UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERIA

ESCUELA DE COMPUTACION

“DISEÑO Y DESARROLLO DE UNA APLICACIÓN PARA LA REPRESENTACION DE OBJETOS EN TRES DIMENSIONES”

TRABAJO DE GRADUACION

PREPARADO PARA LA FACULTAD DE INGENIERIA

PARA OPTAR AL GRADO DE

INGENIERO EN CIENCIAS DE LA COMPUTACION

PRESENTADO POR:

EDILBERTO ISMAEL LOPEZ CORTEZ

CIUADELA DON BOSCO MARZO 2003

INDICE

INTRODUCCION i

CAPITULO I

PLANTEAMIENTO DEL PROBLEMA

1.1. Antecedentes 1
1.2. Importancia y justificación 3
1.3. Proyección social 4
1.4. Objetivos
1.4.1. Objetivo general 5
1.4.2. Objetivos específicos 5
1.5. Alcances y limitaciones
1.5.1. Alcances 6
1.5.2. Limitaciones 8

CAPITULO II

MARCO TEORICO

2.1. OpenGL 9
2.2. Visualización tridimensional 10
2.3. Luz y Sombra 11
2.3.1. Iluminación Ambiental 12

2.3.2. Iluminación Difusa 13
2.3.3. Iluminación Especular 14
2.4. Transformación de Coordenadas 15
2.4.1. Transformaciones de Vista 16
2.4.2. Transformaciones de Modelaje 16
2.4.3. Transformaciones de Poyección 18

CAPITULO III

DISEÑO DEL PROTOTIPO

3.1. Estructura del Prototipo 20
3.2. Arbol de Opciones 24
3.3. Diseño de la interfaz de la aplicación 28

CAPITULO IV

DESARROLLO DEL PROTOTIPO

4.1. Diagrama de funcionamiento de la aplicación 31
4.2. Programa principal 35
4.3. Programa traductor 40
4.4. Programa dibujar 80
4.5. Programa Doom 102

CONCLUSIONES 106
RECOMENDACIONES 107

BIBLIOGRAFIA 109

GLOSARIO 110

ANEXOS 113

I. Manual del usuario

II. Manual del programador

LISTA DE FIGURAS

Figura 2.1

Como los ojos ven en tres dimensiones 10

Figura 2.2

Objeto sólido iluminado por una luz 11

Figura 2.3

Objeto iluminado con luz ambiental 12

Figura 2.4

Calculando el componente de color ambiental de un objeto 13

Figura 2.5

Objeto iluminado con luz difusa 14

Figura 2.6

Objeto iluminado con luz especular 15

Figura 2.7

Transformaciones de modelaje 17

Figura 2.8

Ejemplo de proyección ortográfica y perspectiva 19

Figura 3.1

Proceso para dibujar una escena 20

Figura 3.2

Arbol de opciones de la aplicación 24

Figura 3.3

Pantalla principal de la aplicación 28

Figura 3.4

Pantalla de dibujo con menú conceptual 30

Figura 3.5

Cuadro de dialogo de propiedades 30

Figura 4.1

Diagrama de funcionamiento programa Principal.cpp 31

Figura 4.2

Diagrama de funcionamiento programa Traductor.cpp 32

Figura 4.3

Diagrama de funcionamiento programa Dibujar.cpp 33

Figura 4.4

Diagrama de funcionamiento programa Doom.cpp 34

Figura 4.5

Flujograma inicializar 35

Figura 4.6

Flujograma de manejo de mensajes 36

Figura 4.7

Flujograma para obtener nombre de archivo 38

Figura 4.8

Cuadro de dialogo Abrir archivo de Windows 39

Figura 4.9

Flujograma para inicializar el traductor 40

Figura 4.10

Flujograma de traducción de instrucciones de tipo “Objeto” 42

Figura 4.11

Flujograma de traducción de instrucciones de tipo “Ambiente” 45

Figura 4.12

Flujograma de traducción de parámetros de objetos (a) 47

Figura 4.13

Flujograma de traducción de parámetros de objetos (b) 48

Figura 4.14

Flujograma de traducción de parámetros de ambiente (a) 67

Figura 4.15

Flujograma de traducción de parámetros de ambiente (b) 68

Figura 4.16

Flujograma para la revisión de valores 76

Figura 4.17

Flujograma general del programa Dibujar.cpp 80

Figura 4.18

Flujograma de inicialización de escena (a) 82

Figura 4.19

Flujograma de inicialización de escena (b) 83

Figura 4.20

Flujograma para el dibujo de objetos (a) 88

Figura 4.21

Flujograma para el dibujo de objetos (b) 89

Figura 4.22

Flujograma para el dibujo de escenas DOOM 102

Figura 4.23

Vértices de los polígonos que se pueden utilizar en la simulación DOOM 104

Figura 4.24

Forma de aplicar textura a los objetos 105

INTRODUCCION

La representación de objetos en tres dimensiones es hoy en día parte muy importante en las técnicas de modelaje, publicidad, animaciones, arquitectura e ingeniería, medicina, simulaciones aeronáuticas, etc. Por medio de ella es posible simular lugares que ya no existen, edificaciones que están por construirse o sitios imaginarios en los cuales se puede apreciar, desde una perspectiva, un punto de vista diferente que, en otro tipo de imágenes, no se alcanzaría.

Este tema ha tomado en la actualidad gran auge, debido al avance de la tecnología en el desarrollo de nuevo hardware y software para la representación de imágenes en tres dimensiones, apoyado con el incremento de la velocidad de las computadoras, tanto para el procesamiento de las operaciones, como para mostrar los gráficos. La realidad virtual abarca un gran campo de estudio. El uso original de este concepto se refería a sistemas de inmersión completo, que implicaba el empleo de cascos complejos para proyectar un espacio visual en tres dimensiones y trajes con dispositivos electrónicos para enviar y recibir señales sobre la posición del cuerpo, esto daba la sensación de un mundo virtual con el que podía interactuar.

Los programas que pueden presentar escenas tridimensionales son llamados navegadores (*entiéndase como: Visores de imágenes en tres dimensiones capaces de mostrar escenas virtuales, diferentes a los utilizados para navegar en Internet*) y

pueden ir desde una interacción simple con el usuario con el uso del mouse y el teclado, hasta una compleja, con guantes y cascos especiales.

El presente documento describe el análisis, el diseño y la implementación de una aplicación para representar objetos en tres dimensiones, los cuales son construidos en base a una serie de instrucciones contenidas en archivos de texto. Estas instrucciones están definidas y serán traducidas por la aplicación para poder dibujar los objetos. Además, está organizado en partes donde se presentan los antecedentes del tema, la importancia que tiene el desarrollar proyectos de tipo investigativo, así como la proyección social que pretende alcanzar. Se describen los objetivos de la investigación, los alcances y las limitaciones, estos combinados nos proporcionan un panorama de lo que se pretende desarrollar y una definición del tema.

También se presenta el marco teórico que nos da la definición de conceptos utilizados a lo largo del desarrollo del proyecto, el diseño del prototipo y su desarrollo, explicado mediante diagramas de flujo, para una mejor comprensión de cada una de sus partes.

Con el presente trabajo de graduación se pretende incursionar en la Representación de Imágenes en tres dimensiones; una parte fundamental de la Realidad Virtual.

CAPITULO I

PLANTEAMIENTO DEL PROBLEMA

1.1. ANTECEDENTES

Actualmente la creación de imágenes en tres dimensiones tiene una gran importancia no sólo en el área del entretenimiento, como juegos o películas, sino también en el área de la ciencia, investigación, educación, entrenamiento, construcción y para un sin fin de aplicaciones en las que se desarrollan simulaciones. Entre estas tenemos:

- Químicos pueden ver complejas moléculas en tiempo real y visualizar nuevas formas de crear materiales.
- Estudiantes pueden interactuar con la computadora, para conocer el sistema solar sin necesidad de utilizar una nave espacial.
- Pilotos pueden entrenarse sin la necesidad de volar o bien puede probarse un diseño costoso de un avión sin peligro de estrellarse.
- Un arquitecto y un cliente pueden ver una casa nueva, incluso caminar dentro de ella, antes de comenzar a construirla.
- Un turista puede visitar las ruinas de San Andrés estando en cualquier parte del mundo o bien conocer como eran éstas en el pasado.

Uno de los principales objetivos para el navegador es la representación de objetos tridimensionales en una pantalla que sólo tiene dos dimensiones, pues de esto depende el nivel de realismo que se quiera provocar e implica una gran cantidad de cálculos matemáticos y el uso de recursos de hardware. El número de cálculos se ve directamente reflejado en el tiempo de respuesta del navegador, haciéndolo más lento y, en consecuencia, provocando la perdida de la sensación del movimiento del objeto tridimensional.

En el país se ha trabajado muy poco en el área de gráficos tridimensionales y son muy limitados los antecedentes sobre este tema. Se han desarrollado aplicaciones utilizando gráficos tridimensionales como herramienta auxiliar, pero no como la propuesta en esta investigación, ya que ésta pretende demostrar las habilidades en el desarrollo de una aplicación propia para la simulación de objetos en tres dimensiones, que pueda ser utilizada para fines didácticos.

1.2. IMPORTANCIA Y JUSTIFICACION DEL TEMA

Las aplicaciones destinadas para la representación de objetos en tres dimensiones, son hoy en día de uso muy común entre personas que buscan plasmar sus ideas en imágenes dinámicas, donde se puedan apreciar diversos puntos de vista. De aquí la importancia de implementar una aplicación que pueda satisfacer esas necesidades y, que a su vez, sea de utilidad para fines didácticos, ya que actualmente este tipo de herramientas son para usuarios finales y no permiten tener acceso a su código para saber como éstas utilizan las instrucciones y representar los dibujos.

El desarrollo de la investigación tiene una justificación clara y definida, ésta radica en el hecho que en el país es poco lo trabajado sobre gráficas tridimensionales; se han desarrollado investigaciones sobre técnicas de dibujo en tres dimensiones, pero no una aplicación completa. Otro tipo de trabajos utilizan librerías de gráficos para dibujar objetos ya establecidos, como por ejemplo, dibujar tuberías en tres dimensiones, en el que se pueden hacer cálculos de caudales y pérdidas de presión del agua.

1.3. PROYECCION SOCIAL

Uno de los principales objetivos de la Universidad Don Bosco es la proyección social, y en los trabajos de graduación esto no se puede pasar por alto. Con el desarrollo de proyectos de este tipo, se pretende dejar precedentes a la comunidad estudiantil de las universidades salvadoreñas que quieren introducirse al área de gráficos tridimensionales.

La herramienta desarrollada traerá beneficios a los estudiantes y a la institución. A los estudiantes, para que puedan conocer los métodos y técnicas utilizados en la aplicación, los algoritmos empleados en su programación y los conceptos que se manejan en este tipo de programas. A la institución, ya que por medio de ésta se podrá contar con una aplicación para las prácticas de laboratorio de la materia gráficos por computadora. Además, los respectivos manuales de usuario y programador, en los que se detallará cada uno de los métodos aplicados, facilitarán entender el funcionamiento de la aplicación y los algoritmos utilizados.

1.4. OBJETIVOS

1.4.1. OBJETIVO GENERAL

Analizar, diseñar y desarrollar una aplicación para usos didácticos en el área de gráficos por computadora, que utilice métodos y técnicas de simulación, para representar objetos en tres dimensiones.

1.4.2. OBJETIVOS ESPECIFICOS

- Desarrollar una investigación sobre las técnicas para dibujar objetos en tres dimensiones utilizando C++ y OpenGL.

- Comprender como se efectúa la traducción de instrucciones de un lenguaje propio a figuras en tres dimensiones.

- Sentar las bases para futuras investigaciones en el diseño de aplicaciones orientadas a gráficos tridimensionales.

- Desarrollar una herramienta que pueda ser utilizada en las prácticas de laboratorio de la materia gráficos por computadora.

1.5. ALCANCES Y LIMITACIONES

1.5.1. ALCANCES

- Con el desarrollo de la aplicación se podrá:

  ⇒ Dibujar de entre un conjunto de diez figuras básicas, las cuales son: esfera, cilindro, disco, cubo, pirámide, línea, triángulo, cuadro, círculo y punto.

  ⇒ Aplicar iluminación ambiental, difusa y especular.

  ⇒ Aplicar sombra a los objetos.

  ⇒ Simular un tipo de transparencia.

  ⇒ Cambiar las propiedades de iluminación, escala, traslación y color de fondo de la escena.

  ⇒ Cambiar las propiedades de color, tamaño y posición de los objetos.

  ⇒ Navegar dentro de la escena que se ha construido.

  ⇒ Utilizar texturas para los objetos mediante la aplicación de imágenes de mapa de bits (BMP).

  ⇒ Crear simulación de navegación en espacios, como cuartos del tipo Doom por medio de archivos específicos para este fin.

- Desarrollar la aplicación utilizando el lenguaje de programación Visual C++ 6.0 compatible con Win32 y OpenGL para los gráficos tridimensionales.

- Contar con instrucciones propias para el dibujo de objetos, obtenidas de un archivo de texto, que son tomadas y traducidas por la aplicación, y se diferencian de las instrucciones de C++ y OpenGL, ya que son en español y sirven para guardar los gráficos que se han creado.

- Facilitar la comprensión de las funciones utilizadas para la representación de objetos. Debido a que la aplicación desarrollada es de tipo didáctica, su uso permitirá entender lo que se está haciendo; por ejemplo: al dibujar una esfera y rotarla, se detallará cual es la matriz de rotación y los cálculos necesarios para esto.

1.5.2. LIMITACIONES

Las limitaciones que presenta el prototipo son las siguientes:

- Dibuja únicamente los diez objetos que se definen en los alcances, así como su manipulación por medio de las propiedades previamente definidas.

- Sólo se puede cargar un archivo a la vez, esto implica cerrar la escena actual, para cargar una nueva.

- No se pueden crear escenas dentro del prototipo, solamente mediante los archivos de texto.

- No se detectan colisiones de objetos en la escena que se está representando.

- No se cuenta con animaciones, los objetos son estáticos.

- El tiempo de respuesta por parte de la aplicación depende de los recursos de Hardware.

- La aplicación ha sido desarrollada para la plataforma de Windows de manera que sólo es compatible con Win32.

CAPITULO II

MARCO TEORICO

En el mundo real, existen tres coordenadas para la representación de un lugar en el espacio, se tiene entonces un espacio tridimensional, donde el objeto básico es el sólido. El modelado es la construcción artificial de un objeto para facilitar su estudio; los modelos consisten en información almacenada en archivos que permiten visualizar los objetos y son usados para representar formas que se utilizan en una escena. Las formas simples pueden definirse a partir de expresiones analíticas; las formas más complejas, requieren de métodos más complejos.

2.1. OpenGL

Definido como una interfaz de software para hardware de gráficos; consiste de una librería de modelado y gráficos en tercera dimensión, rápida y portable. Como su definición lo indica, es una librería y no un lenguaje de programación como C/C++ o Visual Basic. Es utilizada por los lenguajes de programación por medio de instrucciones para acceder a esa librería; por ejemplo, en C/C++ se utilizan las librerías gl.h, glu.h, glaux.h, las cuales se incluyen en el directorio de las librerías.

No ofrece funciones para la creación de ventanas o interacción con el usuario; debido a esto, se recurre a las funciones proporcionadas por el sistema operativo. Esta es la razón por la que OpenGL fue creado pensando en la independencia de la plataforma, se puede utilizar en Windows o UNÍX indiferentemente.

2.2. VISUALIZACION TRIDIMENSIONAL

Los ojos humanos ven imágenes que el cerebro combina para dar una percepción tridimensional; aún cuando no se piensa acerca de ello; el entorno visual en su totalidad está dominado por imágenes tridimensionales (Ver figura 2.1). La visualización tridimensional tiene por objetivo representar objetos y escenas en la computadora; permite crear imágenes, animaciones y escenas interactivas tan realistas o fantásticas como se quiera. La representación de objetos en tres dimensiones se ha ido introduciendo en la vida cotidiana quizá sin notarse, desde las ilustraciones de edificios aún no construidos, pasando por los logos de canales de televisión a la hora de las noticias, hasta películas de la calidad visual.

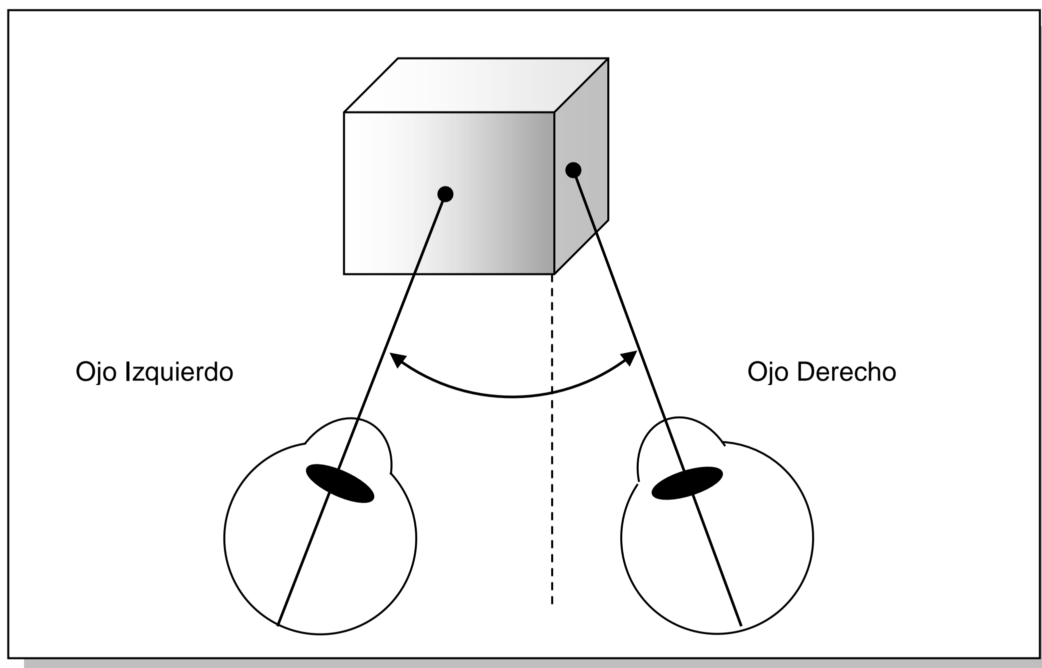


Figura 2.1 Como los ojos ven en tres dimensiones

2.3. LUZ Y SOMBRA

La luz tiene dos efectos fundamentales en los objetos vistos en tres dimensiones. Primero, provoca que una superficie de un color uniforme aparezca sombreada cuando es vista o iluminada de un ángulo. Segundo, los objetos que no transmiten luz (la mayoría de objetos sólidos) provoca una sombra cuando obstruyen la ruta de los rayos de luz (ver figura 2.2).

Dos orígenes de iluminación pueden influenciar los objetos tridimensionales. La *luz ambiental,* la cual no tiene dirección, puede causar efecto de sombreado sobre los objetos de color sólido. Otro tipo de luz, la que tiene un origen, llamada *lámpara,* puede ser usada para cambiar el sombreado de los objetos sólidos y para los efectos de sombra.

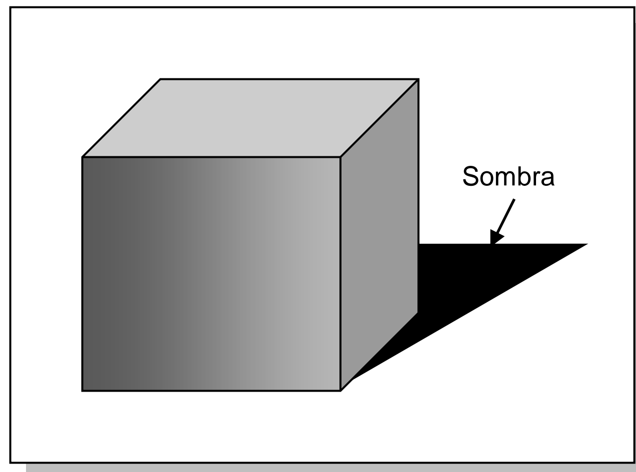


Figura 2.2 Objeto sólido iluminado por una luz

2.3.1. ILUMINACION AMBIENTAL

No es una luz que viene de una dirección en particular. Tiene su origen, pero los rayos de luz tienden a rebotar en la escena y se convierten en rayos sin dirección. Ver figura 2.3.

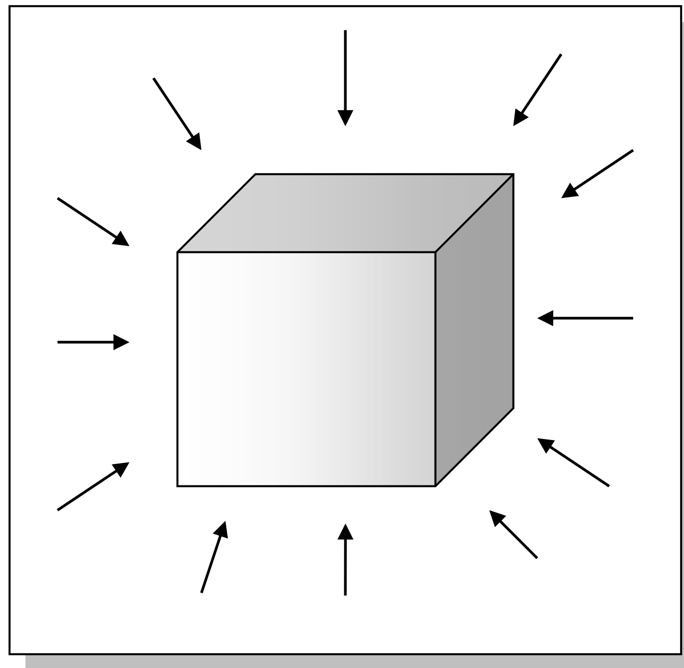


Figura 2.3 Objeto iluminado con luz ambiental

Efectos de la luz ambiental

Debido a la forma que la computadora reconoce valores para los pixeles debe dejarse de lado la noción de colores y pensar solamente en términos de intensidades rojo, verde y azul (Red, Green y Blue en inglés con sus iniciales RGB). Un origen de luz ambiental de media intensidad tendría los valores RGB de (0.5, 0.5, 0.5). Si esta luz ilumina un objeto con

propiedades reflectivas, específicamente en términos de RGB de (0.5, 1.0, 0.5), entonces el componente del color neto de esa luz sería:

(0.5 * 0.5 , 0.5 * 1.0 , 0.5 * 0.5) = (0.25 , 0.5 , 0.25)

Que es el resultado de multiplicar cada uno de los términos de la luz ambiental por cada uno de los términos de los materiales ambientales; esto se puede apreciar en la siguiente figura:

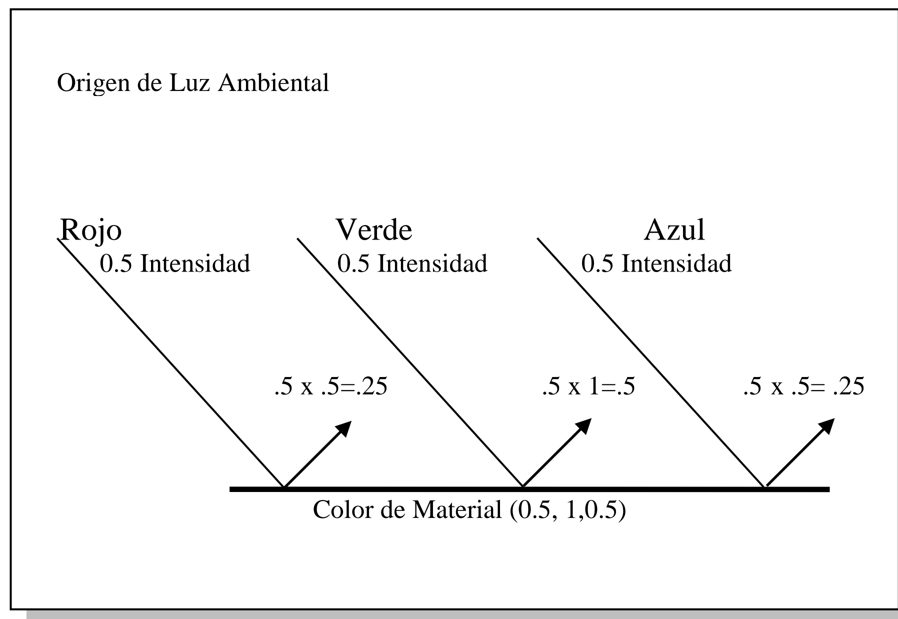


Figura 2.4 Calculando el componente de color ambiental de un objeto

2.3.2. ILUMINACION DIFUSA

La luz difusa viene de una dirección en particular pero es reflejada uniformemente sobre una superficie, la cual brilla más si la luz apunta directamente a la superficie, que si la luz la rozara con un ángulo

determinado, como se puede observar en la figura 2.5. Un buen ejemplo de esto es la luz fluorescente.

