();

        notifyObject.ResetNotifyFunc();

        if (m_pmapPARAMETERS) {
                DestroyCPParamMap(m_pmapPARAMETERS);                    Referencia 2.26
                m_pmapPARAMETERS = NULL;
        }
        ip->DeleteRollupPage(m_hwndPanel);                              Referencia 2.9

        m_hwndPanel = NULL;

        // Liberar puntero a la ventana activa
        ip->ReleaseViewport(m_veActiveViewport);
                this->iu = NULL;
        this->ip = NULL;
}
```

Referencia 2.23

```
BOOL UtilTest::SetValue(int i, TimeValue t, float v) {
        switch (i) {
        case PM_DISTANCIA_PLANO:
                CrearPlanoZPS();
                m_fVisionPlaneDist = v;
                PlanoZPS_TM();
                if(m_nodeZPS)ip->DeleteNode(m_nodeZPS);          Referencia 2.30
                if(m_coUser)
                        m_coUser->SetClipDist(ip->GetTime(), CAM_HITHER_CLIP,
                                                m_fVisionPlaneDist);
                break;
        case PM_MAGNITUD_ESTEREO:
                m_fStereoMagnitudeAdj = v;
                break;
        case PM_AJUSTAR_FOV:
                m_fFOV = v;
                AjustarFOV();
                break;
        case PM_AJUSTAR_HIT:
                m_fHIT = v;
                AjustarHIT();
                break;
        }
        return TRUE;
}
```

Referencia 2.23

```
BOOL UtilTest::GetValue(int i, TimeValue t, float &v, Interval &ivalid) {

        switch (i) {
        case PM_DISTANCIA_PLANO:
                v = m_fVisionPlaneDist;
                 break;
        case PM_MAGNITUD_ESTEREO:
                v = m_fStereoMagnitudeAdj;
                break;
        case PM_AJUSTAR_FOV:
                v = m_fFOV;
                break;
        case PM_AJUSTAR_HIT:
                v = m_fHIT;
                break;
        }
        return TRUE;
}
```

```
BOOL UtilTest::GetUserCamera() {
	// Obtener la ventana activa a representar (tiene que ser Cámara Libre)
	m_veActiveViewport = ip->GetActiveViewport();
	m_nodeCamUser= m_veActiveViewport->GetViewCamera();                    Referencia 2.31
	if(m_nodeCamUser)                                                      Referencia 2.31
	{
		ObjectState os = m_nodeCamUser->EvalWorldState(ip->GetTime()); Referencia 2.31
		//si la ventava activa es una camara libre,
		//la función regresa TRUE
		if (os.obj->SuperClassID() == CAMERA_CLASS_ID &&
			os.obj->ClassID() == Class_ID(SIMPLE_CAM_CLASS_ID,0))
			return TRUE;
	}

	return FALSE;
}

void UtilTest::ActivaControles() {
	// IDD_MENU
	EnableWindow(GetDlgItem(m_hwndPanel, IDC_RENDER),          TRUE);
	EnableWindow(GetDlgItem(m_hwndPanel, IDC_VAL_DEFECTO),TRUE);
	EnableWindow(GetDlgItem(m_hwndPanel, IDC_SAVE_IMAGE),   TRUE);

	m_pmapPARAMETERS = ReplaceCPParamMap(m_pmapPARAMETERS->GetHWnd(),
		descParams, PARAMETERS_DESC_LENGTH,this,ip, hInstance,
		MAKEINTRESOURCE(IDD_PARAMETERS), _T("Parámetros"),
		APPENDROLL_CLOSED);
}

void UtilTest::CrearPlanoZPS() {
	// Crear un nuevo objeto (caja)
	Object *obj = (Object*)ip->CreateInstance(
		GEOMOBJECT_CLASS_ID,
		Class_ID(BOXOBJ_CLASS_ID,0));
	assert(obj);

	// Obtener el bloque de parámetros
	IParamArray *iBoxParams = obj->GetParamBlock();
	assert(iBoxParams);

	// Poner los valores de los parámetros
	int height = obj->GetParamBlockIndex(BOXOBJ_HEIGHT);
	assert(height>=0);
	iBoxParams->SetValue(height,TimeValue(0),0.2f);                        Referencia 2.16

	int width = obj->GetParamBlockIndex(BOXOBJ_WIDTH);
	assert(width>=0);
	iBoxParams->SetValue(width,TimeValue(0),50.0f);                        Referencia 2.17

	int lenght = obj->GetParamBlockIndex(BOXOBJ_LENGTH);
	assert(lenght>=0);
	iBoxParams->SetValue(lenght,TimeValue(0),40.0f);                       Referencia 2.18
```

```
// Crear el nodo en la escena
        m_nodeZPS = ip->CreateObjectNode(obj);                                  Referencia 2.32

        m_nodeZPS->Freeze(TRUE);                                                Referencia 2.32
        m_nodeZPS->SetRenderable(FALSE);                                        Referencia 2.32
        m_nodeZPS->SetWireColor(RGB(0.0f,0.0f,255.0f));                         Referencia 2.32

        // Nombrar el nodo y hacer el nombre único
        TSTR name(_T("Plano ZPS "));
        ip->MakeNameUnique(name);
        m_nodeZPS->SetName(name);                                               Referencia 2.32

        m_nodeCamUser->AttachChild(m_nodeZPS);                                  Referencia 2.32
        PlanoZPS_TM();
}


void UtilTest::PlanoZPS_TM() {
        Matrix3 tmat1(1); // Matriz Identidad                                   Referencia 2.36
        Matrix3 tmat2(1); // Matriz Identidad                                   Referencia 2.36
        Point3 p;

        tmat2 = tmat1 = m_nodeCamUser->GetNodeTM(ip->GetTime());Referencia 2.33 , Referencia 2.36
        tmat1.NoRot();                                                          Referencia 2.36
        tmat1.NoTrans();                                                        Referencia 2.36
        tmat1.SetIdentFlags(tmat1.GetIdentFlags() & ~POS_IDENT);                Referencia 2.36
        m_nodeCamUser->SetNodeTM(ip->GetTime(),tmat1);          Referencia 2.33, Referencia 2.36

        p.x = 0.0f; p.y = 0.0f; p.z = m_fVisionPlaneDist*-1.0f;
        tmat1 = m_nodeZPS->GetNodeTM(ip->GetTime());            Referencia 2.33 Referencia 2.36
        tmat1.NoRot();                                                          Referencia 2.36
        tmat1.NoTrans();                                                        Referencia 2.36
        tmat1.SetTrans(p);                                                      Referencia 2.36
        tmat1.SetIdentFlags(tmat1.GetIdentFlags() & ~POS_IDENT);                Referencia 2.36
        m_nodeZPS->SetNodeTM(ip->GetTime(), tmat1);             Referencia 2.33 Referencia 2.36

        m_nodeCamUser->SetNodeTM(ip->GetTime(),tmat2);          Referencia 2.33 Referencia 2.36
        ip->RedrawViews(ip->GetTime());
}


void UtilTest::AjustarFOV() {
        m_coUser->SetFOV(ip->GetTime(),DegToRad(m_fFOV));
        ip->RedrawViews(ip->GetTime());
}
```

```
////////////////////////// Sección OpenGL ////////////////////////////////////////
```

Referencia 1.8, Referencia 1.10, Referencia 1.12

```
void UtilTest::AjustarHIT() {
// La alineación horizontal deberá estar puesta, tal que, algunos elementos
// de la escena aparezcan con paralaje negativo y otros elementos de la
// escena aparezcan con paralaje positivo.

	if(theUtilTest.m_bRenderFirstFrame) {
		glDrawBuffer(GL_BACK);
		glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

		glFlush();
		glDrawBuffer(GL_BACK_LEFT);//Vista Izquierda
		glMatrixMode(GL_PROJECTION);
		glLoadIdentity();
		glOrtho(0.0, (GLdouble)VIEW_WIDTH, 0.0,(GLdouble)VIEW_HEIGHT,-1.0,1.0);
		glMatrixMode(GL_MODELVIEW);
		glLoadIdentity();
		glTranslatef(-m_fHIT, 0.0f, 0.0f);
		glCallList(1);

		// Limpiar debido a que algunas tarjetas
		// graficas tienen z-buffer compartido
		glClear(GL_DEPTH_BUFFER_BIT);

		glFlush();
		glDrawBuffer(GL_BACK_RIGHT);//Vista Derecha
		glMatrixMode(GL_PROJECTION);
		glLoadIdentity();
		glOrtho(0.0, (GLdouble)VIEW_WIDTH, 0.0,(GLdouble)VIEW_HEIGHT,-1.0,1.0);
		glMatrixMode(GL_MODELVIEW);
		glLoadIdentity();
		glTranslatef(m_fHIT, 0.0f, 0.0f);
		glCallList(2);
		glFlush();
		SwapBuffers(hdc);
		glFlush();
	}
}
```

```
void UtilTest::RenderFrame() {
	SetDlgItemText(m_pmapPARAMETERS->GetHWnd(),IDC_INFO_TC,_T("\0"));

	if (!m_bmIzq) {
		BitmapInfo bi;
		bi.SetWidth(VIEW_WIDTH);
		bi.SetHeight(VIEW_HEIGHT);
		bi.SetType(BMM_TRUE_64);
		bi.SetFlags(MAP_HAS_ALPHA);
		bi.SetAspect(1.0f);
		m_bmIzq  = TheManager->Create(&bi);
		m_bmDer  = TheManager->Create(&bi);
	}

	m_fHIT=0.0f;
```

Referencia 2.25

```
	IParamBlock *pblk=(IParamBlock*)m_pmapPARAMETERS->GetParamBlock();
	pblk->SetValue(PM_AJUSTAR_HIT,GetCOREInterface()->GetTime(),0.0f);
	m_pmapPARAMETERS->SetParamBlock((IParamArray *)pblk);

	if(m_bRenderFirstFrame) {
		glDeleteLists(1,1);
		glDeleteLists(2,1);
	}

	m_bmIzq->Display(_T("Creando Imágen Izquierda"),BMM_LL);
	StereoProjection(m_bmIzq,-1.0);

	m_bmDer->Display(_T("Creando Imágen Derecha"),BMM_LL);
	StereoProjection(m_bmDer,1.0);

	m_bRenderFirstFrame=TRUE;
	AjustarHIT();

	if(IsDlgButtonChecked(m_hwndPanel,IDC_SAVE_IMAGE)==BST_UNCHECKED) {
		m_bmIzq->UnDisplay();
		m_bmDer->UnDisplay();
	}
}
```

```
void UtilTest::StereoProjection(Bitmap *bm,double EyePosition) {
// Ajustar FOV un 20% más del que se pretende usar, ya que se necesitará un
// ancho extra para facilitar la alineación horizontal( HIT ), por ejemplo:
// de 50 grados a 60 grados.

        Matrix3 tmat1(1); // Matriz Identidad                                    Referencia 2.37
        Matrix3 tmat2(1); // Matriz Identidad                                    Referencia 2.37
        Point3 offset;
        TSTR info(_T("0.000000"));

        m_coUser->SetFOV(ip->GetTime(),DegToRad(m_fFOV*1.20f));          Referencia 1.6
                                                  Referencia 1.2, Referencia 1.3, Referencia 1.4
        float tc=STEREO_MAGNITUDE_CONSTANT*m_fStereoMagnitudeAdj*
                  (m_fVisionPlaneDist*2.0f*tan(m_coUser->GetFOV(ip->GetTime())/2.0f));
        if(tc) {
                  tc/=2.0f;                                                      Referencia 1.5
                  tc*=(float)EyePosition;
                  info.printf(_T("%f"),tc);
        }

        SetDlgItemText(m_pmapPARAMETERS->GetHWnd(),IDC_INFO_TC,info.data());

        offset.x = tc; offset.y = 0.0f; offset.z = 0.0f;
        tmat2 = tmat1 = m_nodeCamUser->GetNodeTM(ip->GetTime());Referencia 2.34 Referencia 2.37
        tmat1.NoRot();                                                           Referencia 2.37
        tmat1.NoTrans();                                                         Referencia 2.37
        tmat1.SetTrans(offset);                                                  Referencia 2.37
        tmat1.SetIdentFlags(tmat1.GetIdentFlags() & ~POS_IDENT);                 Referencia 2.37
        m_nodeCamUser->SetNodeTM(ip->GetTime(),tmat1*tmat2);Referencia 2.34 Referencia 2.37
        ip->OpenCurRenderer(m_nodeCamUser,NULL);
        ip->CurRendererRenderFrame(ip->GetTime(),bm);
        ip->CloseCurRenderer();
        m_nodeCamUser->SetNodeTM(ip->GetTime(),tmat2);          Referencia 2.34 Referencia 2.37
        EscribeMapaBits(bm,EyePosition);
        m_coUser->SetFOV(ip->GetTime(),DegToRad(m_fFOV));
        ip->RedrawViews(ip->GetTime());
}
```

Referencia 1.7

```
void UtilTest::EscribeMapaBits(Bitmap *bm,double EyePosition) {
// En la representación para la cámara izquierda se mostrará un 15% más para
// para la parte izquierda que para la parte derecha(5%) de la resolución en
// pantalla y viceversa.