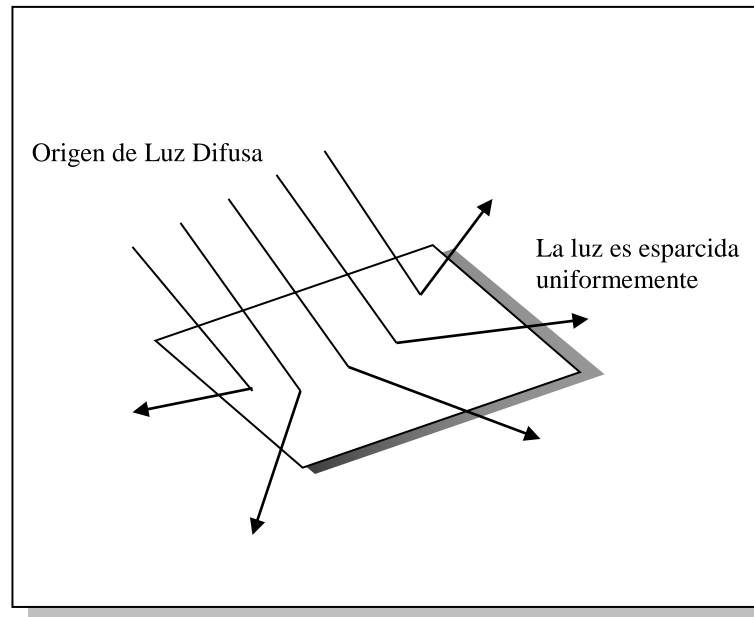


Figura 2.5 Objeto iluminado con luz difusa

2.3.3. ILUMINACION ESPECULAR

Como la luz difusa, la luz especular es direccional, pero es reflejada claramente y en un punto en particular. Una muy alta iluminación especular tiende a causar manchas brillantes sobre la superficie. Un faro giratorio y el sol son ejemplos de luz especular. En la figura 2.6 se muestra como un objeto es iluminado con luz especular.

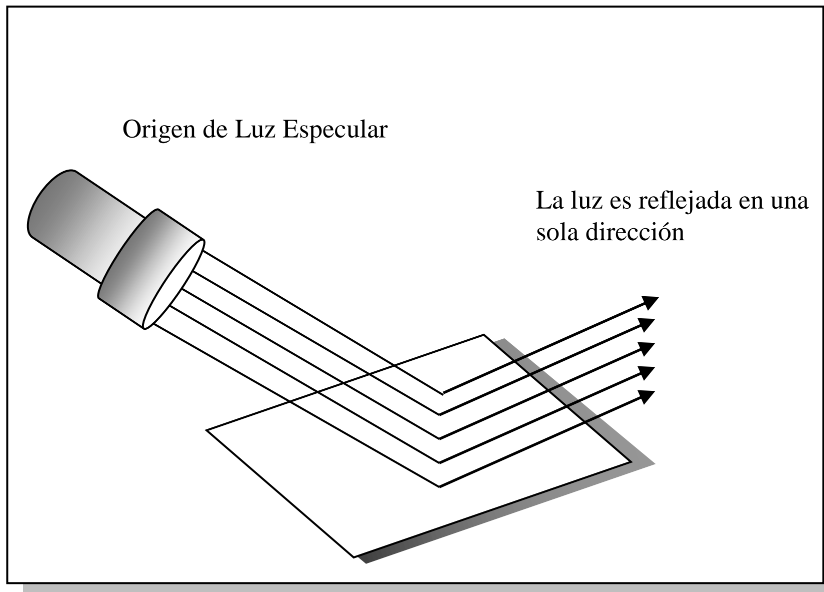


Figura 2.6 Objeto iluminado con luz Especular

2.4. TRANSFORMACION DE COORDENADAS

Las transformaciones hacen posible la proyección de coordenadas 3D a una pantalla 2D. Las transformaciones permiten también rotar, mover y aun agrandar o encoger los objetos de la escena. En lugar de modificar un objeto directamente, una transformación modifica el sistema de coordenadas. Una vez una transformación rota el sistema de coordenadas, los objetos aparecerán rotados al momento de dibujarse. Hay tres tipos de transformación que ocurren entre el tiempo cuando se especifican los vértices y el tiempo en el que aparecen en pantalla: Vistas, Modelaje y Proyección.

2.4.1. TRANFORMACIONES DE VISTA

Las transformaciones de vista presentan las siguientes características:

- Son las primeras en ser aplicadas en la escena. Usadas para determinar el punto de anclaje de la escena. Por defecto, el punto de observación está en el origen (0,0,0). Este punto de observación es movido relativamente al eje de sistema coordenado para proporcionar un punto de anclaje específico. Cuando el punto de observación está ubicado en el origen, los objetos que se dibujen con valores de z positivo estarán detrás del observador.
- Permiten poner el punto de observación donde sea. Determinar la transformación de vista es como poner y apuntar una cámara en la escena.
- Deben ser especificadas antes que otra transformación, porque mueven el actual sistema de coordenadas de trabajo con respecto al sistema de eje de coordenadas. Todas las subsecuentes transformaciones pueden entonces ocurrir basadas en el nuevo sistema de coordenadas modificado.

2.4.2. TRANSFORMACIONES DE MODELAJE

Las transformaciones de modelaje son usadas para manipular el modelo y en particular los objetos dentro de él. Esta transformación mueve objetos a

un lugar, los rota y les da escala. La figura 2.7 ilustra tres transformaciones de modelaje que se pueden aplicar a los objetos.

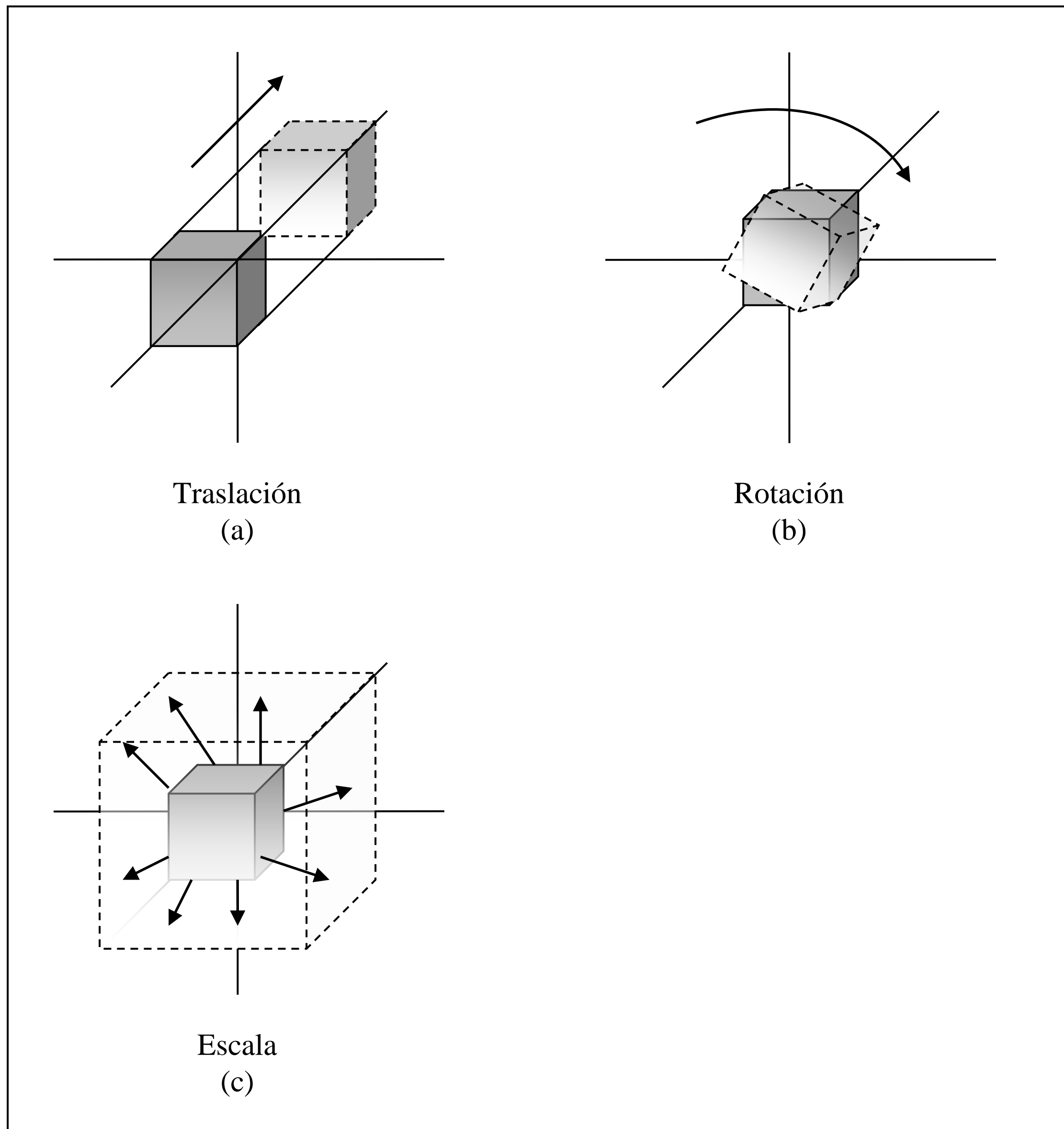


Figura 2.7 Transformaciones de modelaje

La figura 2.7a muestra la *traslación,* donde los objetos son movidos a lo largo de un eje dado. La figura 2.7b muestra la *rotación,* donde el objeto es rotado alrededor de uno de los ejes. Finalmente, en la figura 2.7c se muestra el efecto de la *escala*, donde las dimensiones del objeto son incrementadas o decrementadas en una cantidad especifica. La escala puede ocurrir no uniformemente, y esto puede causar la deformación de los objetos.

2.4.3. TRANSFORMACIONES DE PROYECCION

Las transformaciones de proyección son aplicadas a la orientación final del modelo de vista. Esta proyección define el volumen de vista y establece planos de recorte. Para ser más precisos, las transformaciones de proyección especifican cuando una escena finalizada (después que todo el modelaje es hecho) es trasladada a la imagen final en la pantalla.

En una proyección ortográfica, todos los polígonos son dibujados en la pantalla exactamente con las dimensiones relativas especificadas. Esto es típicamente usado en CAD, o imágenes heliográficas donde se trabaja con dimensiones precisas y exactas.

Una proyección de perspectiva muestra objetos y escenas más como son en la vida real que en planos heliográficos. El objetivo de las proyecciones de perspectiva es el acortamiento, lo cual hace que los objetos más distantes parezcan más pequeños que los que están más cerca, aunque sean de la misma medida. Las líneas paralelas no siempre se dibujarán paralelas. En una línea del tren, por ejemplo, los rieles son paralelos, pero

con la proyección perspectiva parecen converger en algún punto en la distancia. A esto se la llama p*unto de desvanecimiento.*

El beneficio de las proyecciones de perspectiva es que no es necesario imaginarse donde las líneas convergen, o cuanto más pequeños son los objetos distantes. Todo lo necesario es especificar que la escena use las Transformaciones del modelo de vista (Modelview), y luego aplicar la proyección de perspectiva.

La figura 2.8 compara las proyecciones ortográficas y perspectiva en dos diferentes escenas.

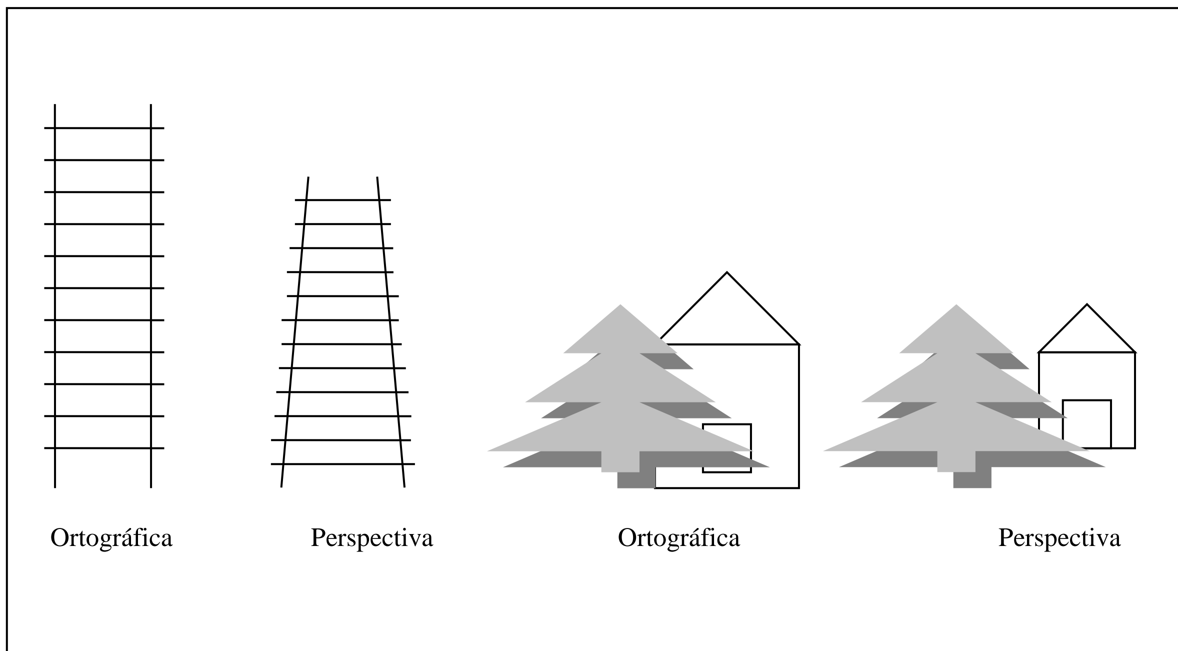


Figura 2.8 Dos ejemplos de proyección ortográfica y perspectiva

CAPITULO III

DISEÑO DEL PROTOTIPO

3.1. ESTRUCTURA DEL PROTOTIPO

La aplicación para la representación de objetos en tres dimensiones tiene una estructura básica, como se muestra en la figura 3.1, en la cual se pueden observar las diferentes etapas que se llevan a cabo para dibujar objetos a partir de archivos de texto.

La primera de estas es el programa principal, encargado de administrar las ventanas y los demás subprogramas necesarios para la representación de los objetos.

La segunda etapa consiste en la traducción del archivo de texto, analizando cada una de sus instrucciones y parámetros, verificando que se encuentre libre de errores para proceder a la etapa de dibujo.

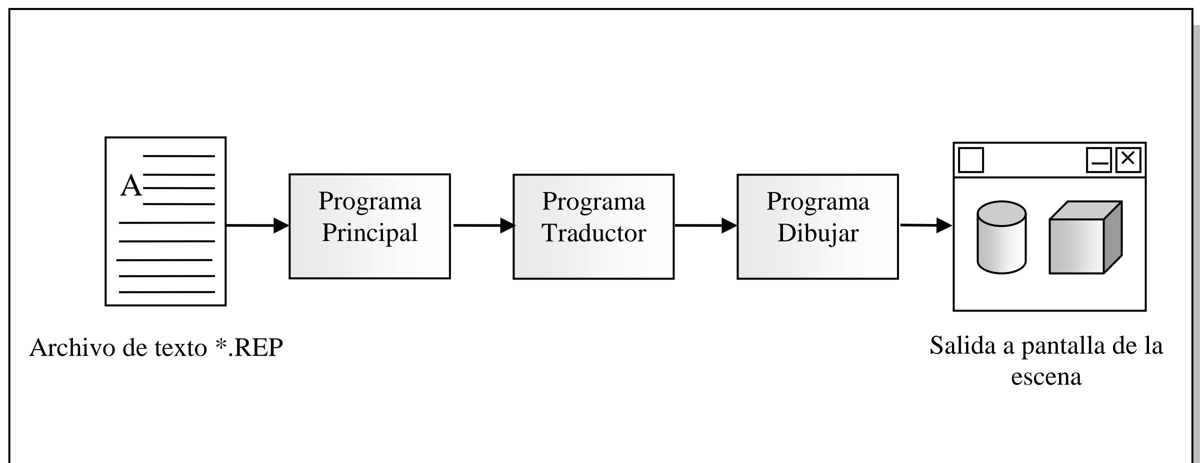


Figura 3.1 Proceso para dibujar una escena

Por ultimo, se realiza el dibujo, y aquí es donde el resultado que se obtuvo de la traducción del archivo de texto se refleja en la escena dibujada en pantalla, dando la opción de poder cambiar aquí las propiedades de los objetos así como el entorno.

- **Programa Principal**

Como controlador de la aplicación, se encarga de manejar los mensajes de Windows para la creación de la ventana principal, así como de verificar las teclas y las opciones de menú solicitadas por el usuario.

Es aquí donde se inicia el proceso de la aplicación, por medio de la opción *abrir* del menú *archivo,* se obtiene el nombre del archivo que contiene la información para la escena que se desea representar; aquí se verifica la extensión del archivo para poder llamar a la función correcta para su traducción. Dependiendo de esto, si un archivo es cargado, se habilitan las opciones del menú que tienen estado inicial deshabilitado.

El programa da como salida el nombre del archivo, y este pasa a la siguiente etapa que es la traducción. Además, envía el controlador de la ventana para su utilización en las siguientes funciones.

- **Programa Traductor**

Con la cadena obtenida del programa principal, inicia el proceso de traducción del archivo, que conlleva, con este, la necesidad de implementar rutinas de verificación de sintaxis así como la creación de un objeto que será utilizado para

vaciar los resultados obtenidos de la traducción y que también servirá para la siguiente etapa, la de dibujo.

Este objeto será una estructura de datos donde se guardarán los atributos que sean extraídos del archivo de texto; al final, esta estructura será el parámetro para el programa de dibujo.

La estructura debe permitir manipular tanto la información de los objetos, como la información de la escena. Las propiedades de cada uno de los objeto variará dependiendo su tipo; por ejemplo, las propiedades para dibujar una esfera no serán las mismas que las utilizadas para dibujar un cubo.

- **Programa dibujar**

Esta es la parte de la aplicación encargada de crear una ventana nueva para dibujar una escena con la información recibida del programa traductor; esta ventana tendrá como padre la ventana principal.

Por medio de la estructura recibida, comenzará el proceso de dibujo con las funciones que se crearán para este fin. Este programa será capaz de modificar las propiedades contenidas en el archivo de texto, y los cambios se producirán en el mismo momento sin necesidad de guardar y abrir nuevamente el archivo.

Dos programas de dibujo serán creados, uno será el dibujador de objetos tridimensionales, los cuales han sido mencionados en las secciones anteriores; el otro, será el encargado de dibujar las simulaciones de cuartos de tipo DOOM. El programa que se ejecute dependerá del tipo de archivo abierto en la aplicación.

- **Programa propiedades**

Las propiedades que poseen los objetos y la escena en general podrán ser modificadas dentro de la aplicación por medio del programa “Propiedades”, este se encargará de aplicar los cambios a las estructuras de datos actuales y redibujar la escena instantáneamente con los nuevos cambios.

La aplicación dará la opción de guardar los cambios que se hayan efectuado a los objetos con el nombre del archivo que se abrió originalmente. Esta opción no estará disponible para las simulaciones de cuartos del tipo DOOM.

3.2. ARBOL DE OPCIONES

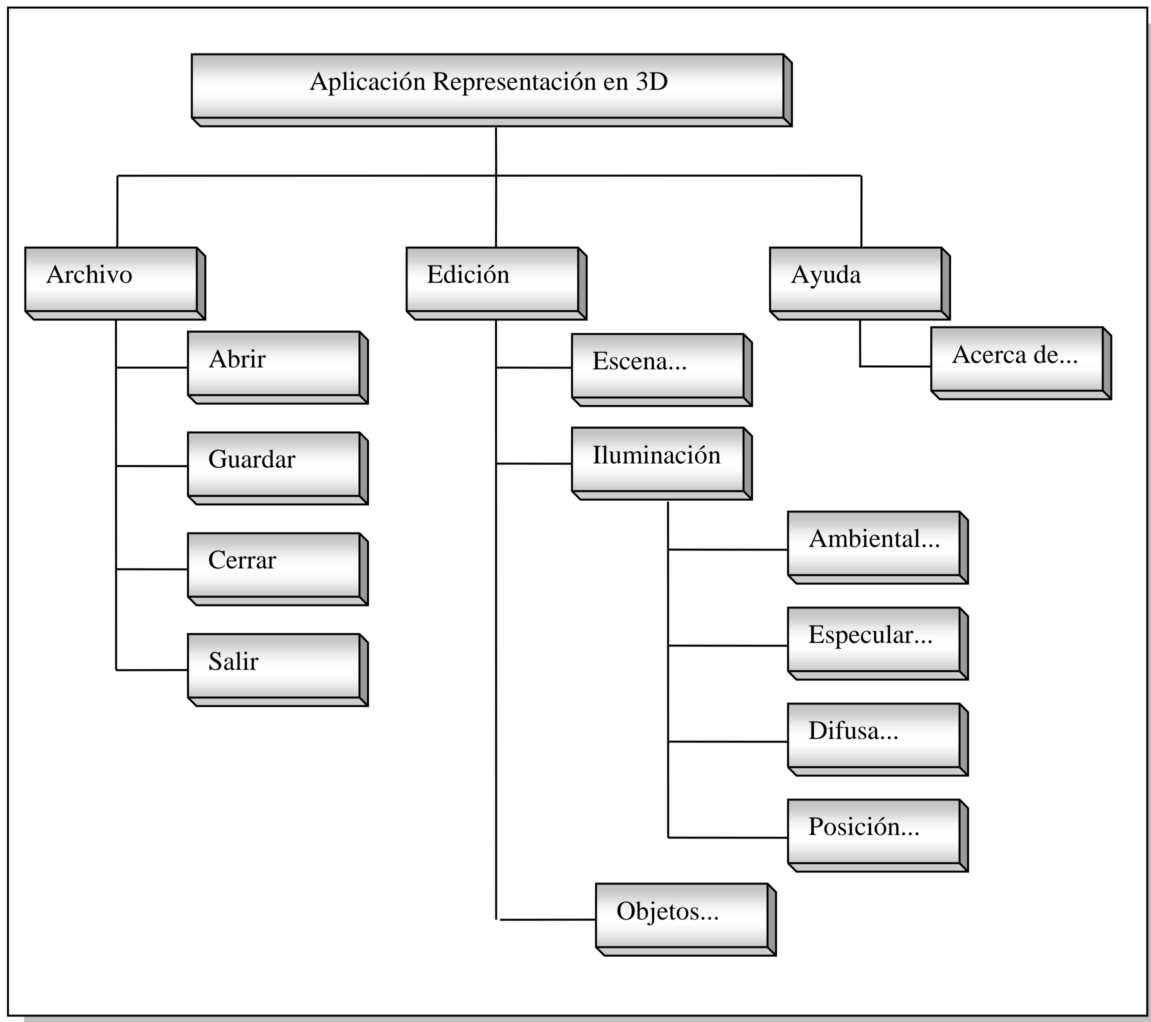


Figura 3.2 Árbol de opciones de la aplicación

El diagrama anterior presenta las opciones de menú con las que contara la aplicación, como parte de su interfaz gráfica al usuario. La interfaz será de tipo sencilla, pues el prototipo contará sólo con las opciones de cargar archivos y editar sus propiedades.

A continuación se define la función de cada una de las opciones.

- **Archivo**

El menú archivo tendrá opciones de manejo de archivos; sus opciones son:

- **Abrir:**

  La opción abrir, mostrará la ventana de dialogo Abrir de Windows, que nos permitirá abrir dos tipos de archivos, esto son:

  - Archivos de tipo Representación de Objetos 3D cuya extensión es *.REP.
  - Archivos de tipo Simulación de Cuartos tipo DOOM con extensión *.SIM.

  Estos archivos son de tipo texto y son los únicos que podrán ser cargados por la aplicación.

- **Guardar:**

  Por medio de esta opción se podrá almacenar los cambios que se efectúen a los objetos o a la escena que se esta representando. Su estado inicial será inactivo, cambiará de estado cuando se cargue un archivo y se le hagan modificaciones, al cerrar la escena vuelve a su estado de inactivo.

- **Cerrar:**

  La escena que esté cargada en la aplicación podrá ser cerrada para poder abrir otro archivo, ya que únicamente se permite abrir un archivo a la vez. El

estado inicial de esta opción es inactivo, cambia de estado cuando se abre un archivo y es dibujada la escena.

- **Salir:**

  Cierra la escena con la que se esta trabajando actualmente y cierra toda la aplicación, su estado es siempre activo.

- **Edición**

Por medio del menú edición se podrá realizar cambios a la escena u objetos que se dibujen en pantalla, esto por medio de las siguientes opciones:

- **Escena:**

  Mediante esta opción se podrá cambiar las propiedades de la escena que se dibuje, esta nos llevará a un cuadro de dialogo que permitirá hacer modificaciones a los siguientes parámetros:

  - Color de Fondo
  - Traslación de la escena
  - Escala con la que se dibuja la escena

- **Iluminación**

  La iluminación de la escena es parte fundamental para una buena visualización de los objetos que en ella se están representando.

Las siguientes sub-opciones permitirán cambiar las propiedades de ésta.

- **Ambiental**

  Abrirá un cuadro de dialogo que permitirá cambiar las propiedades de la iluminación ambiental, estos cambios se verán reflejados al momento de presionar el botón aceptar del cuadro de dialogo.

- **Difusa**

  Similar a la opción anterior, con la diferencia que este presentará los valores para la iluminación difusa.

- **Especular**

  Esta opción permitirá cambiar los valores para la iluminación especular.

- **Posición**

  La posición en la que se encuentra la luz es algo muy importante al dibujar una escena, pues de esto depende mucho la correcta visualización de los objetos. Estos valores podrán ser modificados mediante el cuadro de dialogo Posición de luz.

➢ **Objetos**

Los objetos que se dibujen en la escena podrán ser modificados mediante esta opción, las propiedades que tenga el objeto podrán ser cambiadas y

estos se verán reflejados en la escena al momento de presionar el botón aceptar.

- **Ayuda**

Como un apoyo al usuario se presentará una ventana con información acerca de la aplicación y su modo de operación, así como las instrucciones que puede utilizar para la creación de archivos que la aplicación reconoce para la representación de objetos y escenas.

3.3. DISEÑO DE LA INTERFAZ DE LA APLICACION

La aplicación tendrá una interfaz sencilla que será fácil de usar. Valiéndose de los recursos que proporciona Windows para la creación de ventanas y cuadros de dialogo, se realizará una aplicación que sea fácil de acceder a todas sus opciones, mediante teclas de acceso rápido o por menús desplegables dentro de la aplicación.

La pantalla principal será de la siguiente forma:

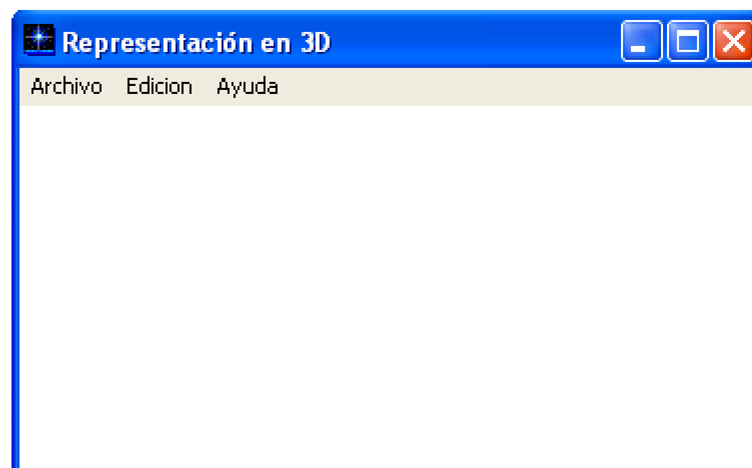


Figura 3.3 Pantalla principal de la aplicación

Esta tendrá los botones de maximizar y minimizar disponibles, además contara con las siguientes teclas de acceso rápido (*hot keys*):

1. **Ctrl + A:** Abrir archivo.
2. **Ctrl + G:** Guardar Archivo.
3. **Ctrl + S:** Propiedades de Escena.
4. **Ctrl + M:** Propiedades de Luz Ambiental.
5. **Ctrl + D:** Propiedades de Luz Difusa.
6. **Ctrl + E:** Propiedades de Luz Especular.
7. **Ctrl + P:** Propiedades de Posición de la Luz.
8. **Ctrl + O:** Propiedades de los Objetos en la escena.

La ventana en la que se dibuje la escena será una nueva dentro de la aplicación, y será hija de la ventana principal, tendrá los botones minimizar y maximizar habilitados; además, contará con un menú desplegable dentro de la escena para poder acceder a las propiedades. El título de esta ventana será la ruta donde se encuentra el archivo que esté en escena.

En la figura 3.4 se puede apreciar la ventana dentro de la aplicación, así como el menú desplegable que estará disponible al dar clic derecho dentro de ella.

En la figura 3.5 se presenta un ejemplo de un cuadro de dialogo que modifica las propiedades de la escena. Los demás cuadros de dialogo poseerán el mismo estándar de éste y servirán para cambiar las propiedades de iluminación y de los objetos; solamente estará disponible el botón de cerrar.

Figura 3.4 Pantalla de dibujo con el menú conceptual

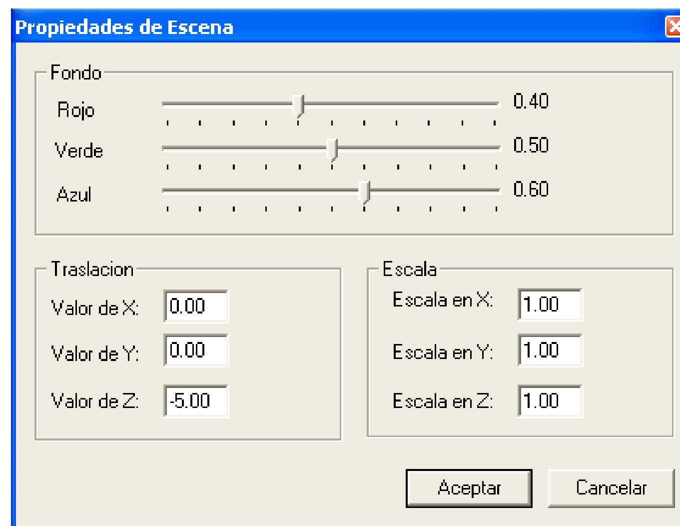


Figura 3.5 Cuadro de dialogo de propiedades

CAPITULO IV

DESARROLLO DEL PROTOTIPO

4.1. DIAGRAMA DE FUNCIONAMIETO DE LA APLICACIÓN

Por medio las siguientes figuras se hace un bosquejo de las principales funciones con las que cuenta cada uno de los programas que conforman la aplicación.

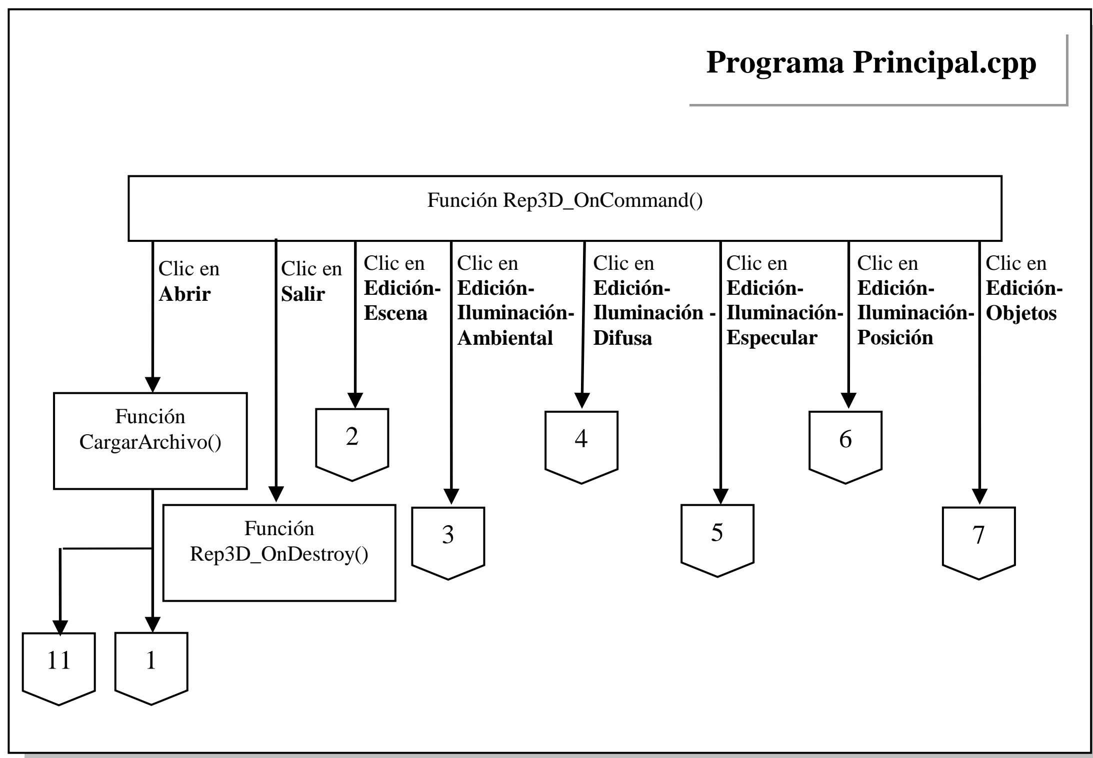


Figura 4.1 Diagrama de funcionamiento programa Principal.cpp

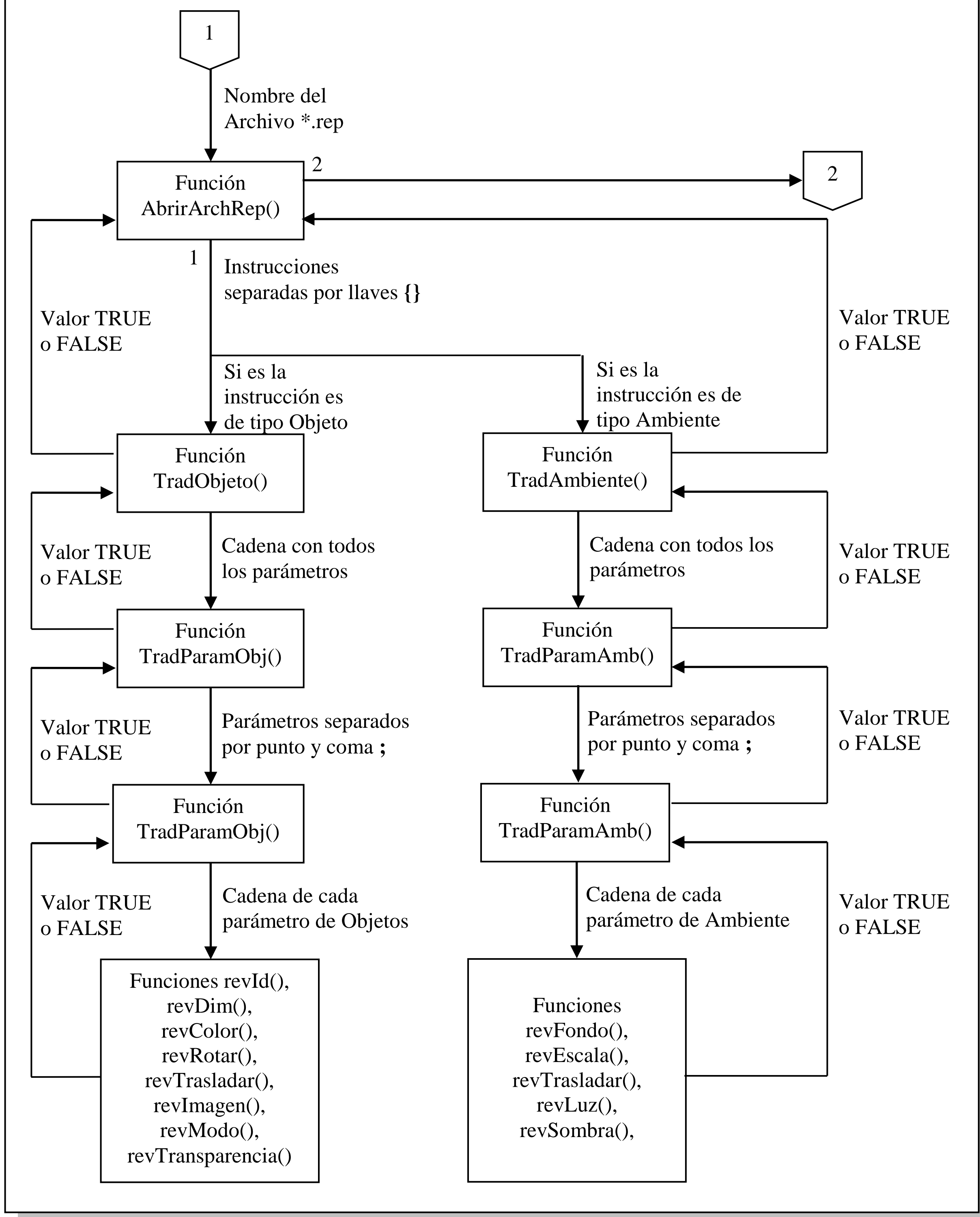


Figura 4.2 Diagrama de funcionamiento programa Traductor.cpp

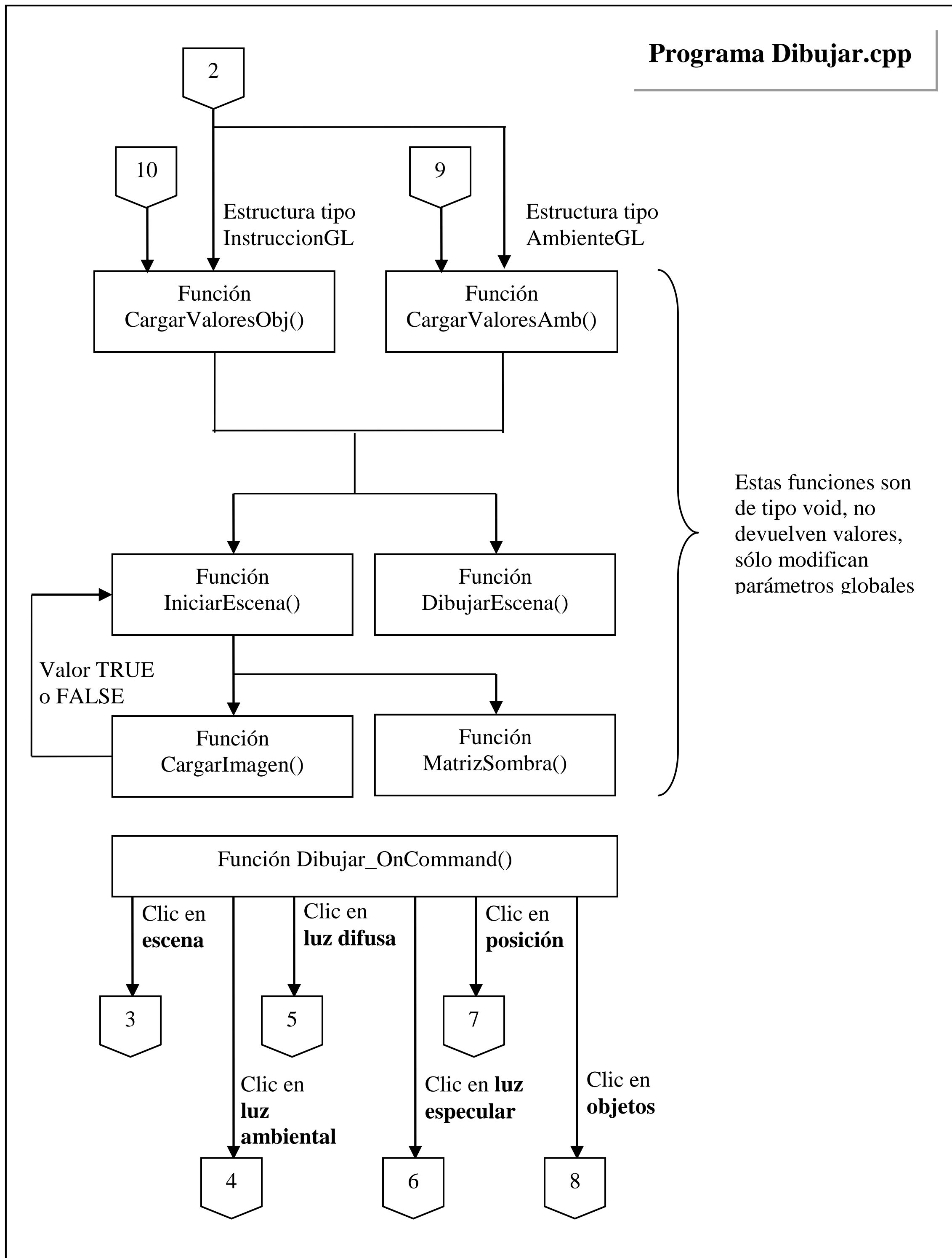


Figura 4.3 Diagrama de funcionamiento programa Dibujar.cpp

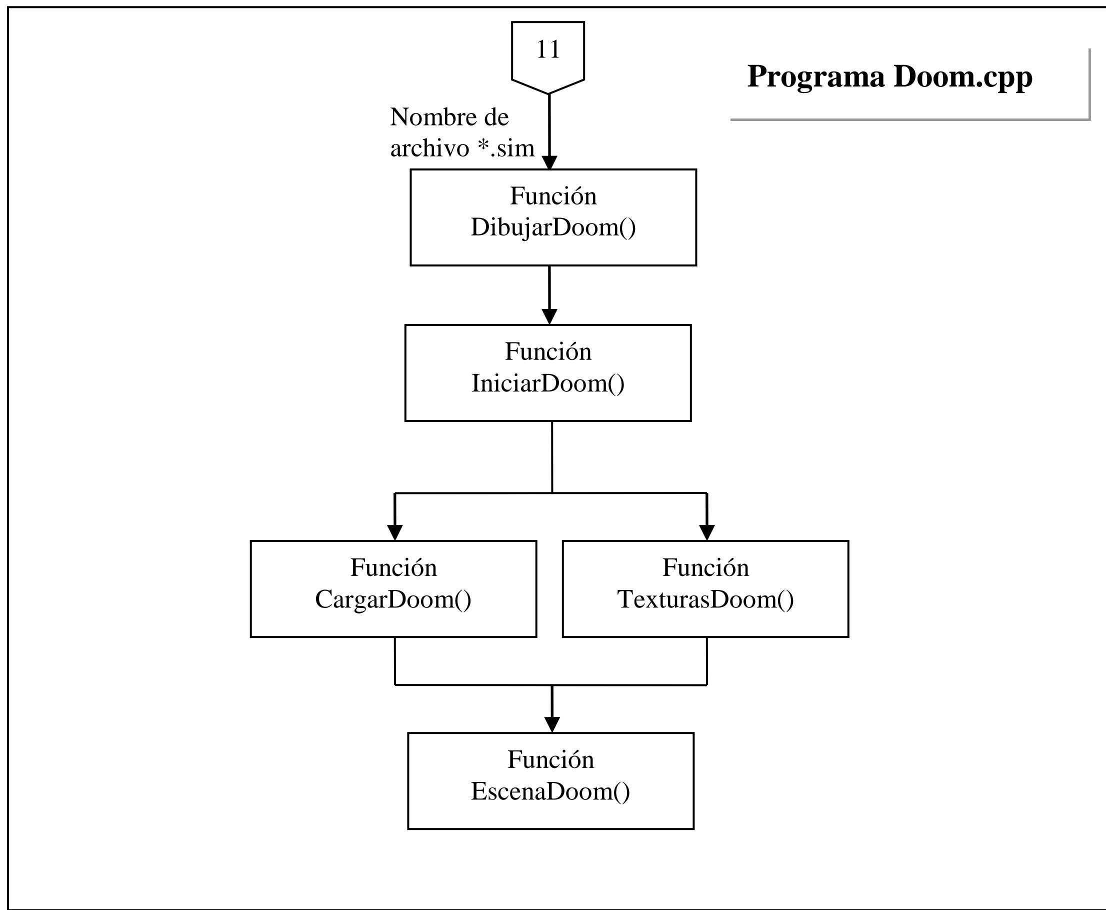


Figura 4.4 Diagrama de funcionamiento programa Doom.cpp

4.2. PROGRAMA PRINCIPAL

Flujograma de inicialización

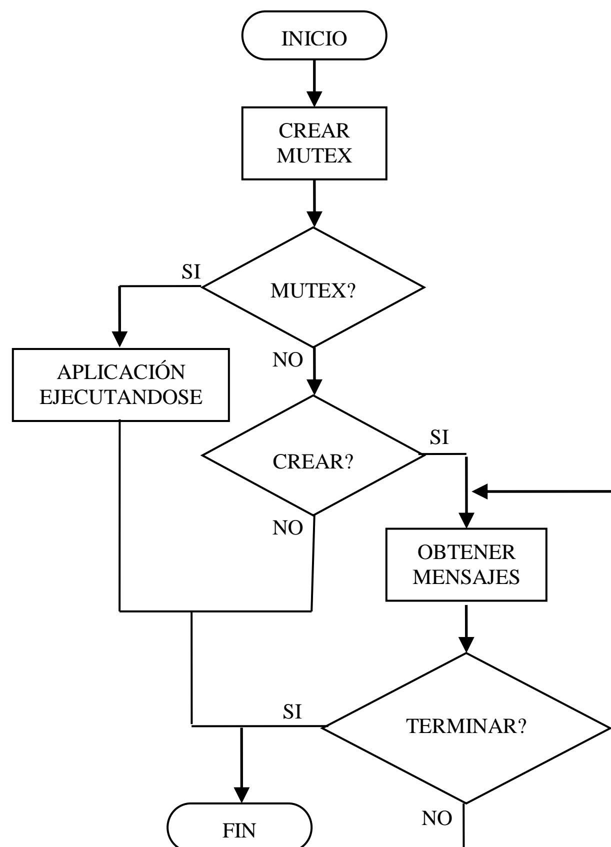


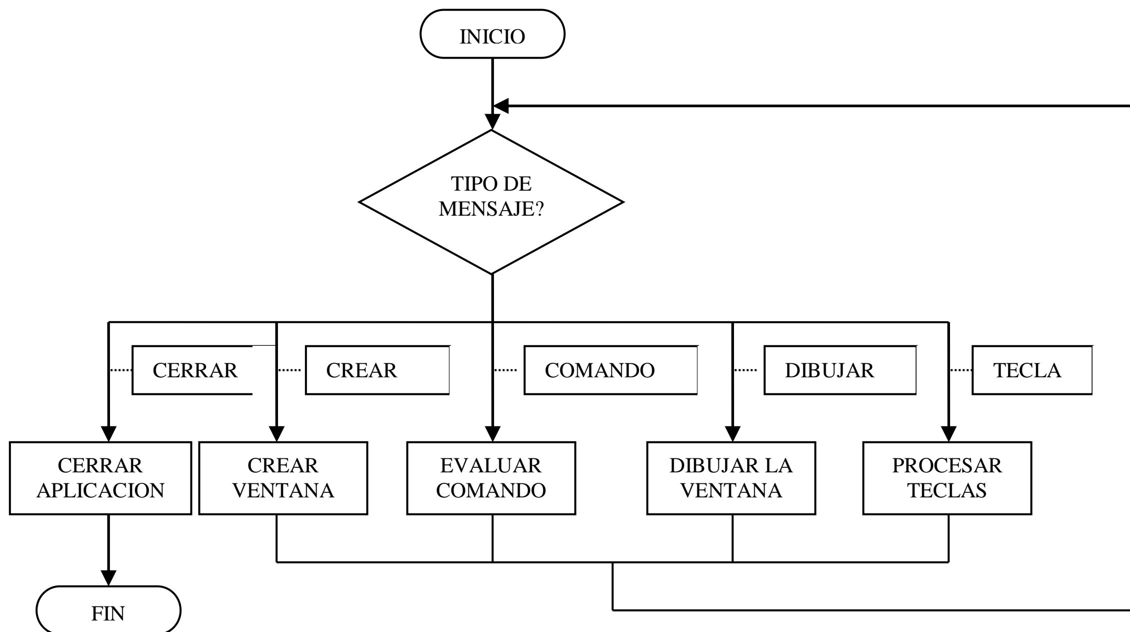
Figura 4.5 Flujograma inicializar

La primera función cargada por la aplicación es la que crea la ventana principal, esta realiza los siguientes pasos:

1. Crear mutex, abreviatura de mutuamente excluyente, es el encargado registrar la instancia de la aplicación, esto debido a que sólo se puede ejecutar una y sólo una instancia de esta a la vez.
2. Verificar si el mutex ya ha sido creado. Si existe una tarea con el nombre que se le ha dado a la aplicación en la barra de tareas de Windows es porque ya se está ejecutando una instancia de la aplicación, en dicho caso se envía un mensaje de error y finaliza la ejecución.
3. Verificar si la ventana ha sido creada, por medio de esta se crea la ventana y se registra la clase de la aplicación, si ocurre un error finaliza la ejecución.
4. Obtener mensajes, por medio de este proceso se leen los mensajes que entran a la aplicación, ya sea por medio del teclado o del mouse en el menú de la aplicación.
5. Si el mensaje obtenido es el de cerrar la aplicación, finaliza la ejecución de esta, en otro caso se continúa la lectura de mensajes.