        BMM_Color_64 *l64,*line64 = NULL;

        line64=(BMM_Color_64 *)calloc(VIEW_WIDTH,sizeof(BMM_Color_64));

        if(EyePosition == -1.0) glNewList(1, GL_COMPILE);
        if(EyePosition ==  1.0) glNewList(2, GL_COMPILE);

        for (int iy = 0; iy < VIEW_HEIGHT; iy++) {
                l64 = line64;
                int ix = (EyePosition==-1.0)?0:ASIMMETRIC_WIDTH;
                bm->GetPixels(ix,iy,VIEW_WIDTH-ASIMMETRIC_WIDTH,line64);

                for (;ix < ((EyePosition==-1.0)?VIEW_WIDTH-ASIMMETRIC_WIDTH:VIEW_WIDTH);
                    ix++,l64++)
                {
                        if(l64->r>>8&&l64->g>>8&&l64->b>>8&&l64->a>>8) {
                                glColor4ub(l64->r>>8,l64->g>>8,l64->b>>8,l64->a>>8);
                                glBegin(GL_POINTS);
                                glVertex2i(ix,VIEW_HEIGHT-iy-1);
                                glEnd();
                        }
                }
        }

        glEndList();

        if (line64)free(line64);
}
```

```
/*============================================================================*\
                        Métodos de NotifyMgr
\*============================================================================*/

NotifyMgr::NotifyMgr() {
        pNotifyFunc = NULL;
        ref0 = NULL;
}

NotifyMgr::~NotifyMgr() {DeleteAllRefsFromMe();}                    Referencia 2.44

Referencia 2.40
RefResult NotifyMgr::NotifyRefChanged(Interval changeInt,
        RefTargetHandle hTarget, PartID& partID,RefMessage message) {
        RefResult res = REF_SUCCEED;
        if (pNotifyFunc) {
                res = pNotifyFunc(changeInt, hTarget, partID, message, pPrivateData);
        }
        return res;
}

Referencia 2.41
int NotifyMgr::NumRefs() {return 1;}

Referencia 2.42
RefTargetHandle NotifyMgr::GetReference(int i) {
        switch (i) {
        case 0: return ref0;
        }
        return NULL;
}

Referencia 2.43
void NotifyMgr::SetReference(int i, RefTargetHandle rtarg) {
        switch (i) {
        case 0: ref0 = rtarg; break;
        }
}

BOOL NotifyMgr::CreateReference(RefTargetHandle hTarget) {
        MakeRefByID(FOREVER, 0, hTarget);                           Referencia 2.39
        return TRUE;
}

BOOL NotifyMgr::RemoveReference() {
        if (ref0) {
                DeleteReference(0);                                 Referencia 2.46
                ref0 = NULL;
        }
        return TRUE;
}
```

Referencia 2.45

```
void NotifyMgr::SetNotifyFunc(RefResult (*func)(Interval changeInt,
        RefTargetHandle hTarget, PartID& partID,RefMessage message,
                DWORD pData), DWORD pData) {
        pPrivateData = pData;
        pNotifyFunc = func;
}

void NotifyMgr::ResetNotifyFunc() {
        RemoveReference();
        pNotifyFunc = NULL;
}

RefResult ProcessRefMessage(Interval changeInt, RefTargetHandle hTarget,
        PartID& partID, RefMessage message, DWORD pData) {

        UtilTest* ut = (UtilTest *)pData;

        if (message == REFMSG_TARGET_DELETED ||
                message == REFMSG_REF_DELETED)ut->iu->CloseUtility();