Flujograma de manejo de mensajes de la aplicación

Figura 4.6 Flujograma de manejo de mensajes



La función que maneja los mensajes de la aplicación es llamada WndProc(), esta se encarga de llamara a la función adecuada dependiendo de la solicitud.

1. Lee el tipo de mensaje, según el tipo de este llama a la función que atenderá la solicitud.
2. Cerrar aplicación, con el mensaje WM_DESTROY se cierra la aplicación y la ventana que este creada dentro de ella.
3. Crear ventana, el tipo de mensaje para la creación de la ventana es WM_CREATE, este llama a la función Rep3D_OnCreate() que es la encargada de deshabitar las opciones de menú que no se utilizarán hasta que se utilicen en las escenas.
4. Comando, puede ser del teclado por medio de las teclas rápidas o por medio del mouse para acceder a las opciones del menú, el mensaje leído es WM_COMMAND, y este llama a la función Rep3D_OnCommand(), donde son procesadas las opciones seleccionadas.

5. Dibujar, por medio del mensaje WM_PAINT se llama a la función Rep3D_OnPaint(), que es la encargada de crear el contexto para la ventana que se dibujará.
6. Teclas, este es utilizado dentro de una escena que ya se ha dibujado parra poder navegar dentro de ella, el mensaje leído es WM_KEYDOWN y llama a la función OnKeyDown().

Flujograma para obtener el nombre del archivo

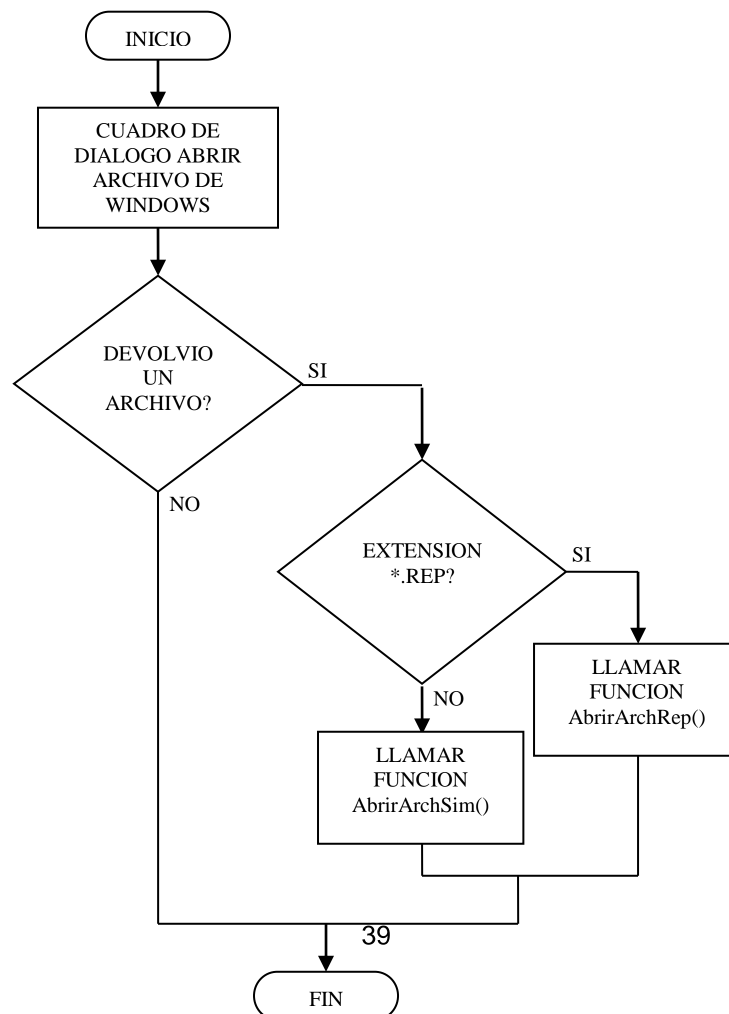


Figura 4.7 Flujograma para obtener nombre de archivo

La función manda el cuadro de dialogo de Windows para abrir archivos (ver figura 4.7), este desplegará únicamente archivos de los siguientes tipos:

- Archivos de representación 3D, *.REP
- Archivos de simulación DOOM, *.SIM

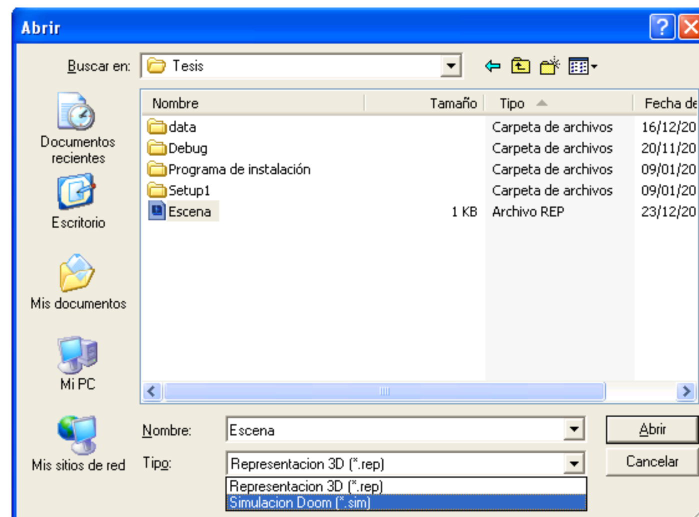


Figura 4.8 Cuadro de dialogo Abrir Archivo de Windows

Los pasos son los siguientes:

1. Muestra el cuadro de dialogo "Abrir".
2. Si presiona aceptar, se verifica la extensión del archivo, caso contrario finaliza la función.

3. Verificación de la extensión del archivo, por medio de esta se podrá determinar la función que evaluará dicho archivo. Si la extensión es de tipo .REP se llama a la función AbrirArchRep(), de no ser así el valor de la extensión es .SIM, para lo cual se llamará a la función AbrirArchSim().

4.3. PROGRAMA TRADUCTOR

Flujograma para inicializar el traductor

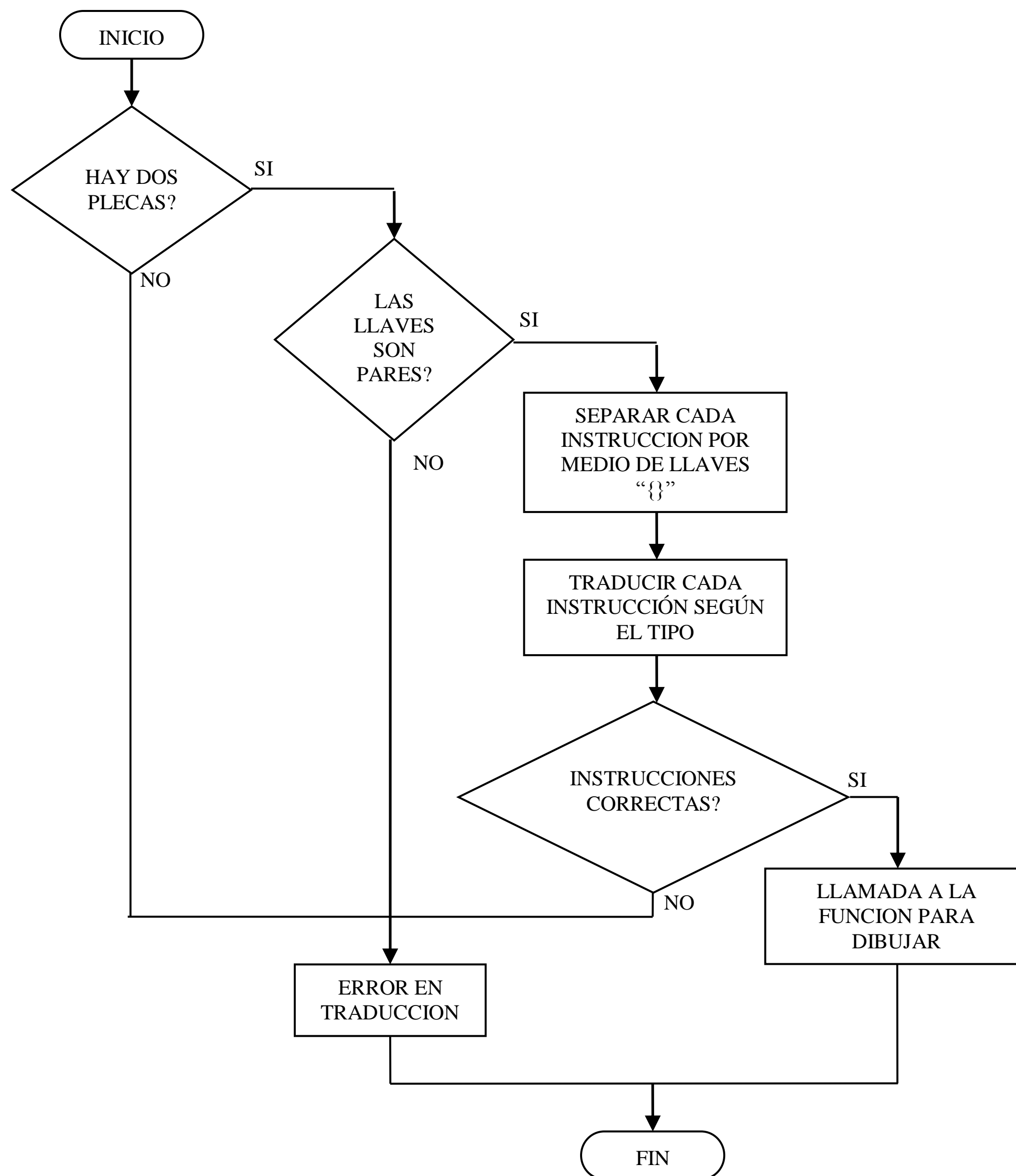


Figura 4.9 Flujograma para inicializar el traductor

El programa traductor inicializa en la función AbrirArchRep(), los pasos se describen a continuación:

1. Verifica si hay dos plecas en el archivo, estas son las que definen el inicio y el fin del archivo que se traducirá, esto por medio del análisis del primer y el último

carácter válido. Si esta condición no se cumple se genera un error de traducción, caso contrario se procede con el análisis del archivo.

2. La siguiente verificación que se lleva a cabo es la de las llaves “{}”, por medio de estas se dividen las instrucciones que contiene el archivo, el análisis es simple: *A cada llave de inicio le corresponde una y sólo una de cierre,* si esta condición se cumple se procede a la división de las instrucciones, estas serán procesadas dependiendo de su tipo.
3. Terminado el proceso de traducción (ver figuras 4.9 y 4.10, flujogramas de traducción de instrucciones) se verifica que todas las instrucciones estén correcta para proceder a llamar a la función dibujar(), caso contrario se produce un error de traducción.

Flujograma de traducción de instrucciones tipo “objeto”

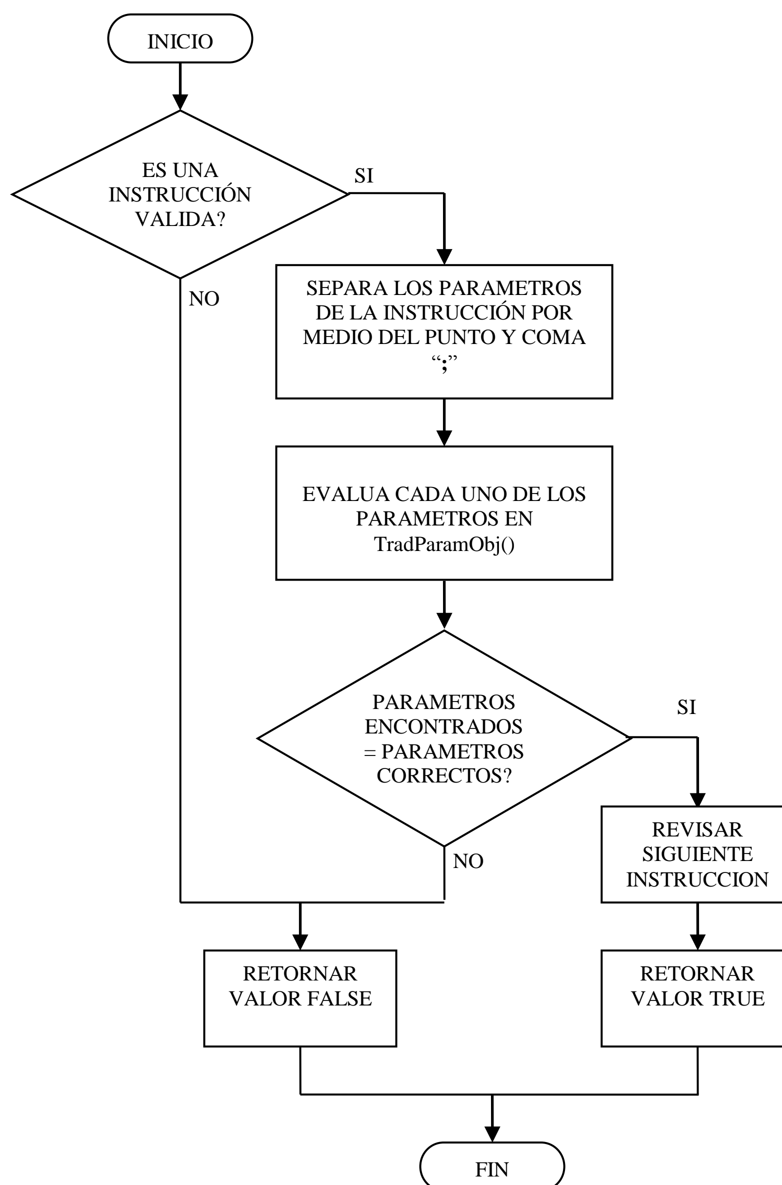


Figura 4.10 Flujograma de traducción de instrucciones tipo “objeto”

Algo que hay tener en cuenta a la hora de la verificación de las instrucciones es el tipo de instrucción que se está tratando, pues existen dos tipos, estos son:

- **Instrucciones de tipo Objeto**: Son las que se utilizan para dibujar los objetos en la escena a crear.

- **Instrucciones de tipo Ambiente**: Utilizadas para crear el ambiente de la escena, por ejemplo color de fondo, luz, traslación de la escena y la escala con la que se trabajará.

Los detalles de los pasos para la traducción de instrucciones tipo objetos son los siguientes:

1. Verificar que la instrucción sea válida, de entre el conjunto de instrucciones que se puede utilizar, se verifica que este bien escrita y que no contenga otros caracteres en la definición de esta.

   Las instrucciones que se pueden utilizar en la aplicación son las siguientes:

   - **esfera { }**
   - **cilindro{ }**
   - **disco{ }**
   - **cubo{ }**
   - **piramide{ }**
   - **linea{ }**
   - **triangulo{ }**
   - **cuadro{ }**
   - **circulo{ }**
   - **punto{ }**

   Si la instrucción es válida se procede al siguiente paso, caso contrario se genera un error de traducción y se retorna un valor FALSE a la función de donde fue llamada.

2. El siguiente paso es la separación de los parámetros de cada una de las instrucciones por medio del separador de parámetros que se ha utilizado, este es el punto y coma, estos son enviados a la función TradParamObj(), que es la encargada de evaluarlos y determinar si son correctos o no (ver figura 4.11, flujograma de traducción de parámetros de objetos).
3. Si todos los parámetros que se encontraron en la instrucción están correctos se procede con la verificación de la siguiente instrucción y se devuelve un valor TRUE a la función de donde fue llamada.

Flujograma de traducción de instrucciones tipo “ambiente”

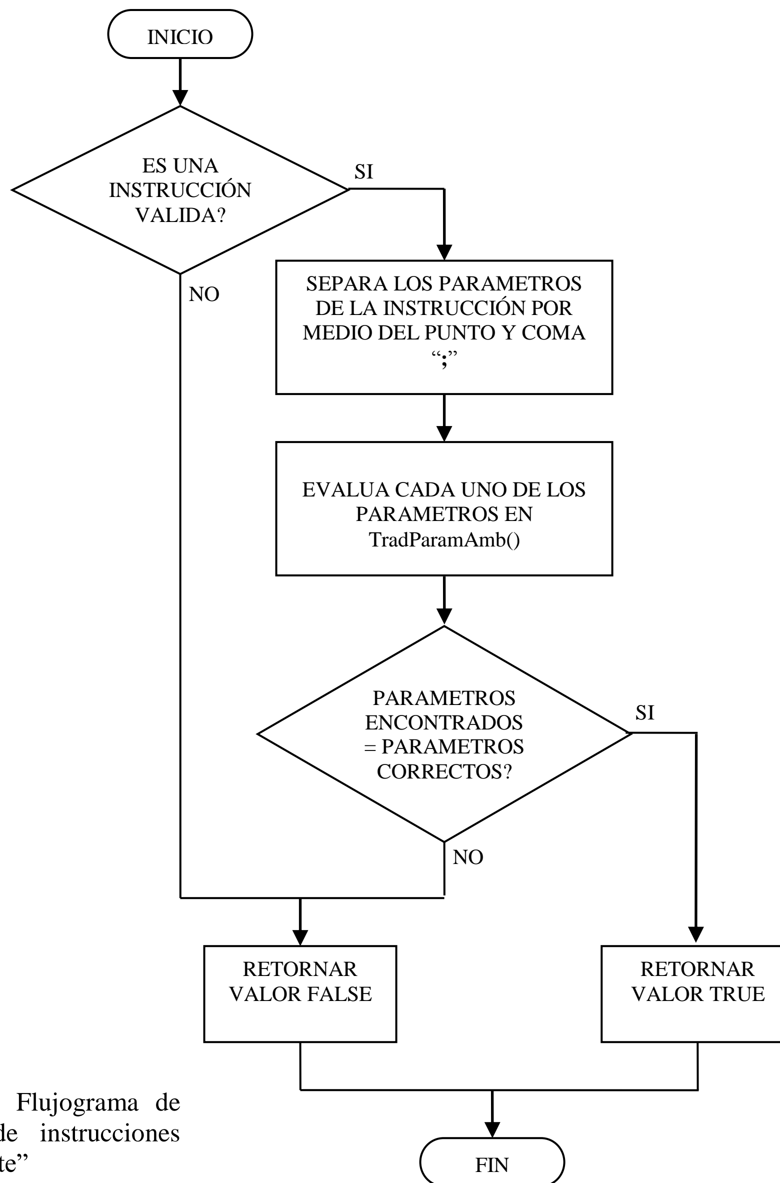


Figura 4.11 Flujograma de traducción de instrucciones tipo "ambiente"

La única instrucción de tipo de ambiente es:

- **ambiente{ }**

Tiene parámetros específicos para su uso, estos son mencionados en la parte donde se explica la traducción de parámetros de tipo ambiente (Figura 4.12, flujograma de traducción de parámetros de ambiente).

Los pasos que se siguen para la verificación de la instrucción son similares a los de la verificación de instrucciones de tipo objeto con la diferencia que al terminar el proceso concluye la función no se esperan más instrucciones para ser evaluadas.

Flujograma para la traducción de parámetros de objetos

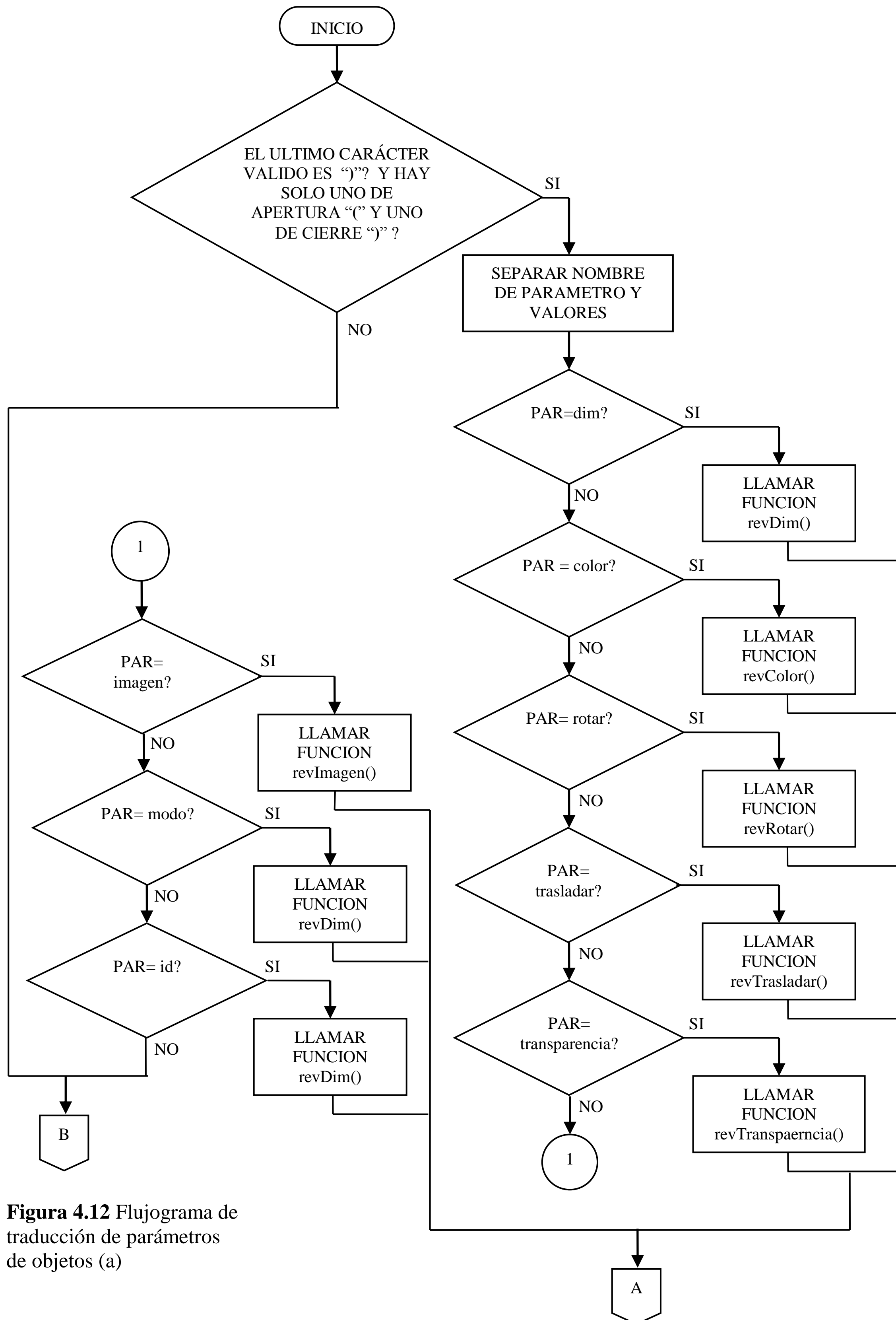


Figura 4.12 Flujograma de traducción de parámetros de objetos (a)

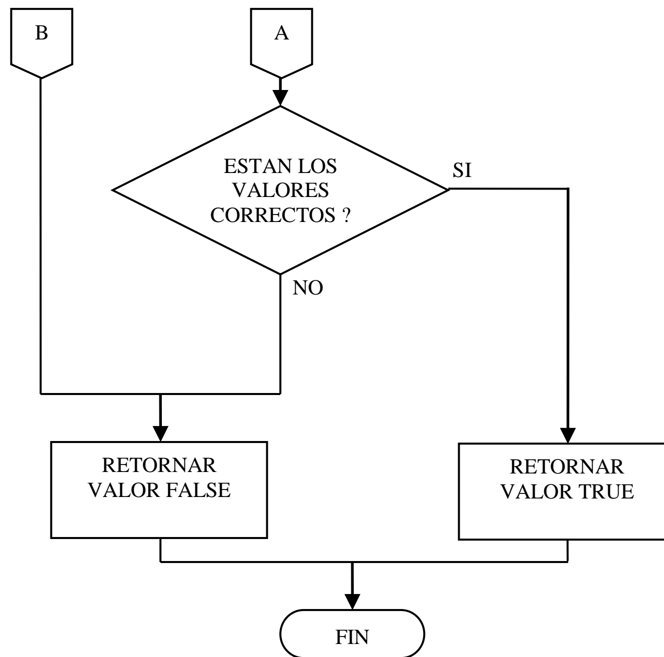


Figura 4.13 Flujograma de traducción de parámetros de objetos (b)

La traducción de parámetros de las instrucciones de tipo objeto esta definida por los siguientes pasos:

1. De la cadena recibida se extrae el ultimo carácter imprimible en pantalla y se verifica que este sea el signo de cierre de los parámetros "**)**", además de verificar que sólo exista uno y sólo un paréntesis de cierre "**)**" para uno y sólo un paréntesis de inicio "**(**", los cuales delimitan el inicio y final de los valores respectivamente, si la condición se cumple se procede al siguiente paso, caso contrario se genera un error de traducción y se devuelve el valor FALSE a la función de donde fue llamado.
2. El nombre y los valores son separados en dos vectores de cadena, esto para verificar que tipo de parámetro es al que se esta haciendo referencia. Se procede a una serie de comparaciones de la siguiente forma:

a. Parámetro es igual a dim(), si el resultado es verdadero se llama a la función revDim(), con la excepción que esta función es una para cada tipo de instrucción, esto debido a que cada objeto que se dibuja tiene diferente número de valores para su representación, caso contrario se realiza la siguiente verificación. Este parámetro es de tipo **obligatorio**.