        return REF_SUCCEED;
}

void UtilTest::CreaReferencia() {
        notifyObject.RemoveReference();
        notifyObject.CreateReference((RefTargetHandle)m_nodeCamUser);
}
```

Referencia 2.35

Componente Estereoscópico para 3DS MAX Versión 5 □

Este capítulo presenta la información correspondiente para operar el componente en 3dsmax, dividida en dos partes: Inicialización y Descripción de Parámetros.

4.1 Inicialización del Componente

Para abrir el componente bastará con ir al panel utilidades dando un clic sobre el botón "Estereoscopía [UASLP]" **(Figura 4.1)**, lo que permitirá el despliegue de las dos persianas sobre el panel de comandos que lo componen.

Figura 4.1 Componente Estereoscopía.

Si el botón antes mencionado no aparece, habrá que agregarlo seleccionándolo de la lista de componentes que aparece al dar un clic en el botón situado a la derecha del botón "Sets", indicado por un rectángulo rojo en la **Figura 4.2**, una vez encontrado el título "Estereoscopía [UASLP]" en la lista, habrá que posicionar el icono del mouse sobre el título dando un clic y dejándolo presionado sobre el título arrastrándolo a una nueva posición en la lista de botones que aparece a la derecha de la lista, ver la parte de la derecha de la **Figura 4.2**.

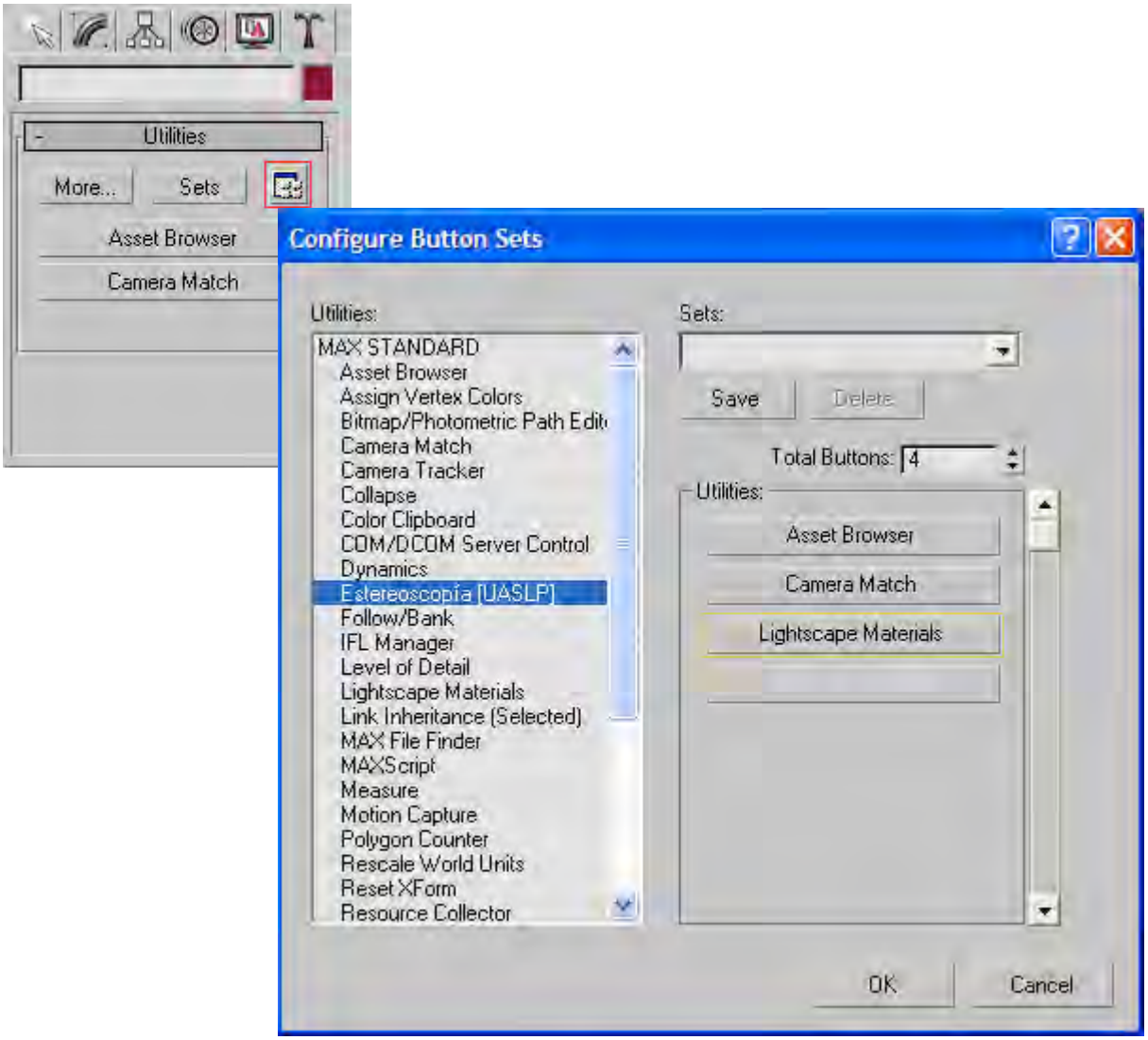

Figura 4.2 Ventana Configurar Conjunto de Botones.

En la **Figura 4.3** se puede observar el caso en el que aparecen algunos controles deshabilitados, esto puede darse debido a que no se ha definido una vista de cámara libre como la vista activa y seleccionada, una vez que se ha definido dicha vista **(Figura 4.4)**, bastará con cerrar el componente y volverlo abrir para habilitar todas las opciones como se muestra en la **Figura 4.5**.

Este componente proporciona todo lo que es necesario para crear visión estéreo desde una escena 3D. El componente genera dos imágenes estéreo basadas en la vista de cámara libre creada y seleccionada.

Si la computadora no esta equipada con una tarjeta gráfica con soporte estéreo, las opciones para este componente estarán disponibles, sin embargo no habrá efecto estereoscópico aunque se utilizaran lentes de cristal líquido para estereoscopía.

Figura 4.3 Controles Deshabilitados.

Figura 4.4 Vista de Cámara Libre.

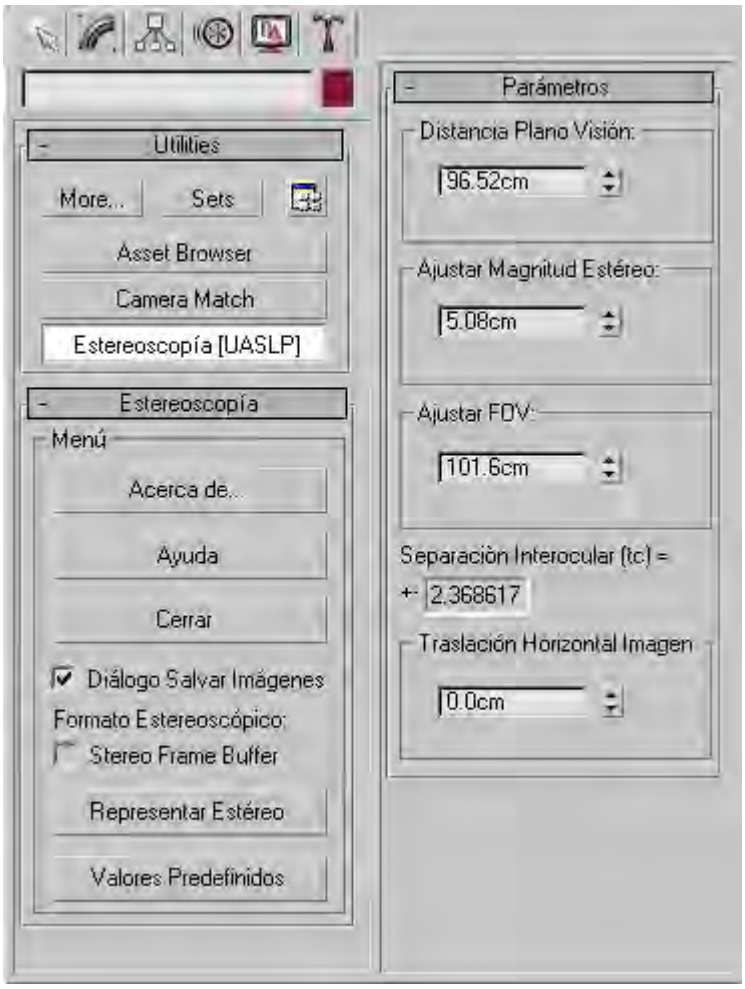

Figura 4.5 Controles Habilitados.

Figura 4.6 Persiana Estereoscopía.

4.2 Descripción de Parámetros

El componente consta de dos persianas desplegables:

- Persiana Estereoscopía.
- Persiana Parámetros.

En la **Figura 4.6** se muestra la persiana Estereoscopía, la cual contiene un menú con opciones básicas para el manejo del componente las cuales se describen a continuación:

- **Acerca de**

Está opción muestra un cuadro de diálogo con una descripción acerca del objetivo de creación del componente **(Figura 4.7)**.

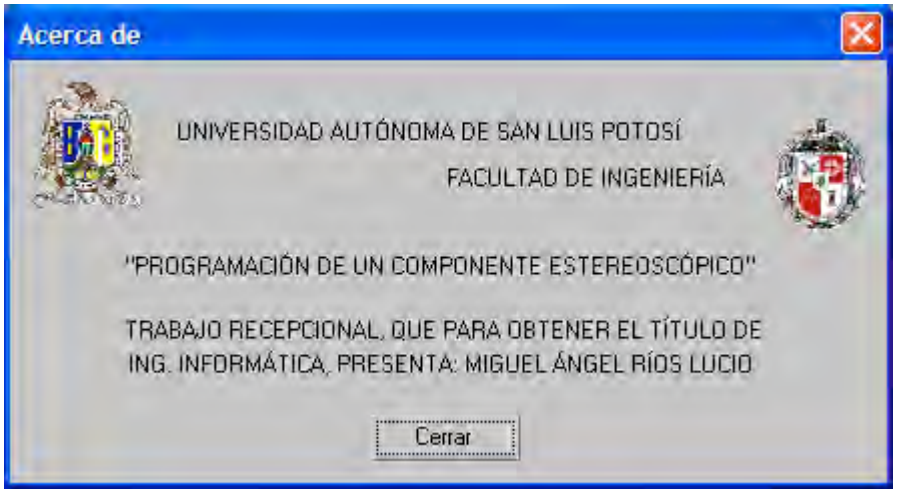

Figura 4.7 Cuadro de Diálogo Acerca de.

- **Ayuda**

Está opción muestra un cuadro de diálogo indicando que el archivo de ayuda del componente está disponible en el menú Ayuda de 3dsmax **(Figura 4.8)**.

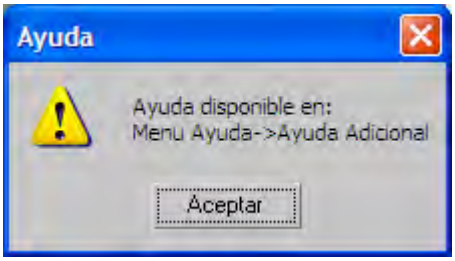

Figura 4.8 Ventana Ayuda.

La **Figura 4.9** muestra la ruta para encontrar el archivo de ayuda.

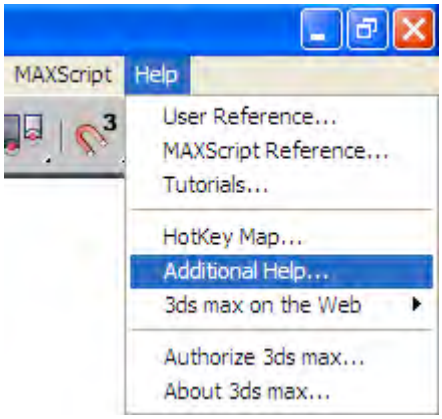

Figura 4.9 Menú Help.

Para ver el archivo de ayuda habrá que seleccionar en la ventana "Additional Help" **(Figura 4.10)**, el título "Estéreo UASLP" con un clic con el botón izquierdo del Mouse, sobre el título, posteriormente dar un clic en el botón "Display Help". Esta acción abrirá una nueva ventana mostrando el archivo de ayuda en formato HTML compilado.

Figura 4.10 Ventana Additional Help.

- **Cerrar**

Esta opción permite cerrar el componente, ocultando del panel de comandos las dos persianas.

- **Cuadro de Diálogo Salvar Imágenes**

Esta opción deja visibles los cuadros de diálogo al activar la casilla de verificación "Diálogo Salvar Imágenes" que muestran las representaciones izquierda y derecha de la vista de cámara original, permitiendo guardar las imágenes en una gran variedad de formatos de archivos de imagen **(Figura 4.11)**.

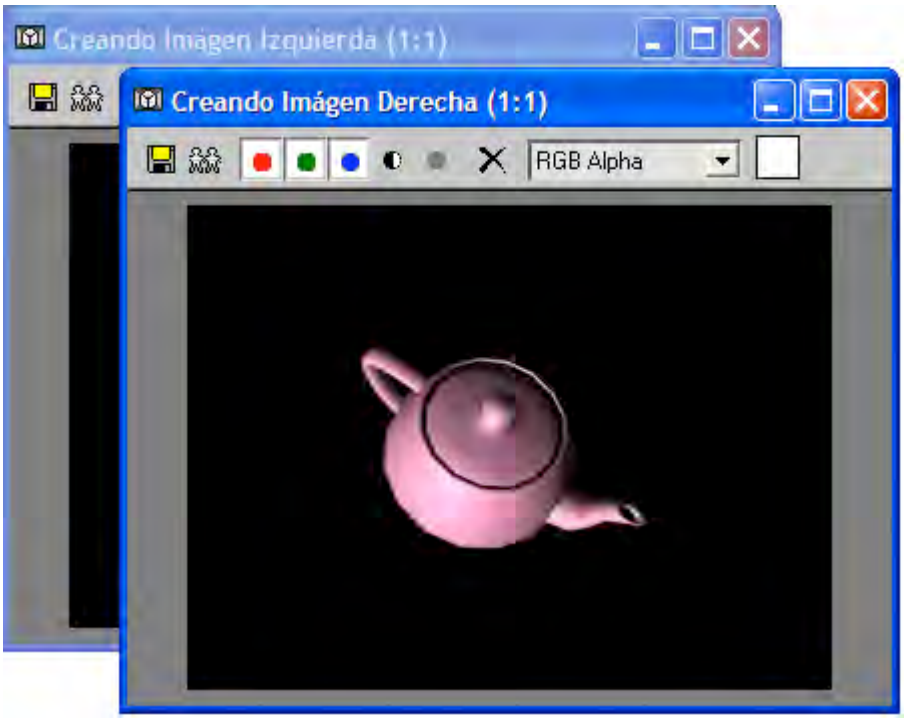

Figura 4.11 Cuadros de Diálogo Salvar Imágenes.

□ **Zona de Imagen Estéreo**

La Zona de Imagen estéreo es una ventana similar a la Zona de Imagen Virtual. Esta ventana desplegará una ventana estéreo con intercambio de páginas que puede ser vista con lentes para estereoscopía **(Figura 4.12)**.

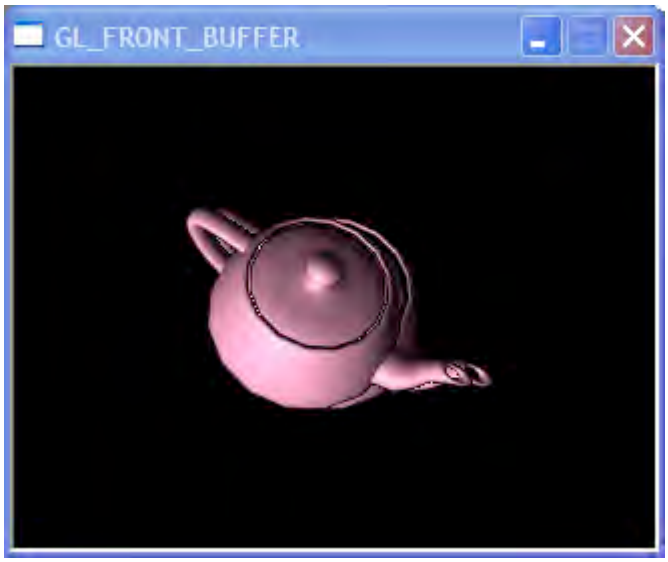

Figura 4.12 Ventana Zona de Imagen Estéreo.

- **Formato Estereoscópico (Stereo Frame Buffer)**

  Esta opción activa la casilla de verificación si es detectada una tarjeta gráfica con soporte estéreo.

- **Representar Estéreo**

  Esta opción obtiene las imágenes correspondientes para el ojo derecho e izquierdo de una escena 3D, trasladándolas de la zona de imagen virtual a las zonas de imagen estéreo y activando los lentes.

- **Valores Predefinidos**

  Esta opción regresa a los valores originales en la persiana parámetros. Los valores predefinidos son un buen punto de partida para crear imágenes estéreo cómodas de ver.

La **Figura 4.13** muestra la persiana Parámetros la cual contiene opciones básicas para controlar el efecto estereoscópico de una escena 3D, tales parámetros se describen a continuación:

Figura 4.13 Persiana Parámetros.

- **Distancia al Plano de Visión**

  Esta opción permite mover el plano de visión, mas distante a la escena (clics en el botón con la flecha en dirección superior) o más cercano a la cámara de representación (clics en el botón con la flecha en dirección inferior).

Un buen lugar para colocar el plano de visión es en el centro del objeto(s) de interés en la escena. Colocando el plano en el centro del objeto, cualquier parte que quede adelante del plano se proyectará con paralaje negativo y dará la sensación de estar en frente del monitor y cualquier parte que quede detrás del plano se proyectará con paralaje positivo y dará la sensación de estar adentro del monitor.

□ **Ajustar Magnitud Estéreo**

Esta opción permite ajustar la magnitud deseada de efecto estereoscópico. Con un valor de 0.0 se obtiene un efecto estereoscópico nulo, un buen valor es 1.0, un valor de 2.0 resulta un efecto bastante fuerte, quizá incomodo de ver.

Para aumentar el efecto dar un clic ó más sobre el botón con la flecha en dirección superior, ó un clic ó más sobre el botón con la flecha en dirección inferior para disminuir.

□ **Ajustar FOV (Campo de Visión)**

Permite cambiar el FOV. El campo de vista es parecido al afecto que se obtiene usando un lente zoom de una cámara. Con valores pequeños el campo será angosto y creará un efecto parecido a un lente telefoto, si los valores son largos (40 grados o más) el efecto es parecido a un lente gran angular, y frecuentemente resalta el efecto estéreo.

Para aumentar e FOV dar un clic ó más sobre el botón con la flecha en dirección superior, ó un clic ó más sobre el botón con la flecha en dirección inferior para disminuir.

□ **Separación Interaxial**

Esta opción indica en el cuadro de texto (solo lectura), la cantidad de traslación sobre el eje de las *x,* hacia la izquierda para la cámara izquierda y hacia la derecha para la cámara derecha ambas desde la posición de la cámara del usuario seleccionada y activa, configurada mediante los parámetros mencionados.

□ **Traslación Horizontal de Imagen**

La consideración más importante es crear una escena estéreo con la cantidad correcta de paralaje. Mucho paralaje, cuando las cámaras están muy separadas provoca dificultad en los observadores para fusionar las dos imágenes en una con profundidad estereoscópica y una cantidad pequeña genera una imagen estéreo con poco paralaje y percibiríamos una imagen común ó con poca profundidad estéreo.

El 7% de paralaje (la distancia entre los puntos de imagen correspondientes izquierda y derecha) está basado en el ancho de la pantalla. Hay una relación entre el objetivo/ distancia de la cámara y la escena / ancho de la pantalla.

Para aumentar la distancia entre las dos imágenes dar un clic ó más sobre el botón con la flecha en dirección inferior, esto hace que los objetos aparezcan en frente del monitor ó un clic ó más sobre el botón con la flecha en dirección superior para disminuir la distancia haciendo aparecer los objetos detrás del monitor.

CONCLUSIONES

Las tecnologías que hacen posible la realidad virtual se han dado apenas en los últimos 25 años. En 1965 Ivan Sutherland habló acerca de los mundos virtuales y en 1966 llevó acabo los primeros experimentos en tres dimensiones. Tres años después demostró el primer sistema capaz de sumergir a la gente en pantallas de información en tres dimensiones.

El paradigma de la realidad virtual estuvo casi perdido, siendo usado en los Estados Unidos sin otros propósitos que los militares. En lo militar se invirtieron millones de dólares en el desarrollo de programas de realidad virtual como simuladores de vuelos. Hasta finales de los ochenta otros países mostraron muy poco interés por esta tecnología. En concreto Japón, Alemania y Francia.

Los avances tecnológicos en la antepenúltima década incluyeron grandes avances en tres campos independientemente de la realidad virtual:

- Dispositivos de presentación en pantalla de cristal líquido (LCD) y tubos de rayos catódicos (CRT);
- Estaciones de trabajo de gráficos a alta velocidad y resolución para producir las imágenes;
- Sistemas de rastreo para convertir la información sobre la orientación y la posición en señales que pueden ser leídas por las computadoras y reflejadas en imágenes.