Sintaxis del parámetro dim() para cada instrucción

ESFERA

dim(

f**lotante** *radio,*

entero *slices,*

entero *stacks*

);

Valores

radio, valor del radio de la esfera.

slices, número de líneas verticales con las que se dibuja la esfera.

stacks, número de líneas horizontales con las que se dibuja la esfera.

Función que lo evalúa

bool revDimEsf (

char * *parDim*,

HWND *hwnd* **);**

parámetros

parDim, cadena de caracteres en la que se encuentran los valores del parámetro.

hwnd, manejador de la ventana.

<u>CILINDRO</u>

dim(

flotante *radio_base,*

flotante *radio_altura,*

flotante *altura,*

entero *slices*,

entero *stacks*,

);

Valores

radio_base, valor del radio de la base.

radio_altura, valor del radio de la altura, para obtener un cono basta con dejar el valor de uno de los dos radios en cero.

altura, altura del cilindro.

slices, número de líneas verticales con las que se dibuja el cilindro.

stacks, número de líneas horizontales con las que se dibuja el cilindro.

Función que lo evalúa

bool revDimCil (

char * *parDim*,

HWND *hwnd* **);**

parámetros

parDim, cadena de caracteres en la que se encuentran los valores del parámetro.

hwnd, manejador de la ventana.

DISCO

dim(

flotante *radio_interno,*

flotante *radio_externo,*

entero *slices,*

entero *lazos*

);

Valores

radio_interno, valor del radio interno.

radio_externo, valor del radio externo, este tiene que ser mayor al radio interno.

slices, número de líneas que dibujan el disco.

lazos, número de lazos que dibujan el disco, estos son los sub-discos que dibujan el disco principal.

Función que lo evalúa

bool revDimDis(

  char * *parDim*,

  HWND *hwnd* **);**

parámetros

parDim, cadena de caracteres en la que se encuentran los valores del parámetro.

hwnd, manejador de la ventana.

CUBO

dim(

  flotante *alto,*

  flotante *ancho,*

  flotante *profundo*

  );

Valores

alto, valor de la altura del cubo, medido con respecto al eje **y**.

ancho, valor del ancho del cubo, medido con respecto al eje **x**.

profundo, profundidad del cubo, medido con respecto al eje **z**.

Función que lo evalúa

bool revDimCub(

char * *parDim*,

HWND *hwnd* **);**

parámetros

parDim, cadena de caracteres en la que se encuentran los valores del parámetro.

hwnd, manejador de la ventana.

PIRAMIDE

dim(

flotante *alto,*

flotante *ancho,*

flotante *profundo*

);

Valores

alto, valor de la altura de la pirámide, medido con respecto al eje **y**.

ancho, valor del ancho de la pirámide, medido con respecto al eje **x**.

profundo, profundidad de la pirámide, medido con respecto al eje **z**.

Función que lo evalúa

bool revDimPir(

char * *parDim*,

HWND *hwnd* **);**

parámetros

parDim, cadena de caracteres en la que se encuentran los valores del parámetro.

hwnd, manejador de la ventana.

LINEA

dim(

flotante *x1,*

flotante *y1,*

flotante *z1,*

flotante *x2,*

flotante *y2,*

flotante *z2*

);

Valores

Se dibuja una línea en el espacio tridimensional a partir del punto (x*1, y1 ,z1)* hasta el punto *(x2, y2, z2).*

Función que lo evalúa

bool revDimLin(

char * *parDim*,

HWND *hwnd* **);**

parámetros

parDim, cadena de caracteres en la que se encuentran los valores del parámetro.

hwnd, manejador de la ventana.

<u>TRIANGULO</u>

dim(

f**lotante** *x1,*

f**lotante** *y1,*

f**lotante** *z1,*

f**lotante** *x2,*

f**lotante** *y2,*

f**lotante** *z2,*

f**lotante** *x3,*

f**lotante** *y3,*

f**lotante** *z3*

);

Valores

Se dibuja un triángulo con los tres puntos (*x1, y1, z1*) (*x2, y2, z2*) (*x3, y3, z3*) dados como vértices del triángulo.

Función que lo evalúa

bool revDimTri (

char * *parDim*,

HWND *hwnd* **);**

parámetros

parDim, cadena de caracteres en la que se encuentran los valores del parámetro.

hwnd, manejador de la ventana.

<u>CUADRO</u>

dim(

f**lotante** *alto,*

f**lotante** *ancho,*

);

Valores

Se dibuja un cuadro desde el punto que actualmente es el origen (0, 0, 0) hasta el vértice que se forma con el punto (*alto, ancho, 0)*.

Función que lo evalúa

bool revDimCua (

char * *parDim*,

HWND *hwnd* **);**

parámetros

parDim, cadena de caracteres en la que se encuentran los valores del parámetro.

hwnd, manejador de la ventana.

CIRCULO

dim(

f**lotante** *radio,*

);

Valores

Se dibuja un círculo desde el punto que actualmente es el origen (0, 0, 0) con el valor de radio igual a *radio.*

Función que lo evalúa

bool revDimCir (

char * *parDim*,

HWND *hwnd* **);**

parámetros

parDim, cadena de caracteres en la que se encuentran los valores del parámetro.

hwnd, manejador de la ventana.

PUNTO

dim(

flotante *x1,*

flotante *y1,*

flotante *z1* **);**

Valores

Se dibuja un punto en el espacio tridimensional con coordenadas (x*1, y1 ,z1)*.

Función que lo evalúa

bool revDimPun(

char * *parDim*,

HWND *hwnd* **);**

parámetros

parDim, cadena de caracteres en la que se encuentran los valores del parámetro.

hwnd, manejador de la ventana.

b. Parámetro es igual a color(), si el resultado es verdadero se llama a la función revColor(), caso contrario se realiza la siguiente verificación.

   Este parámetro es común para todos los objetos, debido a eso sólo se necesita de una función para su evaluación.

Sintaxis del parámetro color

color(

flotante *rojo,*

flotante *verde,*

flotante *azul* **);**

Valores

rojo, verde, azul, Intensidad de rojo, verde y azul respectivamente.

Función que lo evalúa

bool revColor(

char * *parColor*,

HWND *hwnd* **);**

parámetros

parColor, cadena de caracteres en la que se encuentran los valores del parámetro.

hwnd, manejador de la ventana.

c. Parámetro es igual a rotar(), si el resultado es verdadero se llama a la función revRotar(), caso contrario se realiza la siguiente verificación. Parámetro común para todos los objetos.

Sintaxis del parámetro rotar

rotar(

flotante *angulo,*

flotante *x,*

flotante *y,*

flotante *z*

);

Valores

angulo, ángulo de rotación medido en grados. El ángulo produce una rotación en contra de las agujas del reloj

x,y,z, vector dirección desde el origen que es usado como eje de rotación.

Función que lo evalúa

bool revRotar(

char * *parRotar*,

HWND *hwnd* **);**

parámetros

parRotar, cadena de caracteres en la que se encuentran los valores del parámetro.

hwnd, manejador de la ventana.

d. Parámetro es igual a trasladar(), si el resultado es verdadero se llama a la función revTrasladar(), caso contrario se realiza la siguiente verificación.

   Parámetro común para todos los objetos.

   Parámetro de tipo **opcional.**

Sintaxis del parámetro trasladar

trasladar(

  flotante *x,*

  flotante *y,*

  flotante *z*

  );

Valores

x,y,z, coordenadas de un vector de traslación.

Función que lo evalúa

bool revTrasladar(

  char * *parTrasladar*,

  HWND *hwnd* **);**

parámetros

parTrasladar, cadena de caracteres en la que se encuentran los valores del parámetro.

hwnd, manejador de la ventana.

e. Parámetro es igual a transparencia(), si el resultado es verdadero se llama a la función revTransparencia(), caso contrario se realiza la siguiente verificación.

   Parámetro común para todos los objetos.

   Parámetro de tipo **opcional.**

Sintaxis del parámetro transparencia

transparencia(

  flotante *transparencia*

  );

Valores

transparencia, valor de la transparencia en una escala de 0 a 1, entre menor es el valor es más transparente el objeto.

Función que lo evalúa

bool revTransparencia(

  char * *parTrans*,

  HWND *hwnd* **);**

parámetros

parTrans, cadena de caracteres en la que se encuentran los valores del parámetro.

hwnd, manejador de la ventana.

f. Parámetro es igual a imagen(), si el resultado es verdadero se llama a la función revImagen(), caso contrario se realiza la siguiente verificación.

   Parámetro común para los objetos de tipo esférico como lo son esfera, cilindro y disco.

   Parámetro de tipo **opcional.**

Sintaxis del parámetro imagen

imagen(

   cadena *ruta_archivo_BMP*

   );

Valores

ruta_archivo_BMP, nombre de la ruta de donde se obtiene la imagen de tipo BMP, el tamaño de esta tiene que ser en potencia 2 de ancho y de alto, por ejemplo 256x256 píxeles.

Función que lo evalúa

bool revImagen(

char * *parImg*,

HWND *hwnd* **);**

parámetros

parImg, cadena de caracteres en la que se encuentran los valores del parámetro.

hwnd, manejador de la ventana.

g. Parámetro es igual a modo(), si el resultado es verdadero se llama a la función revRotar(), caso contrario se realiza la siguiente verificación.

Parámetro común para todos los objetos.

Parámetro de tipo **opcional.**

Sintaxis del parámetro modo

modo(

cadena *modo*

);

Valores

modo, especifica el modo de dibujar del objeto, este puede ser **lleno, línea** y **punto,** el primero dibuja un objeto de forma llena, el segundo

dibuja sólo las líneas del polígono, y el tercero dibuja los puntos de los vértices.

Función que lo evalúa

bool revModo(

char * *parModo*,

HWND *hwnd* **);**

parámetros

ParModo, cadena de caracteres en la que se encuentran los valores del parámetro.

hwnd, manejador de la ventana.

h. Parámetro es igual a id(), si el resultado es verdadero se llama a la función revId(), caso contrario se retorna un valor FALSE porque no existe parámetro.

   Parámetro común para todos los objetos.

   Parámetro de tipo **obligatorio.**

Sintaxis del parámetro id

id(

flotante *transparencia*

);

Valores

transparencia, valor de la transparencia en una escala de 0 a 1, entre menor es el valor es más transparente el objeto.

Función que lo evalúa

bool revTransparencia(

char * *parTrans*,

HWND *hwnd* **);**

parámetros

parTrans, cadena de caracteres en la que se encuentran los valores del parámetro.

hwnd, manejador de la ventana.

3. Si los valores son correctos en cada una de esta funciones se devuelve un valor TRUE a la función de donde fueron llamadas, caso contrario se devuelve un valor FALSE.

Flujograma para la traducción de parámetros de ambiente

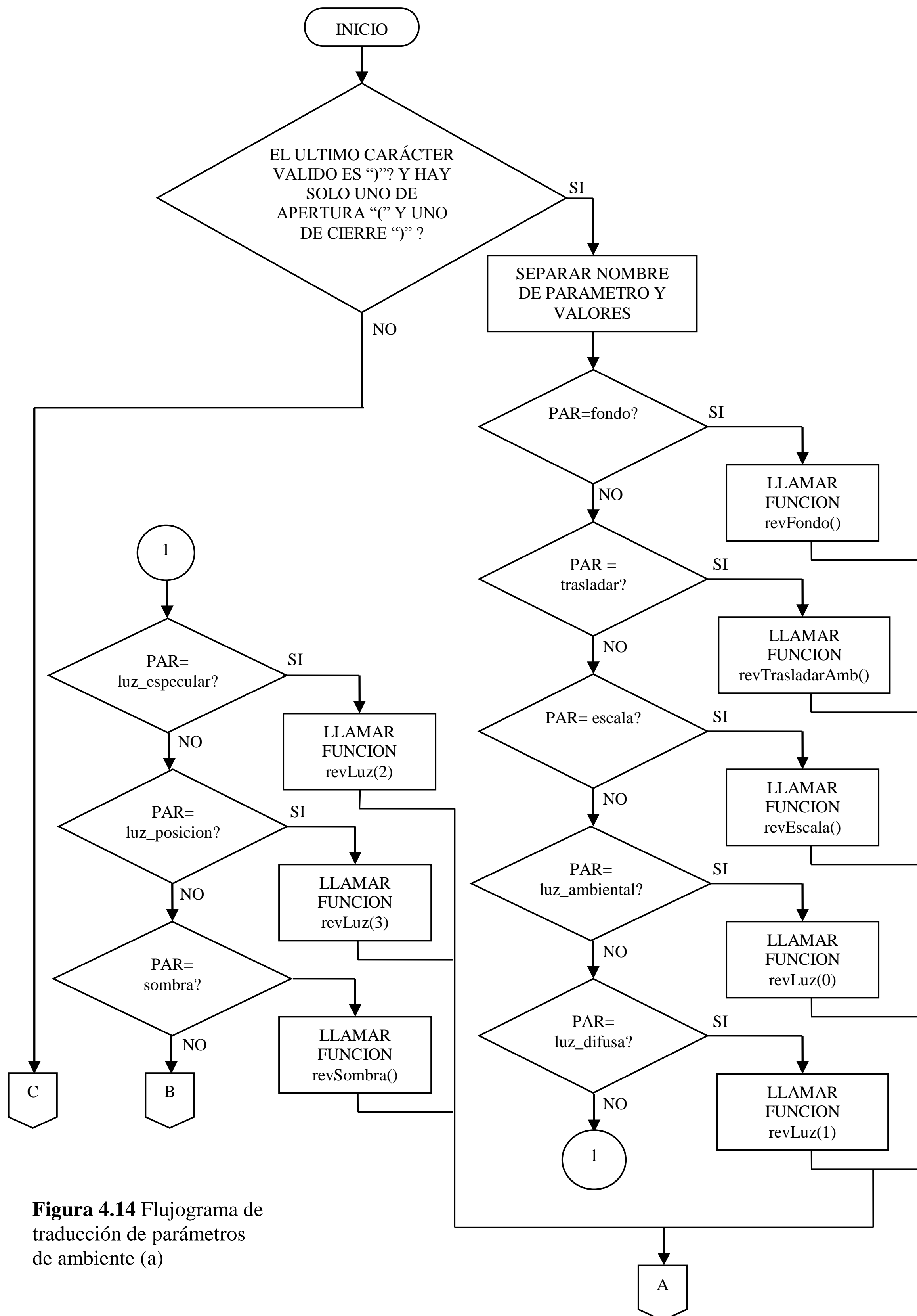


Figura 4.14 Flujograma de traducción de parámetros de ambiente (a)

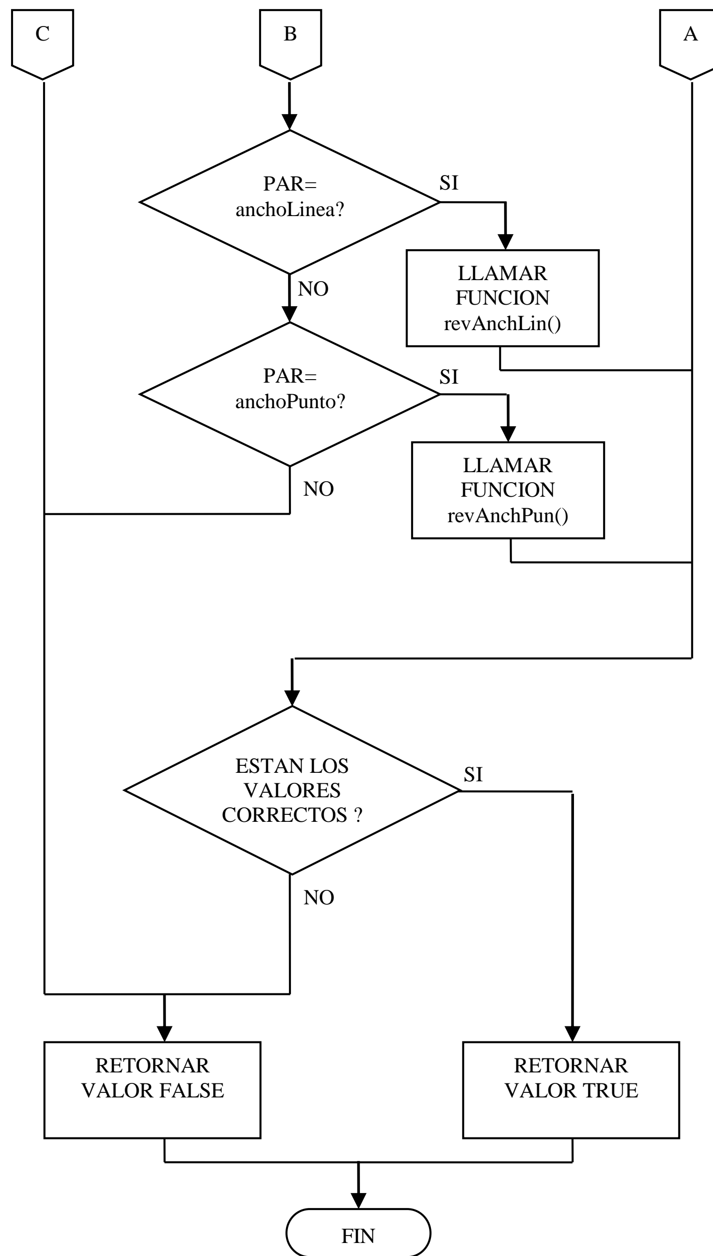


Figura 4.15 Flujograma de traducción de parámetros de ambiente (b)

La traducción de parámetros de las instrucciones de tipo ambiente esta definida por los siguientes pasos:

1. De la cadena recibida se extrae el ultimo carácter imprimible en pantalla y se verifica que este sea el signo de cierre de los parámetros "**)**", además de

verificar que sólo exista uno y sólo un paréntesis de cierre "**)**" para uno y sólo un paréntesis de inicio "**(**", los cuales delimitan el inicio y final de los valores respectivamente, si la condición se cumple se procede al siguiente paso, caso contrario se genera un error de traducción y se devuelve el valor FALSE a la función de donde fue llamado.

2. El nombre y los valores son separados en dos vectores de cadena, esto para verificar que tipo de parámetro es al que se esta haciendo referencia. Se procede a una serie de comparaciones de la siguiente forma:

    a. Parámetro es igual a fondo(), si el resultado es verdadero se llama a la función revFondo(),caso contrario se realiza la siguiente verificación. Este parámetro es de tipo **opcional**.

Sintaxis del parámetro fondo

fondo(

flotante *rojo,*

flotante *verde,*

flotante *azul*

);

Valores

rojo, Intensidad de rojo.

verde, intensidad de verde.

azul, intensidad de azul.

Función que lo evalúa

bool revFondo(

char * *parFondo*,

HWND *hwnd* **);**

parámetros

parColor, cadena de caracteres con los valores del parámetro.

hwnd, manejador de la ventana.

b. Parámetro es igual a trasladar(), si el resultado es verdadero se llama a la función revTrasladarAmb(). Esta función es diferente a la utilizada en los parámetros del objeto, pues se utiliza para trasladar toda la escena antes que esta sea dibujada, como se mencionaba en el capitulo III de este libro son las *transformadas de vista*. Si el parámetro no es trasladar() se realiza la siguiente verificación.

   Este parámetro es de tipo **opcional**.

Sintaxis del parámetro trasladar

trasladar(

flotante *x,*

flotante *y,*

flotante *z*

);

Valores

x,y,z, coordenadas de un vector de traslación.

Función que lo evalúa

bool revTrasladarAmb(

char * *parTrasAmb*,

HWND *hwnd* **);**

parámetros

parTrasAmb, cadena de caracteres con los valores del parámetro.

hwnd, manejador de la ventana.

c. Parámetro es igual a luz_ambiental() ó luz_especular() ó luz_difusa ó luz_posicion(). Si el resultado es verdadero se llama a la función revLuz(). La función que evalúa los parámetros de iluminación es la misma para los tres tipos de luz que se pueden utilizar Ambiental, Difusa y Especular, así como para la posición de esta dentro de la escena.

Sintaxis de parámetros luz_ambiental | luz_difusa | luz_especular | luz_posicion

trasladar(

flotante *rojo,*

flotante *verde,*

flotante *azul,*

flotante *intensidad*

);

Valores

rojo, intensidad de rojo ó posición en **x** para luz_posicion().

verde, intensidad de verde ó posición en **y** para luz_posicion().

azul, intensidad de azul ó posición en **z** para luz_posicion().

intensidad, factor de incidencia de los colores sobre los objetos. Cuando es el parámetro luz_posicion, este maneja la posición de la luz, si tiene 1 como valor, significa que la luz está ubicada exactamente en ese lugar, otro valor, la luz es direccional y los rayos serán paralelos.

Función que lo evalúa

bool revLuz(

char * *parLuz,*

entero *tipo*,

HWND *hwnd* **);**

parámetros

parTrasAmb, cadena de caracteres con los valores del parámetro.

tipo, maneja el tipo de luz, los valores pueden ser:

- 0, luz ambiental
- 1, luz difusa

- 2, luz especular
- 3, posición de la luz

hwnd, manejador de la ventana.

d. Parámetro es igual a sombra(), si el resultado es verdadero se llama a la función revSombra(), caso contrario se devuelve valor FALSE y finaliza la revisión de parámetros de tipo ambiente. Este parámetro es de tipo **opcional**.

Sintaxis del parámetro sombra

sombra(

  flotante *transparencia*

  );

Valores

transparencia, valor de transparencia con que se dibujara la sombra.

Función que lo evalúa

bool revSombra(

  char * *parSombra,*

  HWND *hwnd* **);**

parámetros

parSombra, cadena de caracteres con los valores del parámetro.

hwnd, manejador de la ventana.

e. Parámetro es igual a anchoLinea(), si el resultado es verdadero se llama a la función revAnchLin(), caso contrario se devuelve valor FALSE y finaliza la revisión de parámetros de tipo ambiente. Este parámetro es de tipo **opcional**.

Sintaxis del parámetro anchoLinea

anchoLinea(

flotante *anchoLinea*

);

Valores

anchoLinea, valor en pixeles con el cual se dibujarán las líneas dentro de la escena.

Función que lo evalúa

bool revAnchLin(

char * *parAnchLin,*

HWND *hwnd* **);**

parámetros

parAnchLin, cadena de caracteres con los valores del parámetro.

hwnd, manejador de la ventana.

f. Parámetro es igual a anchoPunto(), si el resultado es verdadero se llama a la función revAnchPun(), caso contrario se devuelve valor FALSE y finaliza la revisión de parámetros de tipo ambiente. Este parámetro es de tipo **opcional**.

Sintaxis del parámetro anchoPunto

anchoPunto(

flotante *anchoPunto*

);

Valores

anchoPunto, valor en pixeles con el cual se dibujarán los puntos dentro de la escena, por ejemplo, los vértices de los polígonos cuando se dibuja en modo *punto.*

Función que lo evalúa

bool revAnchPun(

char * *parAnchPun,*

HWND *hwnd* **);**

parámetros

parAnchPun, cadena de caracteres con los valores del parámetro.

hwnd, manejador de la ventana.