Estas tres tecnologías se pusieron al alcance en sistemas relativamente accesibles permitiendo a los investigadores aplicar la tecnología en más campos que el militar. Hoy en día, todos los componentes hardware de un sistema virtual están a la venta y muchas aplicaciones ya están siendo utilizadas, una situación muy diferente a la de hace 25 años. Mientras las tecnologías continúen evolucionando y haciéndose accesibles, más aplicaciones podrán ser desarrolladas.

En mi trayectoria de estudiante con apenas un libro leído en cuanto al paradigma de la realidad virtual pero con bases en gráficos en 3D y programación, me nació la curiosidad por aprender más acerca de este paradigma y decidí iniciar a adentrarme en este rama comenzando proponiendo este trabajo recepcional para investigar por uno de los componentes primarios para un sistema virtual o mejor conocida como: visión estereoscópica.

De entre todos nuestros sentidos, la vista es la primera receptora de la información. Lo que vemos es recibido como una imagen en nuestra retina, traducido a símbolos y enviado a nuestra mente. Es ahí donde nosotros reconstruímos y sintetizamos la información que hemos recibido en algo que ya conocemos.

Invirtiendo el proceso, cuando queremos transmitir una idea o una imagen desde nuestra mente, lo expresamos en signos (palabras, dibujos, gestos…) que pueden ser entendidos por los demás directa o indirectamente, por ejemplo con la ayuda de las computadoras. De este modo los mensajes pasan de unas personas a otras.

Las computadoras colaboran en este proceso mediante el almacenamiento de la información (que ha sido previamente traducida a símbolos) de manera que puede ser expuesta, por ejemplo mediante un dibujo en la pantalla o mediante la visión estereoscópica. De esta manera una persona puede acceder la información, descifrar mentalmente los símbolos y recrear la idea.

Pero un mensaje no solo depende de su contenido, también depende de como sea recibido y del conocimiento del receptor. Cualquier cosa que interfiera entre el emisor y el receptor, aunque el sistema sea muy bueno, puede poner en peligro la comunicación entre ambos, a veces son las computadoras las que entorpecen el proceso. El grado de eficacia en la transmisión de la información depende en gran medida de la exactitud con que la versión reproducida represente la idea original y de la capacidad del receptor para captar la idea original de lo que le es expuesto.

Durante el desarrollo de este trabajo me encuentro la Página Web de la firma StereoGraphics una compañía estadounidense cuyos propósitos son desarrollar sistemas de realidad virtual y proveer equipo para dichos sistemas, este encuentro fue de gran ayuda aportando gran parte de la bibliografía. Mi contribución es la programación del componente para exponer la técnica de reproducción de imágenes usada para lograr la visión estereoscópica para "3D Studio Max 5" carente de tal función.

La realidad virtual explota muchas técnicas de reproducción de imágenes, usándolas dentro del entorno en el que el usuario puede examinar, manipular e interaccionar con los objetos expuestos. De este modo, investigadores y usuarios son capaces de usar imágenes para trasmitir, no solo la información sino también la capacidad de interpretarla.

La realidad virtual es finalmente un excelente medio para alcanzar un alto nivel de comunicación, mejor y más efectivo de lo que haya existido nunca y también un excelente medio para alentar la intuición.

BIBLIOGRAFÍA

[1] Stereographics Developers Handbook, Background on Creating Images for CrystalEyes and SimulEyes. 1997, StereoGraphics Corporation.

[2] Gráficas por Computadora, Segunda Edición. Donald Eran, M. Pauline Baker. Prentice Hall, 1995, ISBN: 968-880-482-7.

[3] CrystalEyes Software Development Kit A StereoGraphics Product. 1997, StereoGraphics Corporation.

[4] Writing Stereoscopic Software for StereoGraphics. Bob Akka. 1998, StereoGraphics Corporation.

[5] 3D Studio MAX, Animación Tridimensional Para Cine, Video y Multimedia. Daniel Vendetti. Users, 2000, ISBN: 987-526-039-8.

[6] 3D Studio MAX 5 Software Development Kit Reference. 2002, Autodesk Corporation.

[7] Programación en C. Byron S. Gottfried. Mc Graw Hill, 1991, ISBN: 84-7615-572-7.

[8] Apuntes de la Materia Programación Orientada a Objetos. Facultad de Ingeniería, UASLP.

[9] Charles Petzold, Programming Windows 95 (Redmond, Washington: Microsoft Press, 1996). ISBN: 1-55615-676-6.

[10] Jeffrey Richter, Programing Applications for Microsoft Windows (Microsoft Press, 1999). ISBN: 1-57231-996-8.

[11] OpenGL Programing Guide (Addison-Wesley, 1999). Mason Woo, Jackie Neider, Tom Davis, Dave Shreiner. ISBN: 0-201-60458-2.

GLOSARIO

Terminología General relacionada en Estereoscopía

Ajuste: Enfoque que realizan los ojos.

Binocular: Dos ojos.

Componente: Véase **Plugin**.