3. Si los valores son correctos en cada una de esta funciones se devuelve un valor TRUE a la función de donde fueron llamadas, caso contrario se devuelve un valor FALSE.

Flujograma para la revisión de valores

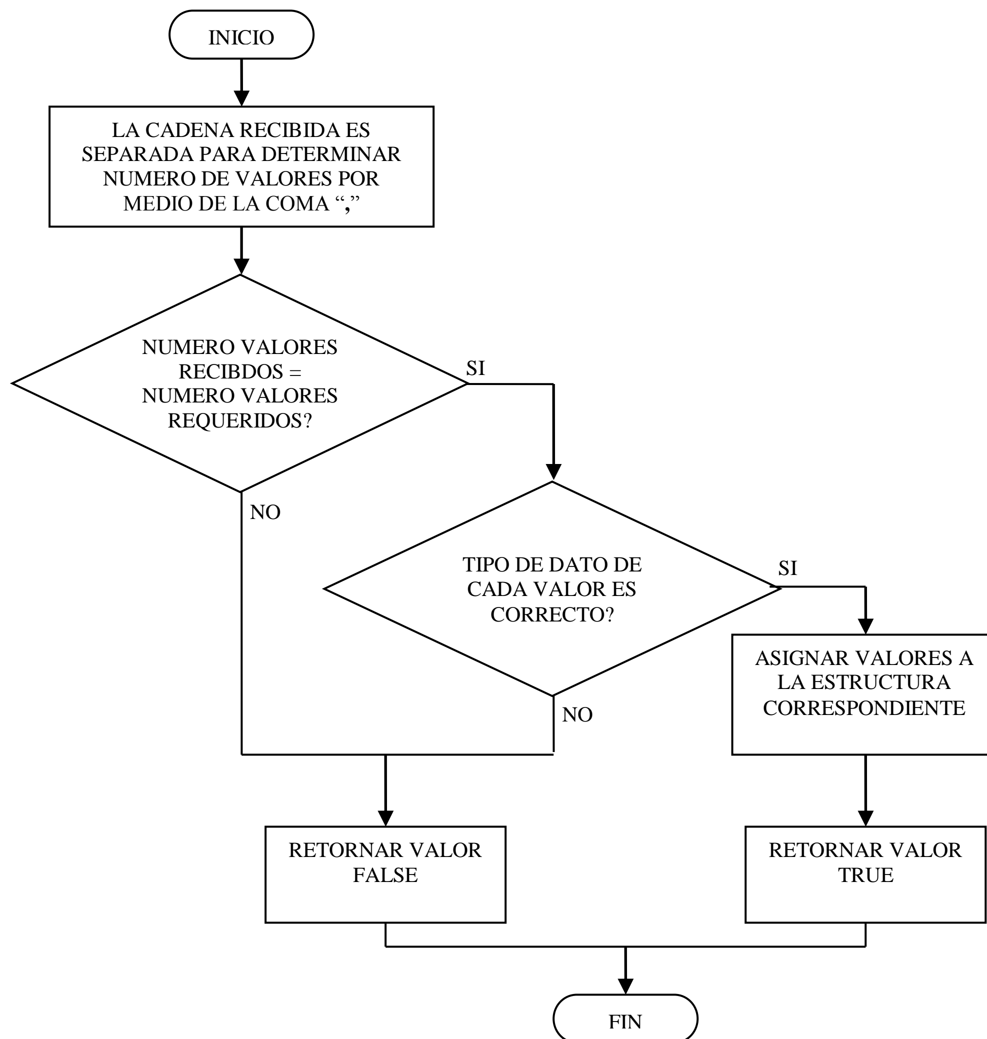


Figura 4.16 Flujograma para la revisión de valores

El flujograma mostrado en la figura 4.13 es el mismo para todas las funciones que se encargan de la revisión de los valores de cada uno de los parámetros, los pasos que se siguen son los siguientes:

1. La cadena que fue recibida es dividida mediante las comas que separa cada uno de los valores, aquí se puede determinar el número de estos para su futura comparación.
2. Si el número de valores obtenidos es igual al número de valores requeridos por el parámetros se procede a la verificación del tipo de dato de cada uno de estos, caso contrario se retorna el valor FALSE.
3. Si el tipo de dato de cada uno de los valores es correcto, se procede a la asignación de estos a la estructura que es utilizada en el programa Dibujar.cpp y se retorna el valor TRUE. Caso contrario, se retorna el valor FALSE.

Estructuras utilizadas para la asignación de los valores

```
typedef struct Objeto
{
    char tipo[4];                                          Tipo de objeto
    bool bId;
    char cId[15];                                          Identificador
    bool Dim;
    double radio, radio2, alto, x2, y2, z2, x3, y3, z3
    int slices, stacks;                                    Dimensión
```

```
    bool Color;
    double red, green, blue;              // Color
    bool modo;
    int nModo;                            // Modo de dibujo
    bool Trasladar;
    double tx, ty, tz;                    // Traslación
    bool Rotar;
    double angulo, rx, ry, rz;            // Rotación
    bool Transpar;
    double Alpha;                         // Factor de Transparencia
    bool img;
    char imgName[256];                    // Ruta de la imagen
}ObjetoGL

typedef struct ambiente
{
    bool AmbProp;                         // Verificar cambio en propiedades
    bool fondo;
    double fRed,fGreen,fBlue;             // Fondo
    bool trasladar;
    double tx,ty,tz;                      // Traslación de la escena
    bool luzAmb;
    double aRed,aGreen,aBlue,aAlpha;      // Luz Ambiental
    bool luzDif;
```

```
double dRed,dGreen,dBlue,dAlpha;          // Luz Difusa
bool luzEsp;
double eRed,eGreen,eBlue,eAlpha;          // Luz Especular
bool posicion;
double pRed,pGreen,pBlue,pos;             // Fondo
bool escala;
double ex,ey,ez;                          // Escala
bool Sombra;
double sValor;                            // Sombra
bool AnchLin;
double LinValor;                          // Ancho de línea
bool AnchPun;
double PunValor;                          // Ancho de punto
}AmbienteGL;
```

4.4. PROGRAMA DIBUJAR

Flujograma general del programa Dibujar.cpp

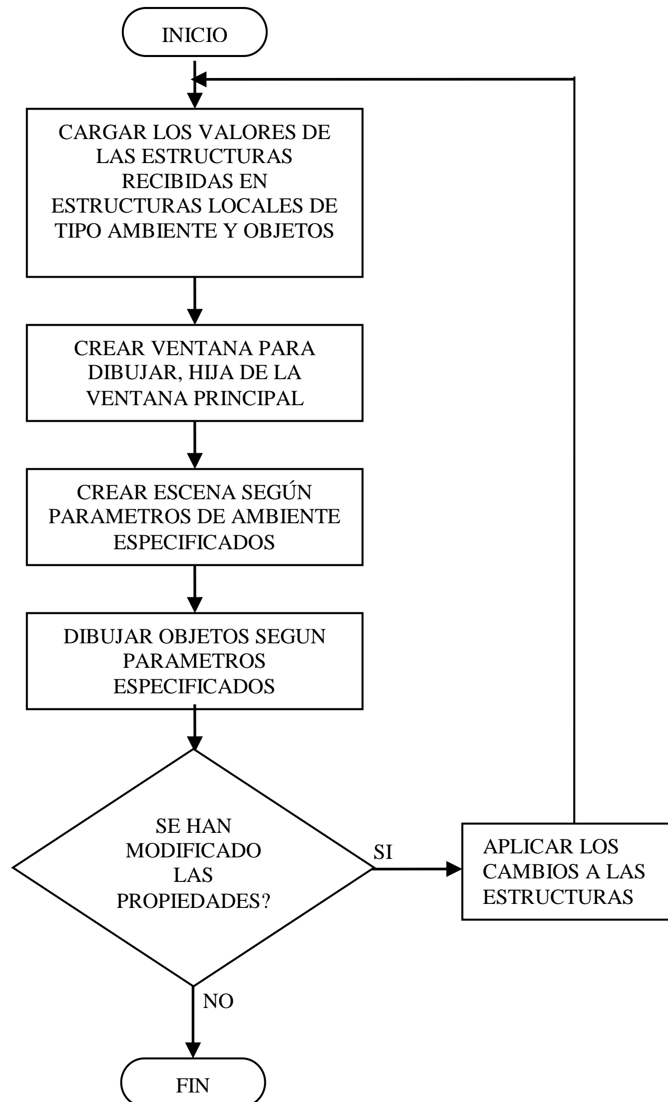


Figura 4.17 Flujograma general del programa Dibujar.cpp

El proceso general para dibujar se resume en los siguientes pasos:

1. Las estructuras que se obtuvieron del programa Traductor.cpp son cargadas en estructuras locales del programa Dibujar.cpp, esto con el fin de poder trabajar con una copia de cada estructura para modificaciones que se realicen dentro de este.
2. Se crea la ventana en la que será dibujada la escena. Esta ventana es hija de la ventana principal y el título de esta es la ruta de donde se obtuvo el archivo para la escena que se dibuje.
3. Los objetos son dibujados en la escena según los parámetros que se hayan especificado en el archivo de texto (este proceso es detallado en las figuras 4.17 y 4.19, flujograma de inicialización de escena y flujograma para el dibujo de objetos respectivamente).
4. Se verifica si se ha solicitado la modificación de objetos o propiedades de escena para aplicar estos cambios inmediatamente, caso contrario finaliza el proceso.

Flujograma de inicialización de la escena

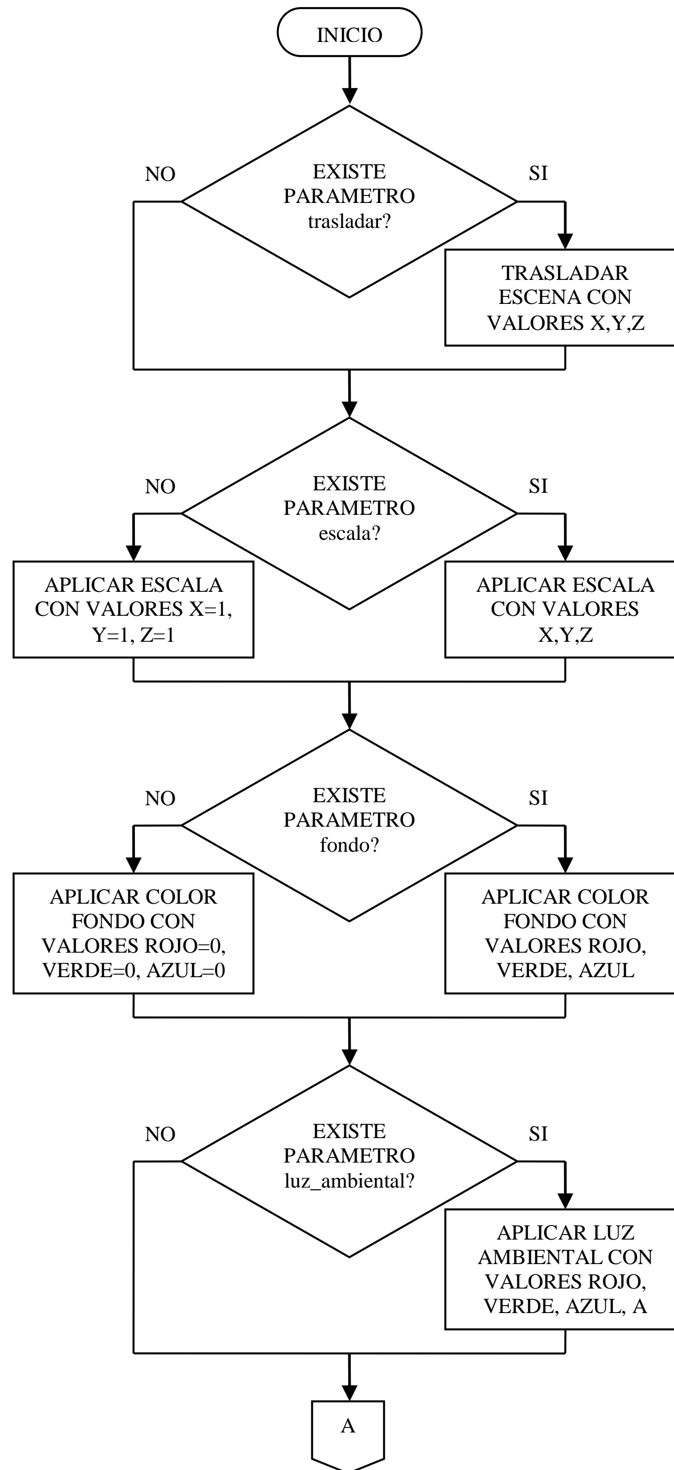


Figura 4.18 Flujograma de inicialización de escena (a)

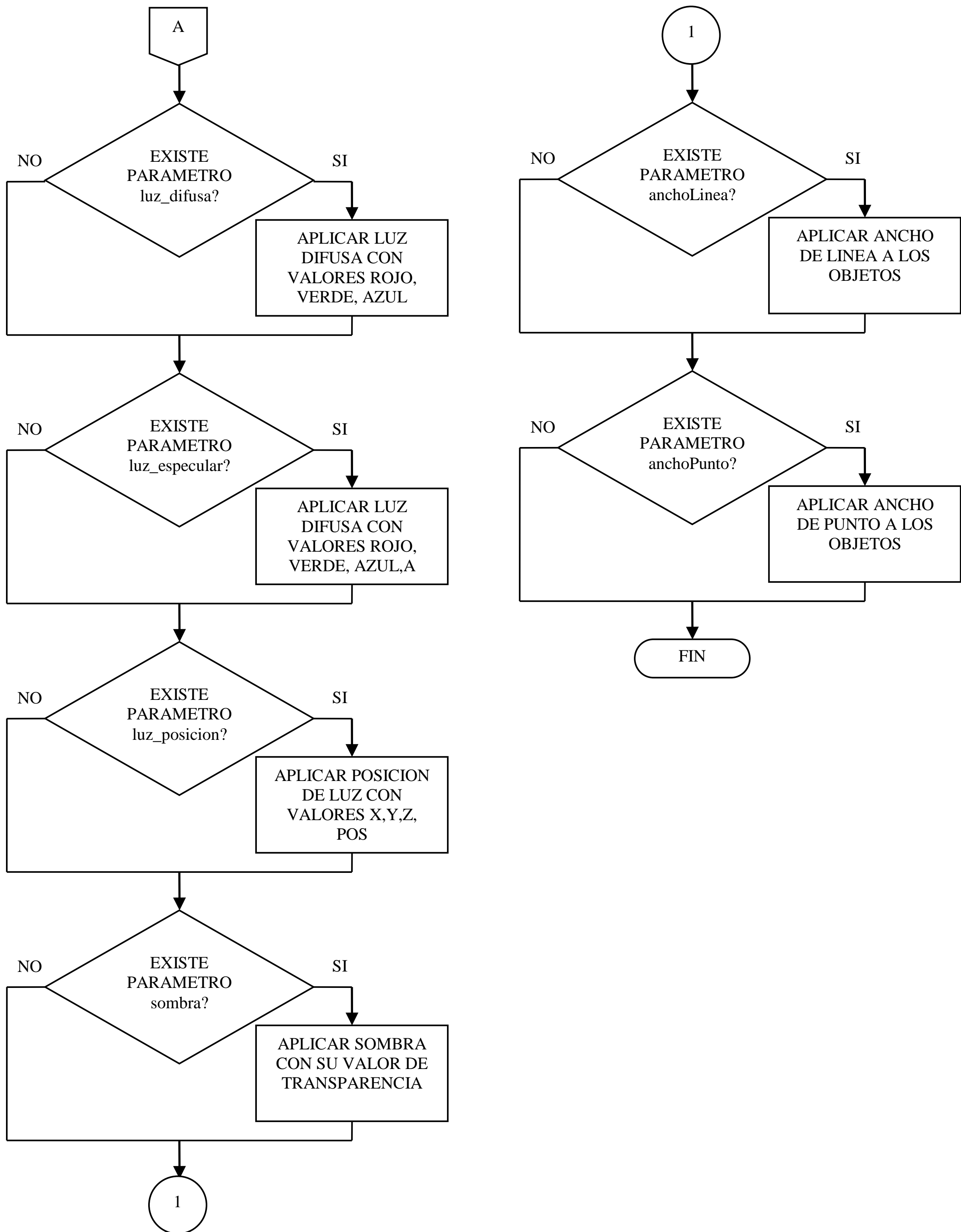


Figura 4.19 Flujograma de inicialización de escena (b)

La preparación de la escena a representar es llevada a cabo por la función de inicialización, esta se encarga de mostrar la escena según los parámetros que se han especificado en el archivo, los pasos que se siguen para lograr dicho fin son los siguientes:

1. Se verifica si existe el parámetro trasladar. Si existe, se traslada la escena con los valores de x, y, z especificados en el archivo, caso contrario la escena aparecerá con traslación x=0, y=0, z=0, esto implica que la escena estará en el monitor y los objetos con valor de z positivo estarán detrás del observador.

2. Se verifica si existe el parámetro escala. El valor de escala es aplicado a todos los objetos que se dibujen en la escena si este es definido en el archivo en la parte de ambiente, caso contrario se toma el valor por defecto de x=1, y=1, z=1, para que los objetos aparezcan con su tamaño real. Los valores de las coordenadas x, y, z, pueden ser diferentes pero esto provocaría el estiramiento o acortamiento de los objetos que se estén dibujando.

3. Se verifica si existe el parámetro fondo. El color de fondo de la escena se especifica mediante este parámetro con los valores de rojo, verde y azul. Si éste no existe se toma el valor por defecto de rojo=0, verde=0, y azul=0, esto implica un color negro de fondo.

   La instrucción utilizada para fijar el color de la escena es la siguiente:

   glClearColor(float *rojo,* **float** *verde,* **float** *azul,* **float** *alpha***);**

Parámetros

rojo, verde, azul, alpha, valores de rojo, verde, azul y alfa con los que se le aplica color al fondo. El valor de *alpha* no es requerido en la instrucción del archivo *.rep, se toma en la aplicación con valor de 1.

4. Se verifica si existen los parámetros de luz_ambiental, luz_difusa, luz_especular, luz_posicion. Estos parámetros son los encargados de aplicar la iluminación a la escena y la posición de esta, si se encuentran dentro del archivo se habilita el tipo de luz que corresponde el parámetro, esto por medio de la siguiente instrucción dentro de la aplicación:

glLightfv(GLenum *luz,* **GLenum** *nombre,* **GLfloat** **parametros* **);**

Parámetros

luz, número de luz, que para la aplicación toma el valor de LIGHT0.

nombre, nombre de la luz, esta puede ser:

- GL_AMBIENTAL
- GL_DIFFUSE
- GL_SPECULAR
- GL_POSITION

parametros, vector que especifica los valores de la luz o la posición.

5. Se verifica si existe el parámetro anchoLinea. Por medio de este parámetro se selecciona el ancho de la línea con la que los objetos se dibujarán cuando se seleccione modo de dibujo *linea,* este es medido en pixeles. La instrucción utilizada es la siguiente:

   glLineWidth(GLfloat *ancho* **);**

   Parámetros

   ancho, ancho de las líneas medidas en pixeles. Si este valor no es encontrado en el archivo por defecto las líneas se dibujarán con ancho igual a uno.

6. Se verifica si existe el parámetro anchoPunto. Cuando se selecciona el modo de dibujo *punto* en los objetos se aplica el ancho del punto para los vértices que están siendo representados. La instrucción utilizada en la aplicación para este fin es mostrada a continuación:

   gPointSize(GLfloat *ancho* **);**

   Parámetros

   ancho, ancho de los puntos medidos en pixeles. Si este valor no es encontrado en el archivo por defecto los puntos se dibujarán con ancho igual a uno.

7. Se verifica si existe el parámetro sombra. La sombra recibe como parámetro el valor de la transparencia que se le aplicará a esta para simular un efecto de sombra completamente negra o sombra transparente.

   El proceso de aplicar sombra a los objetos es uno de los más complejos ya que requiere de muchos cálculos matemáticos así como de uso de los recursos del sistema, debido a esto se puede notar una considerable reducción del tiempo de respuesta por parte de la aplicación.

8. Una vez terminado el proceso de fijar los parámetros de la escena se procede a la representación de los objetos que esta contiene.

Flujograma para el dibujo de objetos

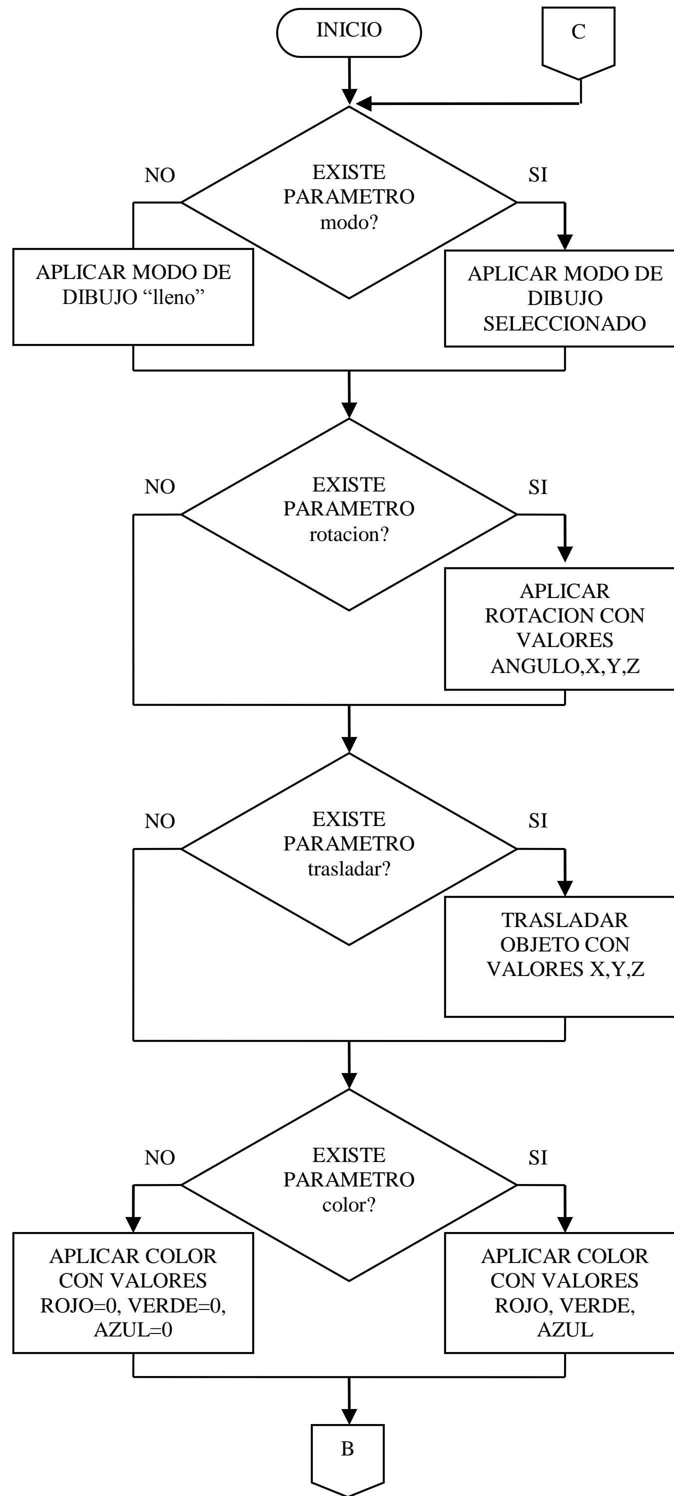


Figura 4.20 Flujograma para el dibujo de objetos (a)

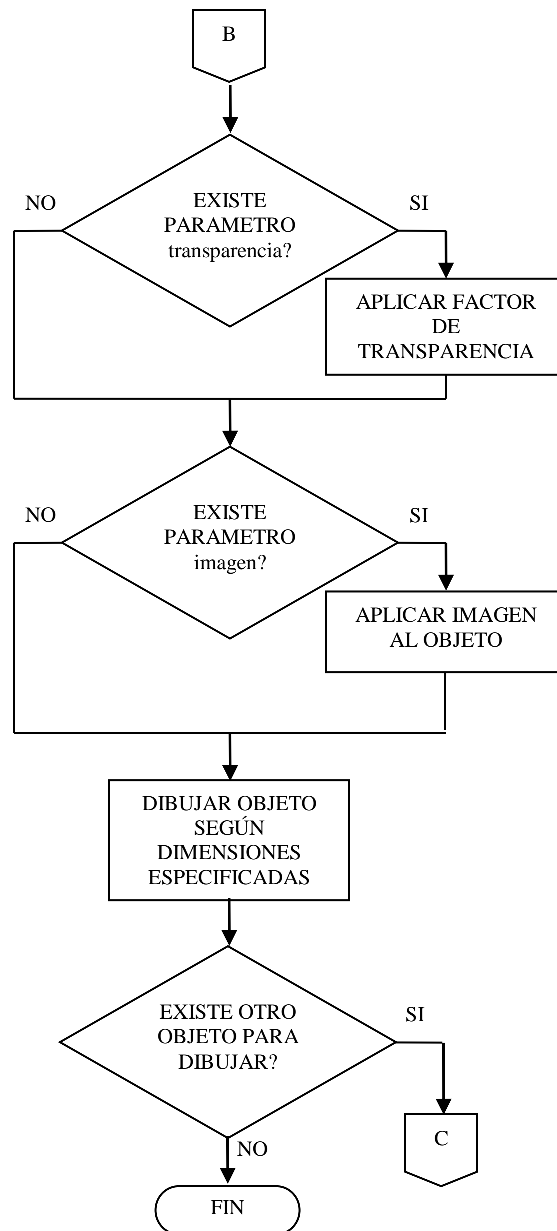


Figura 4.21 Flujograma para el dibujo de objetos (b)

Los objetos son tomados de la estructura del tipo ObjetoGL, donde se encuentra la colección de propiedades que cada uno de estos posee, cada una de estas propiedades tiene una estructura definida que consta de, un valor de tipo booleano que especifica si el objeto posee o no la propiedad y el conjunto de valores numéricos o de caracteres de la propiedad.

Cada una de las propiedades conlleva a la realización de procesos por parte de la aplicación, estos se detallan con más claridad en cada uno de los siguientes pasos:

1. Verificar si existe el parámetro modo. Este parámetro especifica el modo en que los objetos serán dibujados, si existe el parámetro el objeto se dibuja según el modo que se haya especificado, de otra forma se toma el valor por defecto de dibujo lleno.

   El modo del objeto se selecciona mediante la siguiente instrucción de OpenGL:

   glPolygonMode(
   GLenum *cara*,
   GLenum *modo*
   );

   Parámetros

   cara, las caras a las que el modo se le aplicará. Estas pueden ser GL_FRONT para la cara de enfrente del polígono, GL_BACK para la cara de atrás del polígono, ó GL_FRONT_AND_BACK para ambas caras del polígono. Esta última es la utilizada por la aplicación para aplicar uniformemente el modo de dibujo de los objetos.

modo, modo en que los objetos serán dibujados. Los siguientes son los modos que se utilizan:

GL_FILL: Dibuja el interior del polígono de una forma llena.

GL_LINE: Dibuja solamente las líneas que definen al objeto.

GL_POINT: Dibuja los vértices del polígono representándolos con puntos en cada uno de ellos.

2. Verificar si existe el parámetro rotación. La rotación del objeto esta dada por el ángulo medido en grados y en contra de las agujas del reloj, y el vector que se utiliza para realizar la rotación, si este parámetro es encontrado en el archivo de entrada de la aplicación, entonces se multiplica la matriz actual de rotación por el valor especificado y las nuevas rotaciones se realizarán a partir de esta nueva matriz. Si el parámetro no existe, se dibuja el objeto con sin aplicar cambios a la matriz de rotación actual.

La instrucción que se encarga de esto en OpenGL es la siguiente:

glRotated(

GLdouble *angulo,*

GLdouble *x,*

GLdouble *y,*

GLdouble *z,*

);

Parámetros

angulo, valor del ángulo medido en grados.

x, y, z, coordenadas de un vector x, y, z respectivamente. Este vector es utilizado como eje de rotación.

Explicación

La llamada a la función **glRotate** genera una matriz que realiza una rotación en contra de las agujas del reloj del ángulo medido en grados sobre desde el origen hasta el punto (x, y, z).

La matriz actual es multiplicada por esta matriz de rotación reemplazando sus valores con el resultado de la operación. De esta forma, si M es la matriz actual y R es la matriz de traslación, entonces M es reemplazada por M•R.

Todos los objetos que se dibujen a continuación de la llamada de **glRotated** son rotados.

3. Verificar si existe el parámetro traslación. Así como el parámetro de rotación, el parámetro de traslación afecta la matriz actual con la que se esta trabajando, si no se especifica este parámetro el objeto que se dibuje se sobrepondrá al primero que se dibujó.

La instrucción OpenGL es la siguiente:

```
glTranslated(
    GLdouble angulo,
    GLdouble x,
    GLdouble y,
    GLdouble z,
);
```

Parámetros

x, y, z, coordenadas de un vector de traslación.

Explicación

Esta función mueve el sistema de coordenadas origen a un punto especificado por (x, y, z). El vector de traslación es utilizado para generar una nueva matriz de traslación de 4x4:

$$\begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & x \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & y \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & z \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{pmatrix}$$

La matriz actual es multiplicada por esta matriz de traslación reemplazando sus valores con el resultado de la operación. De esta forma, si M es la matriz actual y T es la matriz de traslación, entonces M es reemplazada por M•T.

Todos los objetos que se dibujen a continuación de la llamada de **glTranslated** son trasladados.

4. Verificar si existe el parámetro color. Si este parámetro se encuentra, entonces se le aplica al objeto el color especificado con las intensidades de rojo, verde y azul, estos valores tienen que estar en el rango de 0 a 1, por ejemplo el color negro que es la ausencia de color le corresponde el valor rojo=0, verde=0 y azul=0, lo contrario al color blanco que es la presencia de todos los colores donde rojo=1, verde=1 y azul=1. Si este parámetro no es encontrado entonces la aplicación le asigna al objeto el color negro por defecto.

La instrucción OpenGL utilizada es la siguiente:

glColor3d(

GLdouble *red,*

GLdouble *green,*

GLdouble *blue*

);

Parámetros

red, green, blue, nuevos valores para rojo, verde y azul para el color actual.

5. Verificar si existe el parámetro transparencia. La transparencia se le puede aplicar a los objetos para simular múltiples objetos dentro de uno sólo por ejemplo, otra aplicación es el montaje de imágenes que parecen sobrepuestas una sobre la otra.

Instrucciones OpenGL utilizadas

glEnable(GL_BLEND**);**

glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA**);**

Esta combinación de instrucciones toma el color origen y le da una escala basándose en el componente alpha, luego lo agrega al color del píxel destino en la escala de 1 menos el valor alpha. De una forma más simple, esta función de degradación de colores (blending por su traducción al inglés) toma una fracción del color actual de dibujo y lo incrusta en el píxel de la pantalla. El componente alfa del color puede estar en un rango de 0 (completamente transparente) hasta 1 (completamente opaco), como se detalla a continuación:

Rd = Ro * Ao + Rd * (1 – Ao)

Vd = Vo * Ao + Vd * (1 – Ao)

Ad = Ao * Ao + Ad * (1 – Ao)

Donde

Rd: Rojo destino	Ro: Rojo origen	Ao: Alpha origen
Vd: Verde destino	Vo: Verde origen	
Ad: Azul destino	Ao: Azul origen	

6. Verificar si existe el parámetro imagen. El valor que recibe este parámetro es una cadena de texto que especifica la ruta física del archivo de imagen BMP que se le aplicará al objeto, el tamaño de esta debe ser en potencia de 2 (2^n) de ancho y de alto.

Instrucciones OpenGL utilizadas:

Instrucción

glGenTextures(

GLsizei *n*,

GLuint * *texturas*

);

Parámetros

n, número de nombres de imágenes a ser generadas, este valor es 1 dentro de la aplicación, puesto que sólo se puede generar una imagen por cada objeto.

texturas, puntero al primer elemento de un vector que almacena los nombres generados para las texturas. Este vector es declarado en la aplicación como GLuint * textura, en el que se almacena el nombre de la imagen que contiene cada objeto.

Instrucción

glBindTexture(

GLenum *destino*,

GLuint *textura*

);

Parámetros

destino, destino al cual la imagen será aplicada, puede tener cualquiera de los siguientes valores: GL_TEXTURE_1D ó GL_TEXTURE_2D, el valor que se ha tomado en la aplicación es el segundo, puesto que serán imágenes en dos dimensiones con alto y ancho.

textura, nombre de la textura.

Explicación

La principal función de **glBindTexture** es asignar un nombre de textura a un conjunto de datos de textura. En este caso se le está diciendo a OpenGL que

hay memoria disponible en *&textura.* Cuando se cree la textura será, almacenada en el espacio de memoria al que hace referencia *&textura.*

<u>Instrucción</u>

glTexImage2D(

GLenum *destino*,

GLint *nivel*,

GLint *componentes*,

GLsizei *ancho*,

GLsizei *alto*,

GLint *borde*,

GLenum *formato*,

GLenum *tipo*,

const GLvoid **pixeles*

);

Parámetros

destino, la imagen destino. Esta debe ser GL_TEXTURE_2D.

nivel, nivel de detalle de la imagen, el valor de este es 0.

componentes, número de componentes de la imagen, este es fijado a 3 debido a que la imagen esta compuesta por datos rojos, verdes y azules.

ancho, ancho de la imagen. Debe ser 2^n para cualquier n entero.

alto, alto de la imagen. Debe ser 2^m para cualquier m entero.

borde, ancho del borde. Debe estar entre 0 y 1. Para la aplicación es fijado en 0.

formato, formato del dato de los píxeles. Este se toma como GL_RGB, lo que significa que cada elemento es un conjunto de RGB, convertido a punto flotante y ensamblado en formato RGBA anexándole 1.0 para el valor de alpha.

tipo, tipo de datos por los que está hecha la imagen, para el caso de la aplicación se toma como GL_UNSIGNED_BYTE.

pixeles, puntero a los datos de la imagen en memoria. Aquí es donde se encuentra almacenada la información de la imagen. Esta puede ser referenciada por medio de textura->data.

Instrucción

glTexParameteri(

GLenum *destino,*

GLenum *nombrePar,*

GLint *valor*

);

Parámetros

destino, imagen destino, este debe ser GL_TEXTURE_2D.

nombrePar, valor del filtro que se le aplicará a la imagen. Los dos que se han utilizado en la aplicación son: GL_MIN_FILTER y GL_MAG_FILTER.

valor, valor de *nombrePar.* Este se toma como GL_LINEAR.

Si los objetos son de forma esférica

Instrucción

glTexGeni(

    GLenum *coordenada,*

    GLenum *nombrePar,*

    GLint *valor*

);

Parámetros

coordenada, coordenada de una imagen. En la aplicación se ha utilizado GL_S y GL_T

nombrePar, nombre simbólico de la función de generación de coordenadas de la imagen. Este debe se GL_TEXTURE_GEN_MODE.

valor, valor del parámetro de generación de la imagen. Para el caso de los objetos esféricos se utiliza GL_ SPHERE_MAP.

Si los objetos son de forma plana

Instrucción

glTexCoord2f(

GLfloat *s,*

GLfloat *t*

);

Parámetros

s, t, coordenadas de la textura.

Explicación

La función **glTexCoord2f** fija las coordenadas actuales de la imagen que son parte del dato y que es asociada con los vértices del polígono.

Del uso de esta función depende como se verá la imagen en las figuras de tipo plano como los son el cubo y la pirámide, con la función **glTexCoord2f,** se fijan los puntos de la imagen que serán aplicados a la figura.

7. Se procede a dibujar el objeto con las dimensiones especificadas y con los parámetros que se hayan encontrado en el archivo.

4.5. PROGRAMA DOOM

Flujograma general del programa Doom.cpp

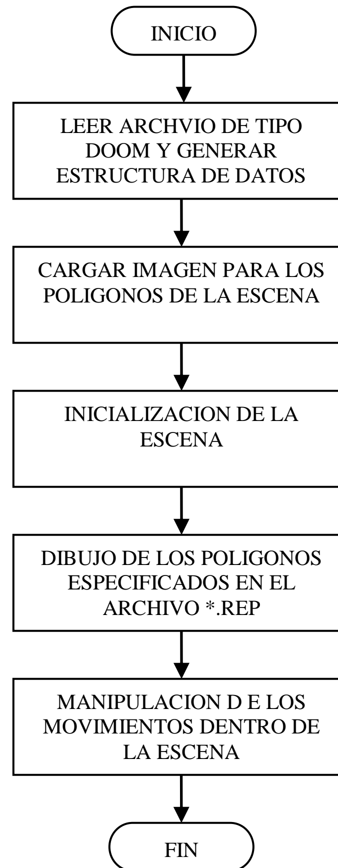


Figura 4.22 Flujograma para el dibujo de escenas DOOM

La generación de la escena para los archivos de simulación de cuartos tipo DOOM es bastante sencilla, la sintaxis de estas instrucciones requiere únicamente de conocimientos de coordenadas en el espacio.

El procedimiento a seguir se muestra a continuación:

1. Se lee el archivo que contiene las instrucciones, que para este caso son las coordenadas de un cuadro o de un triangulo.

   La especificación de la sintaxis de estas instrucciones es la siguiente:

   poligonos *n*

   ‘comentarios en cualquier lugar del archivo

   imagen *ruta_archivo_BMP*

   cuadro

   x1 y1 z1 px1 py1

   x2 y2 z2 px2 py2

   x3 y3 z3 px3 py3

   x4 y4 z4 px4 py4

   triangulo

   x1 y1 z1 px1 py1

   x2 y2 z2 px2 py2

   x3 y3 z3 px3 py3

   Donde:

   n, es el número de polígonos que contendrá la escena. Este valor debe ser especificado en la primera línea válida del archivo y sólo se dibujarán los polígonos especificados por **n,** los demás serán ignorados. Si **n** es mayor que el número de polígonos encontrados en el archivo se genera un error.

ruta_archvio_BMP, es la ruta del archivo de imágenes BMP el cual se aplicará a los polígonos.

(x1, y1, z1) (x2, y2, z2) (x3, y3, z3) (x4, y4, z4), son los vértices del cuadro para el cual se ocupan los cuatro conjuntos de valores, para el triangulo se utilizan únicamente los primeros tres.

Ejemplo:

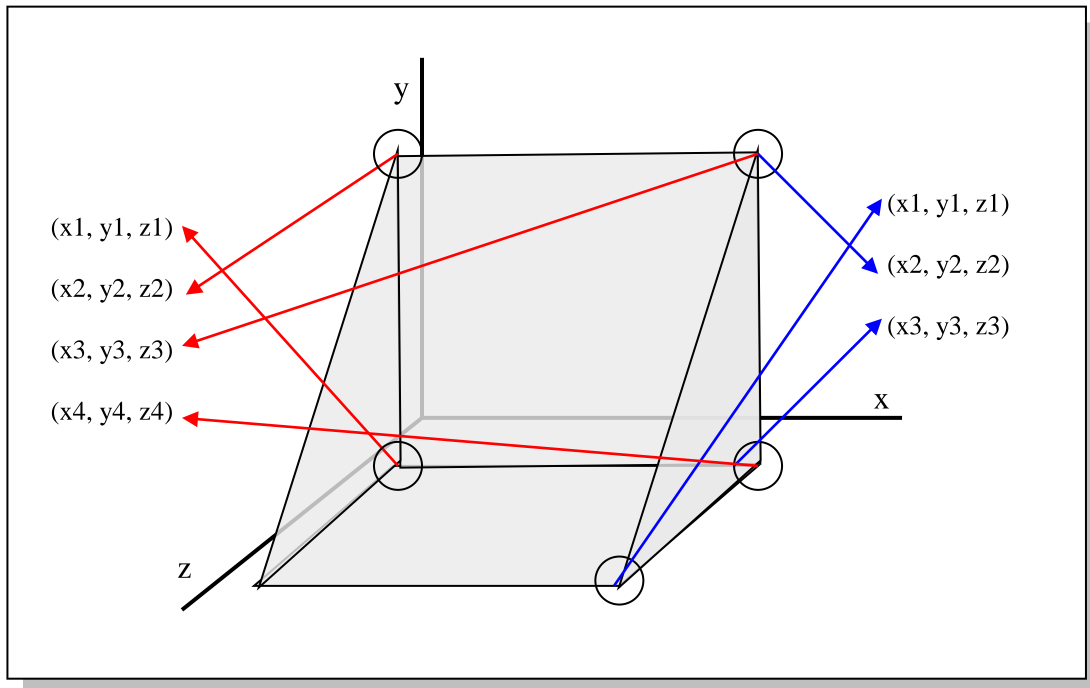


Figura 4.23 Vértices de los polígonos que se pueden utilizar en la simulación DOOM

px1, py1, px2, py2, px3, py3, px4, py4, especifican las coordenadas planas de cómo se aplicará la textura sobre el polígono.

Ejemplo:

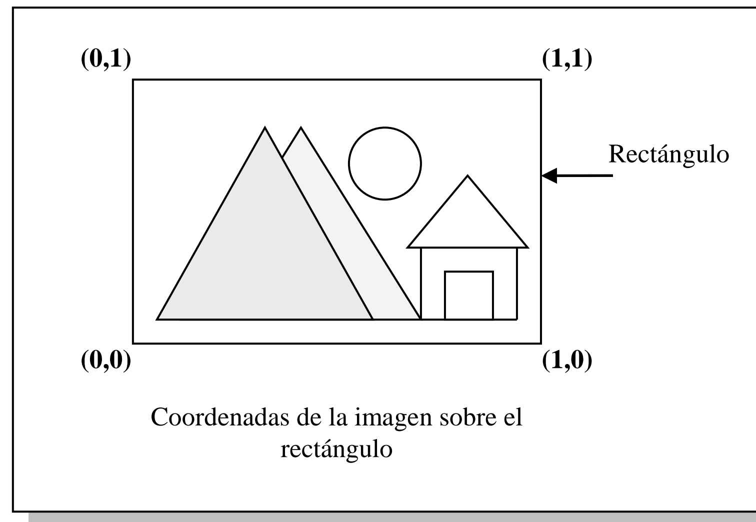


Figura 4.24 Forma de aplicar la textura a los polígonos

cuadro, triangulo, son las palabras que se utilizan para especificar el tipo de polígono a dibujar.

2. Se verifica la existencia del archivo de imagen, si este existe se aplica la imagen a cada uno de los polígonos dinujados en la escena, caso contrario se genera un error.
3. Se inicializa la escena fijando el color de fondo, habilitando la aplicación de texturas y seleccionando el modo de dibujo a utilizar.
4. Se dibuja cada uno de los poligonos según sus vertices y el tipo de estos, ya sea cuadro o triangulo.
5. Una vez creada la escena, los movimientos dentro de ella se pueden realizar mediante el teclado provocando cierto realismo en el modo que se mueve el observador dentro de la escena.

CONCLUSIONES

- Los gráficos tridimensionales pueden ser considerados como parte fundamental de la realidad virtual.

- Por medio de los objetos básicos que dibuja la aplicación se pueden crear escenas complejas dependiendo del nivel de creatividad del usuario.

- El presente trabajo de graduación servirá como base para futuras expansiones de la aplicación y como incentivo para los estudiantes a incursionar en el mundo de los gráficos tridimensionales.

- La aplicación desarrollada servirá como apoyo a las materias de gráficos por computadora, ya sea en la parte teórica ó práctica.

- El estudio de los gráficos tridimensionales abre un gran campo de investigación en el país debido a que este tema no ha sido desarrollado a profundidad y es poca la información bibliográfica que se puede encontrar al respecto en los trabajos de graduación.

RECOMENDACIONES

- Se recomienda dar seguimiento al presente trabajo de graduación debido a que puede considerarse como la base de múltiples proyectos como por ejemplo:
  - Sistemas de Información Geográfica (GIS, por sus siglas en inglés)
  - Programas para crear animaciones o películas.
  - Aplicaciones para simulación.

- Incluir el presente trabajo como fuente bibliográfica de la materia Gráficos por Computadora, así como el uso de la aplicación desarrollada en las prácticas de laboratorio de la misma.

- A los profesionales que utilizan los gráficos en tres dimensiones, como ingenieros civiles, arquitectos, proponer nuevas mejoras a la aplicación que sean aplicables a su área de desempeño.

- Estudiar la posibilidad de abrir múltiples ventanas de dibujo dentro de la aplicación, con el fin de poder copiar y pegar objetos de una ventana a otra.

- Agregar la opción de combinar múltiples archivos para la creación de una sola escena.

- Incluir las colisiones dentro de la escena ya que estas pueden dar mayor sensación de realismo con los objetos que se están representando.

- Expandir el número de objetos que puede dibujar la aplicación.

- Permitir guardar los cambios efectuados en la escena según la posición final de ésta, esto implica guardar coordenadas de rotación y traslación final en la que se encuentran los objetos.

- Mejorar la navegación dentro de las escenas de tipo “representación de objetos tridimensionales”, esto con el fin de provocar una mejor percepción de los objetos que en ella se dibujan.

BIBLIOGRAFIA

- Charles Calvert (1995). "*Teach yourself Windows 95 programming in 21 days*". Second Edition. SAMS Publishing.
- Richard S. Wright Jr., Michael Sweet (1996). "*OpenGL Superbible*". First Edition. Waite Group Press.
- Karla Elizabeth Choto, Sócrates Ulises Escobar, Karina Lissette Quiñónez, Alfredo Omar Rodríguez. "*Análisis de sistemas de tuberías en tres dimensiones*". Tesis Licenciatura. Universidad Centroamericana José Simeón Cañas (UCA), El Salvador.
- María Mercedes Calderón, Yanira del Carmen Figueroa, Ana Sofía Herrera, Marta Elena Ramos. "*Realismo gráfico por medio del trazo de rayos*". Tesis Licenciatura. Universidad Centroamericana José Simeón Cañas (UCA), El Salvador.
- Sitio web de OpenGL, http://www.opengl.org
- Diseño de gráficos y juegos con OpenGL, http://nehe.gamedev.net
- Microsoft Corporation, MSDN Library Visual Studio 6.0

GLOSARIO

- **CAD**

  Computes Aided Design, es el uso de la computadora para asistir al diseño.

- **Instancia**

  Es una copia de una aplicación que se ejecuta en la computadora. La computadora puede ejecutar múltiples instancias de la aplicación.

- **Mensaje**

  Es un paquete de datos utilizado para comunicar información o en un requerimiento. Los mensajes pueden ser pasados entre el Sistema Operativo y una aplicación o bien entre aplicaciones.

- **Modelo**

  Simulación generada por computadora de algo que es real.

- **Tiempo Real**

  Tiempo entre la entrada de los datos y la salida de la solución, donde la respuesta a la entrada es lo suficientemente rápida como para afectar las entradas posteriores.

- **OpenGL**

  Interfaz de software para hardware de gráficos; consiste de una librería de modelado y gráficos en tercera dimensión, rápida y portable.

- **Pixel**

  Punto que representa la menor unidad gráfica de medida en una pantalla. Un píxel depende de la pantalla; es decir, las dimensiones de los elementos de la pantalla varían con la pantalla y la resolución.

- **Proyección Ortográfica**

  Proyección en la que todos los polígonos son dibujados en la pantalla exactamente con las dimensiones relativas especificadas. Utilizada en aplicaciones CAD y planos heliográficos.

- **Proyección Perspectiva**

  Proyección que muestra los objetos y escenas más como son en la vida real que como son presentados en planos heliográficos. Utilizada para simular la sensación de alejamiento o acercamiento.

- **Simulación**

  Proceso mediante el cual se generan condiciones de ensayo aproximadas a las condiciones reales o condiciones de operación reales de algún sistema.

- **Ventana**

  Es un área rectangular sobre la pantalla donde una aplicación despliega información de salida y recibe entradas del usuario. Las ventanas son también los medios por los cuales las aplicaciones envían y reciben mensajes del Sistema Operativo.

ANEXOS

ANEXO No. 1

MANUAL DEL USUARIO

UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERIA

ESCUELA DE COMPUTACION

MANUAL DEL USUARIO

APLICACIÓN PARA LA REPRESENTACION DE OBJETOS EN TRES DIMENSIONES

CIUDADELA DON BOSCO MARZO 2003

INDICE

1. Descripción de la aplicación 1
2. Requerimientos de Hardware y Software 2
3. Instrucciones de instalación de la aplicación 3
4. Creación de archivos de tipo objeto 6
5. Creación de archivos de simulación DOOM 14
6. Interfaz de la aplicación 16
7. Teclas de acceso rápido de la aplicación 21
8. Guía paso a paso para la creación de escenas 23
   - 8.1. Pasos para la creación de escenas del tipo “Objetos Tridimensionales” 23
   - 8.2. Pasos para la creación de escenas del tipo “Cuartos DOOM” 31

I. DESCRIPCION DE LA APLICACION

La representación de objetos en tres dimensiones como parte de la realidad virtual nos permite tener una mejor apreciación de cada uno de los componentes de una escena, y por medio de una aplicación con fines didácticos para dibujar este tipo de objetos tridimensionales se puede conocer los métodos aplicados para lograr dicho fin.