Convergencia: La rotación de los ojos, en dirección horizontal, produce fusión.

Distancia Interaxial: La distancia entre los lentes de 2 cámaras.

Distancia Interocular: La distancia entre los ejes de los ojos. Leer **t**.

Disparidad: La distancia entre los puntos de las imágenes izquierda y derecha en la sobreposición de retinas, algunas veces llamada **Disparidad Retinal**. El término correspondiente para un despliegue en pantalla es **Paralaje**.

Disparidad Retinal: Véase **Disparidad**.

Espacio CRT: Es la región aparentando estar dentro de la pantalla ó detrás de la superficie de la pantalla. Las imágenes con paralaje positivo aparentarán estar en el espacio CRT.

Espacio del Observador: Es la región entre la superficie de la pantalla y el observador. Las imágenes aparecerán en esta región si tienen paralaje negativo.

Estéreo: Es la abreviatura de estereoscópico.

Estereopsis: Es la sensación de profundidad binocular.

Estereoscopía: Es el arte o la ciencia de crear imágenes con una sensación de profundidad binocular.

Fantasmas: Es la percepción de interferencia.

Fusión: La combinación por la mente de las imágenes (izquierda y derecha), vistas por los ojos en una única imagen.

HIT: Es el desplazamiento horizontal de las dos imágenes para cambiar el valor de la paralaje de los puntos correspondientes. El término **Convergencia** ha sido usado para denotar este concepto.

Infrarrojo (IR): Radiación invisible usada en la comunicación para lentes de entornos virtuales.

Interferencia: Es la isolación incompleta de los canales de imagen (izquierdo y derecho). La interferencia es una entidad física, mientras que los fantasmas son entidades psicofísicas.

Paralaje: La distancia entre los puntos de las imágenes (izquierda y derecha) en una computadora.

Puntos Correspondientes: Cuando los puntos de una imagen izquierda y derecha coinciden o se encuentran uno encima del otro.

Relación Ajuste/Convergencia: La relación habitual establecida entre el enfoque y la convergencia de los ojos, cuando se visualizan objetos.

Screen Surround: El área horizontal y vertical inmediatamente adyacente a la pantalla para la presentación estéreo.

t*:** En estereocopía, ***t es usada para denotar la distancia entre los ojos llamada **Distancia Interocular**. $\mathbf{t_c}$ es usada para denotar la distancia entre los lentes de las cámaras estereoscópicas.

Ventana Estéreo: Es la pantalla de una computadora conformada entre los límites horizontales y verticales de la pantalla.

ZPS: Configuración de cero paralaje. El término usado para describir la paralaje en la pantalla para colocar un objeto, o una porción del objeto, en el plano de la pantalla. ZPS puede ser controlado mediante HIT. Cuando los puntos de imagen tienen ZPS, por terminología se dirá que tienen "convergencia". Este término en este caso no se usa por que causaría confusión con la convergencia de los ojos, y debido a que la palabra implica rotación de la cámara ó algún algoritmo de software empleando rotación. Tal rotación produce invariablemente distorsión geométrica.

Terminología General relacionada con 3D Studio MAX

Arreglo Virtual: Cuando 3dsmax necesita acceder a un grupo de elementos guardados por un componente, un mecanismo de arreglo virtual es frecuentemente usado. El desarrollador asigna un arreglo entero de índices para cada elemento (0, 1, 2, ...). El desarrollador usa el índice para recuperar o guardar un valor apropiado.

Clase: Un tipo de dato definido por el usuario que puede consistir de funciones miembro y de datos miembro.

Callback: Es un puntero a una función, usado por un servidor para llamar al código que solicitó el servicio.

Campo Visual ó Campo de Visión (FOV): Ángulo en grados que abarca todo lo que se puede ver a través de una cámara o de un visor.

Clase Abstracta: Una clase abstracta es una clase que contiene al menos una función virtual pura. No se puede declarar una instancia de una clase base abstracta, solo se puede usar como una clase base cuando se declaran otras clases.

Controlador: Es un componente que sirve para controlar la animación de elementos. Hay una gran posibilidad de controladores. Por ejemplo, los controladores de transformación determinan la localización de los nodos en la escena, mientras que los controladores de expresiones permiten a los parámetros de un elemento ser controlados por una expresión matemática.

Clase Descriptiva: Una clase descriptiva provee información acerca de la clase de un componente. Los métodos definidos en esta clase acceden a tal información.

Controlador de Transformación: Un controlador de transformación es un controlador que controla una matriz (Matrix3). Los controladores de transformación controlan por ejemplo matrices de 4x3 usadas para definir la posición de nodos en la escena.

DLL: Librería de Enlace Dinámico, son librerías de código objeto que permiten a múltiples programas compartir código, datos y recursos. Los componentes para 3dsmax están implementados como DLLs.

Dispositivo de Contexto: Este es un término de la API de Windows. Un dispositivo de contexto es una estructura que define un conjunto de objetos gráficos, sus atributos asociados y los modos gráficos que afectan la salida. Los objetos gráficos incluyen una pluma para dibujar líneas, una brocha para pintar y rellenar, un mapa de bits para copiar o recorrer partes de la pantalla, una paleta para definir un conjunto de colores disponibles, una región para recorte y otras operaciones. Una aplicación nunca tiene acceso directo a un dispositivo de contexto; en vez de esto, esta opera en la estructura indirectamente llamando a varias funciones.

Instancia: Este término tiene varios significados. Un significado es la definición C++ de una instancia. Esto es simplemente la asignación de una clase (un objeto). Otro significado de la interfase de usuario de 3dsmax relacionada con una copia de un objeto, modificador, controlador, etc. Eso es conceptualmente similar a simplemente un puntero al original.

Intervalo: Esta es una clase definida en el SDK. Esta describe un periodo de tiempo usando un tiempo inicial y un tiempo final. Hay métodos de esta clase para recuperar y poner el tiempo inicial y final, calcular la duración, y verificar si un tiempo específico está en el intervalo.

ID de Clase: Este es el único ID requerido por los componentes. El SDK provee un programa para generar estos IDs de clase.

Intervalo Válido: Este es un término usado para describir un periodo de tiempo sobre el cual un elemento es válido.

ID de Súper Clase: Un ID de Súper Clase es una constante definida por el sistema describiendo el tipo de objeto que una clase representa. Algunos ejemplos son: GEOMOBJECT_CLASS_ID, CAMERA_CLASS_ID, LIGHT_CLASS_ID, SHAPE_ CLASS_ID, HELPER_CLASS_ID, SYSTEM_CLASS_ID, etc.

Jerarquía de Clase: Un sistema base y clases derivadas.

Modificador: Un modificador es un tipo de componente el cual altera un objeto de alguna forma.

Matriz Ortonormal: Una matriz ortogonal tiene un sistema de ejes donde cada eje está 90 grados de los demás (no están sesgados). Una matriz ortonormal también es ortogonal y la longitud de los vectores renglón son todas uno.

Matriz de Transformación: Transformaciones 2D de traslación, rotación y escala pueden ser representadas por una matriz de 3x2. Transformaciones 3D pueden ser representadas por una matriz de 4x3. Estas matrices son referidas como matrices de transformación.

Nodo: La representación visual de un objeto en las ventanas de 3dsmax es un nodo. Existe una correspondencia uno a uno entre lo que puede ser seleccionado y movido en la ventana y un nodo. Los nodos soportan cinemática. Los nodos tienen un puntero a su padre y una lista de punteros a sus hijos. Los nodos también tienen un puntero a su controlador de transformación.

NTSC: Norma de video empleada en los Estados Unidos y en Japón. La velocidad es de 30 (29,97) cuadros por segundo (cps) ó de 60 campos por segundo, siendo un campo la mitad de las líneas de barrido entrelazadas de la pantalla de un televisor.

PAL: Es la norma de video en casi todos los países europeos y latinoamericanos. La velocidad es de 25 cuadros por segundo (cps) ó de 50 campos por segundo, siendo un campo la mitad de las líneas de barrido entrelazadas de la pantalla de un televisor.

Plugin: Un plugin es una función extra al programa que le es agregada para potenciar sus capacidades. En general los plugins son desarrollados y comercializados por empresas independientes a la creadora del software, aunque esta también puede desarrollar sus propios plugins.

Panel de Geometría: Es el sistema usado por 3dsmax que permite a un nodo ser alterado, quizás repetitivamente, a través de la aplicación de modificadores. En el comienzo del panel está el objeto base. Este es un objeto procedural (ó una malla). Al final del panel está el estado de espacio universal del objeto, este estado de espacio universal es el que aparece en las ventanas 3D. El panel consiste de objetos, objetos derivados y una lista de modificadores.

Procedimiento de Diálogo: Una función de Procedimiento de diálogo es una aplicación que procesa los mensajes enviados de un cuadro de diálogo, esta función es de tipo callback.

Representación (Render): Proceso en el cual se calculan y procesan todos los datos de luces, geometrías, materiales y efectos de la escena, y se crea una imagen bidimensional de tamaño configurable.

TimeValue: Es un tipo de dato que representa un simple instante en el tiempo.

Variable de Clase: En C++, cuando se modifica un dato miembro en una declaración de clase, la palabra clave estática específica que una copia del miembro está compartida para todas las instancias de la clase.