La aplicación además permite la modificación de cada uno de los objetos que se encuentran en escena y los cambios son aplicados instantáneamente, dando la opción de guardar esos cambios realizados.

II. REQUERIMIENTOS DE HARDWARE Y SOFTWARE

Requerimientos mínimos de Hardware

- Procesador de 266 MHz
- 128 MB de Memoria RAM
- 5 MB espacio en disco duro
- Tarjeta de video de 4 MB

Requerimientos mínimos de Software

- Sistema Operativo Windows 98 o posterior

III. INSTRUCCIONES DE INSTALACION DE LA APLICACION

1. En el CD de instalación ubique y ejecute el archivo instalar.exe.
2. Se desplegará el cuadro de diálogo de bienvenida de la instalación de la aplicación, el cual pedirá que se cierren todas las aplicaciones que se estén ejecutando.

Figura 1 Cuadro de bienvenida a la instalación

3. Cierre todas las aplicaciones y presione el botón de Continuar.
4. Se presenta un cuadro de diálogo en el cual se especifica la ruta donde serán guardados los archivos de instalación del programa. Por defecto la carpeta de instalación estará ubicada bajo la carpeta de Archivos de Programa. Presione el botón **Siguiente** para continuar la instalación.

Figura 2 Especificar directorio

5. Presione **Si** para crear el nuevo directorio donde se instalará la aplicación.

Figura 3 Confirmar la creación del directorio

6. Automáticamente inicia el proceso de copia de los archivos. Ver figura 4.
7. Se despliega el cuadro de diálogo que informa que la aplicación fue instalada satisfactoriamente. Ver figura 5.

Figura 4 Proceso de copia de los archivos

Figura 5 Finalización de la instalación

Una vez terminado el proceso de instalación de la aplicación esta puede ser ejecutada mediante el acceso directo creado en el Menú Inicio de Windows bajo "Programas".

IV. CREACION DE ARCHIVOS DE TIPO OBJETO

Los archivos utilizados para la representación de objetos son de tipo texto, lo que quiere decir que pueden ser creados en cualquier editor, como por ejemplo en el Bloc de Notas de Windows. La sintaxis a seguir es la siguiente:

/

‘Comentarios únicamente al inicio de cada instrucción

intruccion1

{

parámetro1(*valores permitidos* **);**

⋮

parámetroN(*valores permitidos* **)**

}

⋮

‘Comentario

instrucciónN

{

parámetro1(*valores permitidos* **);**

⋮

parámetroN(*valores permitidos* **)**

}

/

Donde:

/ Indica el inicio y fin del archivo.

{} Indica el inicio y fin de los parámetros respectivamente.

() Indica el inicio y fin de los valores permitidos para el parámetro respectivamente.

; Indica la separación de cada uno de los parámetros.

instrucciónN, número máximo de instrucciones permitidas dentro del archivo, las cuales son 50.

parámetroN, número máximo de parámetros permitidos para la instrucción.

valores permitidos, Son los valores que acepta cada uno de los parámetros.

Comentario, puede ir un comentario de no más de 75 caracteres únicamente antes de definir la instrucción, este tiene que ir precedido de la comilla simple (**‘**).

Las instrucciones son divididas en dos tipos, los cuales son:

- Instrucción de tipo ambiente

  Esta instrucción es utilizada para fijar los valores iniciales de la escena a dibujar, su estructura es la siguiente:

  ambiente

  {

  fondo(*r, g, b*);

  escala(*x, y, z*);

  trasladar(*x, y, z*);

  luz_ambiental(*r, g, b, a*);

  luz_difusa(*r, g, b, a*);

luz_especular(*r, g, b, a*);

luz_posicion(*x, y, z, pos*);

sombra(*valor_sombra*)

}

- Instrucciones de tipo Objeto

  Utilizadas para crear los objetos dentro de la escena, estos pueden ser:

  - **esfera{}**
  - **cilindro{}**
  - **disco{}**
  - **cubo{}**
  - **piramide{}**
  - **circulo{}**
  - **triangulo{}**
  - **cuadro{}**
  - **linea{}**
  - **punto{}**

  La estructura para todos objetos es similar, con la diferencia en el parámetro que especifica la dimensión, el cual tiene valores específicos para cada uno de ellos.

  El orden de los parámetros no es estricto, y pueden aparecer o no en una instrucción, con la excepción de los parámetros dimensión e identificador, los cuales son requeridos al momento de dibujar un objeto.

La estructura para los objetos es la siguiente:

NombreObjeto

{

id(*identificador*)**;**

dim(*dimensión_objeto*)**;**

color(*r, g, b*)**;**

trasladar(*x, y, z*)**;**

rotar(*angulo, x, y, z*)**;**

imagen(*ruta_archivo_BMP*)**;**

transparencia(*valor_transparencia*)**;**

modo(*modo_dibujo*)

}

Donde:

NombreObjeto, objeto a dibujar de entre el conjunto de los diez disponibles.

id, cadena de caracteres que especifican como se llamará el objeto dentro de la escena.

dim, valores para la dimensión del objeto. Cada objeto posee un conjunto de valores específicos que definen su dimensión, estos valores pueden ser:

- esfera{ dim(*radio, slices, stacks*) }
    - ❖ *radio,* radio de la esfera.
    - ❖ *slices,* líneas verticales que dibujan la esfera.

    - *stacks,* líneas horizontales que dibujan la esfera.

- cilindro{ dim(*radio_base, radio_altura, altura, slices, stacks*) }
    - *radio_base,* valor del radio de la base
    - *radio_altura,* valor del radio de la altura, para obtener un cono basta con dejar el valor de uno de los dos radios en cero.
    - *altura,* altura del cilindro.
    - *slices,* número de líneas verticales con las que se dibuja el cilindro.
    - *stacks,* número de líneas horizontales con las que se dibuja el cilindro.

- disco{ dim(*radio_interno, radio_externo, slices, lazos*) }
    - *radio_interno,* valor del radio interno.
    - *radio_externo,* valor del radio externo, este tiene que ser mayor al radio interno.
    - *slices,* número de líneas que dibujan el disco.
    - *lazos,* número de lazos que dibujan el disco, estos son los sub-discos que dibujan el disco principal.

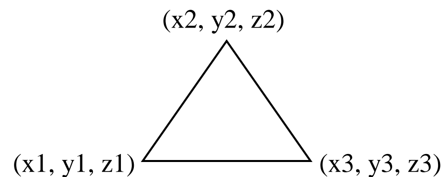
- cubo{ dim(*alto, ancho, profundo*) }
    - *alto,* valor de la altura del cubo, medido con respecto al eje **y**.
    - *ancho,* valor del ancho del cubo, medido con respecto al eje **x**.

- *profundo,* profundidad del cubo, medido con respecto al eje **z**.

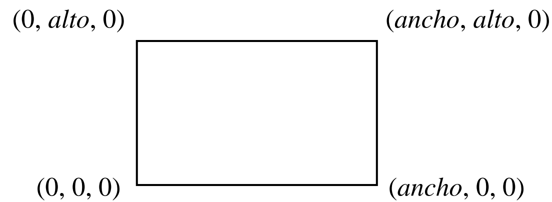
- piramide{ dim(*alto, ancho, profundo*) }
  - *alto,* valor de la altura de la pirámide, medido con respecto al eje **y**.
  - *ancho,* valor del ancho de la pirámide, medido con respecto al eje **x**.
  - *profundo,* profundidad de la pirámide, medido con respecto al eje **z**.

- circulo{ dim(*radio*) }
  - *radio,* radio del círculo a dibujar, medido desde el origen actual de la escena.

- triangulo{ dim(*x1, y1, z1, x2, y2, z2, x3, y3, z3*) }
  - Se dibuja un triángulo con los puntos dados, esto de la siguiente forma:



- cuadro{ dim(*alto, ancho*) }
    - Se dibuja un cuadro rectángulo desde el origen hasta el punto (*alto, ancho*), de la siguiente forma:



- linea{ dim(*x1, y1, z1, x2, y2, z2*) }
    - Se dibuja una línea en el espacio tridimensional a partir del punto (x*1, y1 ,z1)* hasta el punto *(x2, y2, z2).*

- punto{ dim(*x, y, z*) }
    - Se dibuja un punto en el espacio tridimensional en la posición (*x, y, z*).

color, intensidad de color aplicado al objeto en términos de RGB, este parámetro se puede obviar en la instrucción. Por defecto la aplicación dibuja los objetos sin este parámetro con el color (0, 0, 0) que representa el color negro.

trasladar, especifica un vector de traslación de la escena. Por medio de este se mueve el objeto hasta el punto determinado por (x, y, z). Si este parámetro no es encontrado entonces el objeto actual se dibujará sobre el último objeto que se haya dibujado.

rotar, gira el objeto alrededor del vector de rotación especificado por (x, y, z) con el ángulo medido en grados y en contra de las agujas del reloj.

imagen, la aplicación de textura a los objetos da mayor realismo a la escena que se está representando, por medio de este parámetro se puede especificar la ruta de un archivo de imagen BMP el cual es aplicado al objeto. El tamaño de este debe ser en potencia de 2, por ejemplo:

128x128 pixeles

256x128 pixeles

1024x256 pixeles

transparencia, el efecto de sobreponer un objeto sobre y poder visualizar ambos a la vez es logrado por medio de la transparencia. El valor que se especifique tiene que estar entre el rango de 0.0 (completamente transparente) a 1.0 (completamente opaco). El color juega un papel importante al momento de aplicar la transparencia, ya que si este es de intensidad baja el objeto no se visualizará de una manera correcta.

modo, el modo en que son dibujados los objetos puede ser cualquiera de los siguientes:

- lleno, el objeto es dibujado de completamente.
- linea, solamente se dibujan las líneas que definen el polígono.
- punto, sólo son dibujados los puntos de los vértices.

V. CREACION DE ARCHIVOS DE SIMULACION DOOM

Los archivos para la simulación de cuartos del tipo DOOM no son estrictos en cuanto a sintaxis, siempre son de tipo texto, lo cual significa que pueden ser creados en cualquier editor. Su estructura es la siguiente:

poligonos *n*

‘comentarios en cualquier lugar del archivo

imagen *ruta_archivo_BMP*

cuadro

x1 y1 z1 px1 py1

x2 y2 z2 px2 py2

x3 y3 z3 px3 py3

x4 y4 z4 px4 py4

triangulo

x1 y1 z1 px1 py1

x2 y2 z2 px2 py2

x3 y3 z3 px3 py3

Donde:

n, es el número de polígonos que contendrá la escena. Este valor debe ser especificado en la primera línea válida del archivo y sólo se dibujarán los polígonos especificados por **n,** los demás serán ignorados. Si **n** es mayor que el número de polígonos encontrados en el archivo se genera un error.

ruta_archivo_BMP, es la ruta del archivo de imágenes BMP el cual se aplicará a los polígonos.

(x1, y1, z1) (x2, y2, z2) (x3, y3, z3) (x4, y4, z4), son los vértices del cuadro para el cual se ocupan los cuatro conjuntos de valores, para el triángulo se utilizan únicamente los primeros tres.

px1, py1, px2, py2, px3, py3, px4, py4, especifican las coordenadas planas de cómo se aplicará la textura sobre el polígono.

VI. INTERFAZ DE LA APLICACION

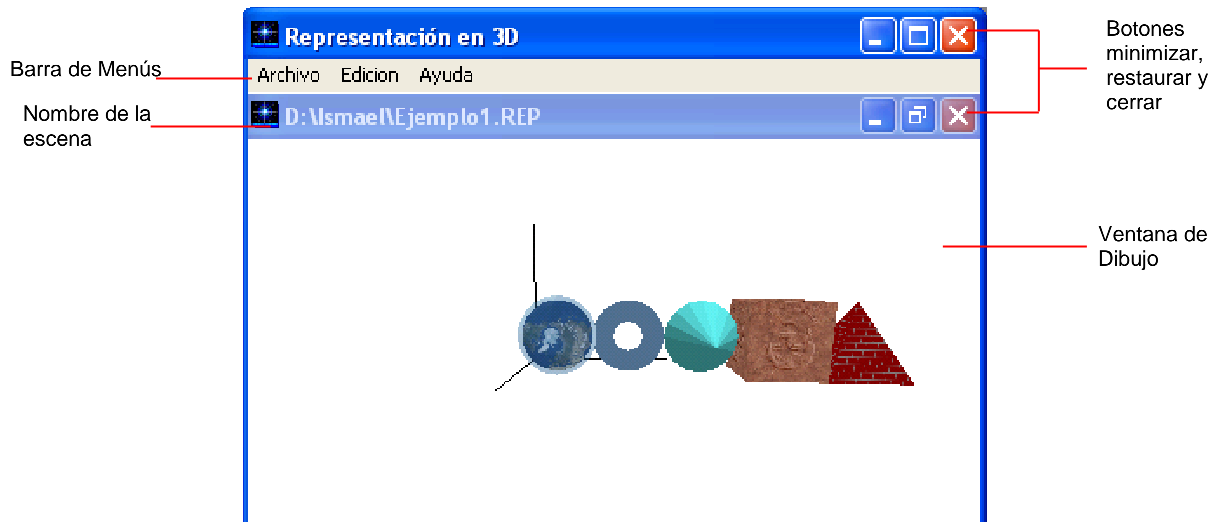


Figura 6 Pantalla principal de la aplicación

Elementos de la interfaz

- Ventana de dibujo en la cual se desplegarán los objetos que se encintraron en el archivo cargado por la aplicación.
- Nombre de la escena, que es el nombre del archivo que se encuentra dibujado en la escena.
- Botones de control de las ventanas, estarán habilitados los botones de Minimizar, Restaurar y Cerrar.
- Barra de Menús, que permite acceder a las opciones para manipular los objetos y la escena en general. Sus opciones son las siguientes:

Menú Archivo

- **Abrir:** Muestra el cuadro de dialogo Abrir Archivo de Windows (Ver figura 7). Este nos permite abrir dos tipos de archivos, estos son:
  - Archivos de tipo Representación de Objetos 3D cuya extensión es *.REP.
  - Archivos de tipo Simulación de Cuartos tipo DOOM con extensión *.SIM.

  Estos archivos son de tipo texto y son los únicos que pueden ser cargados por la aplicación.

- **Guardar:** Por medio de esta opción se puede almacenar los cambios que se efectúen ya sea a los objetos o a la escena que se está representando, su estado inicial es inactivo, cambia de estado cuando se carga un archivo y se le hacen modificaciones, al cerrar la escena vuelve a su estado de inactivo.

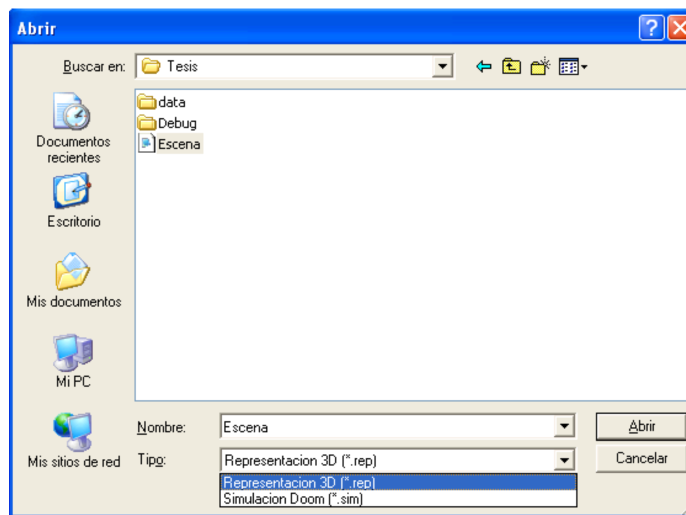


Figura 7 Cuadro de dialogo abrir archivo

- **Cerrar:** La escena que ya se ha cargado en la aplicación puede ser cerrada para poder abrir otro archivo, ya que sólo se permite abrir un archivo a la vez. El estado inicial de esta opción es inactivo, cambia de estado cuando se abre un archivo y es dibujada la escena.

- **Salir:** Cierra la escena con la que se está trabajando actualmente y cierra toda la aplicación, su estado es siempre activo.

Menú Edición

- **Escena:** Por medio de esta opción se puede cambiar las propiedades de la escena que se dibuje, esta nos lleva a un cuadro de dialogo que es estándar para todas las propiedades (Ver figura 8, cuadro de diálogo de propiedades) y permite hacer modificaciones a los siguientes parámetros:
    - Color de Fondo
    - Traslación de la escena
    - Escala con la que se dibuja la escena

- **Iluminación:** La iluminación de la escena es parte fundamental para una buena visualización de los objetos que en ella se están representando.

  Por medio de las siguientes sub-opciones se puede cambiar las propiedades de ésta.

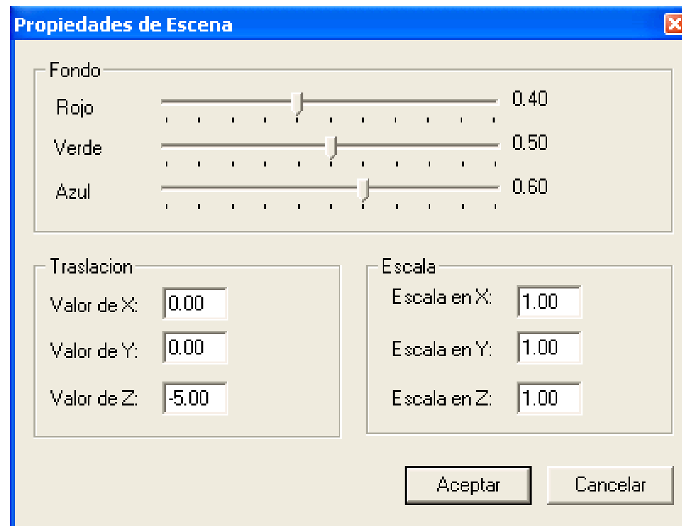


Figura 8 Cuadro de dialogo de propiedades

- **Ambiental:** Abre un cuadro de dialogo que permite cambiar las propiedades de la iluminación ambiental, estos cambios se verán reflejados al momento de presionar el botón aceptar del cuadro de dialogo.

  - **Difusa:** Similar a la opción anterior, sólo que este presenta los valores para la iluminación difusa.

  - **Especular:** Por medio de esta opción se puede cambiar los valores para la iluminación especular.

  - **Posición:** La posición en la que se encuentra la luz es algo muy importante al dibujar una escena, pues de esto depende mucho la

correcta visualización de los objetos. Estos valores pueden ser modificados mediante el cuadro de dialogo Posición de luz.

- **Objetos:** Las propiedades que tenga el objeto pueden ser cambiadas mediante esta opción, estos cambios se verán reflejados en la escena al momento de presionar el botón aceptar.

Ayuda

Como un apoyo al usuario se presenta una ventana con información acerca de la aplicación y su modo de operación, así como las instrucciones que puede utilizar para la creación de archivos que la aplicación reconoce para la representación de objetos y escenas.

VII. TECLAS DE ACCESO RAPIDO DE LA APLICACION

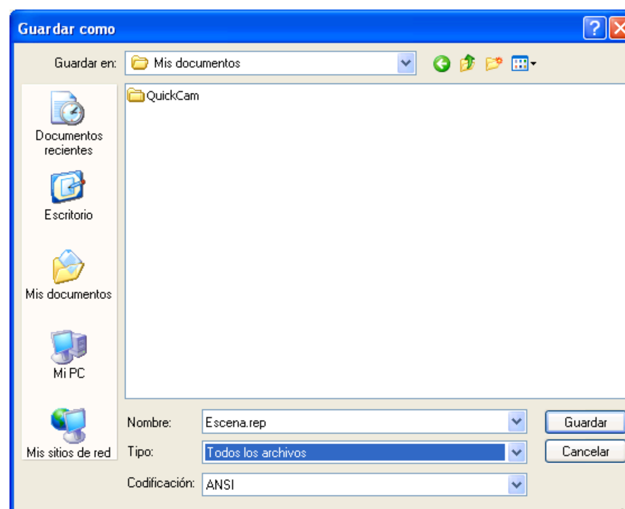
- **Ctrl + A:** Abrir archivo.
- **Ctrl + G:** Guardar Archivo.
- **Ctrl + S:** Propiedades de Escena.
- **Ctrl + M:** Propiedades de Luz Ambiental.
- **Ctrl + D:** Propiedades de Luz Difusa.
- **Ctrl + E:** Propiedades de Luz Especular.
- **Ctrl + P:** Propiedades de Posición de la Luz.
- **Ctrl + O:** Propiedades de los Objetos en la escena.
- **Alt + F4:** Cierra la aplicación.
- **Flecha Izquierda (←):** En el caso de los archivos *.rep, rota los objetos dentro de la escena en torno al eje **Y** en sentido de las agujas del reloj. En el caso de los archivos de simulación DOOM, rota la cámara hacia la izquierda.
- **Flecha Derecha (→):** En el caso de los archivos *.rep, rota los objetos dentro de la escena en torno al eje **Y** en sentido contrario de las agujas del reloj. En el caso de los archivos de simulación DOOM, rota la cámara hacia la derecha.
- **Flecha Arriba (↑):** En el caso de los archivos *.rep, rota los objetos dentro de la escena en torno al eje **X** en sentido de las agujas del reloj. En el caso de los archivos de simulación DOOM, simula el movimiento de la cámara hacia adelante.

- **Flecha Abajo (↓):** En el caso de los archivos *.rep, rota los objetos dentro de la escena en torno al eje **X** en sentido contrario de las agujas del reloj. En el caso de los archivos de simulación DOOM, simula el movimiento de la cámara hacia atrás.
- **Signo menos (-) teclado numérico:** Traslada los objetos en relación al eje Z negativo.
- **Signo más (+) teclado numérico:** Traslada los objetos en relación al eje Z positivo.

VIII. GUIA PASO A PASO PARA LA CREACION DE ESCENAS

8.1. Pasos para la creación de escenas del tipo "Objetos Tridimensionales"

1. Abra el bloc de notas de Windows ubicado en el menú Inicio/Accesorios/Bloc de Notas.
2. En el menú Archivo seleccione la opción Guardar como. Se desplegará el siguiente cuadro de diálogo.



3. En el cuadro de texto desplegable llamado “Tipo” seleccione “Todos los archivos”, esto con el fin de guardar nuestro archivo con la extensión *.rep.
4. En el cuadro de texto llamado “Nombre”, digite el nombre del archivo, a continuación del nombre digite la extensión *.rep, por ejemplo: **Escena1.rep,** donde “Escena1” es el nombre del archivo seguido del punto y la extensión “rep”.
5. Presione el botón “Guardar” para poder comenzar a digitar las instrucciones.
6. Para iniciar el bloque de instrucciones se debe digitar la pleca “***/***”, la cual indica el inicio del archivo que será leído por la aplicación.
7. La primera instrucción que debe aparecer en el archivo es la de tipo ambiente, ésta es digitada de la siguiente forma:

Instrucción o Parámetro		Explicación
‘comentario	→	Comentarios para las instrucciones, permitidos en todo el archivo con la restricción que deben aparecer antes de declarar una instrucción y no debe exceder los 75 caracteres.
ambiente{	→	Nombre de la instrucción seguida por la llave de inicio.
fondo(r, v, a);	→	Fondo de la escena, definido por los colores rojo, verde y azul con un valor en el intervalo de 0 a 1 para cada uno de ellos.

trasladar(x, y, z); → Traslación de la escena en general, esto significa mover el punto de origen (0, 0, 0) de la escena, al punto (x, y, z).

escala(x, y, z); → Valor de la escala par cada uno de los componentes en x, y, z.

luz_ambiental(r, v, a ,A); → Valores de la luz ambiental. Los primeros tres valores son utilizados para el color de la luz en términos de rojo, verde y azul. El cuarto valor es el factor de incidencia de la luz sobre los objetos.

luz_difusa(r, v, a ,A); → Valores de la luz difusa. Los valores son similares a los de la luz ambiental, con la diferencia en el tipo de luz.

luz_especular(r, v, a ,A); → Valores de la luz especular. Igual que las dos anteriores.

luz_posicion(x, y, z ,pos); → Posición de la luz en la escena. Los primeros tres valores indican un punto del espacio donde estará ubicada la luz. El cuarto valor especifica si la luz estará exactamente en ese punto cuando este toma el valor de uno, caso contrario, los rayos serán paralelos a los objetos.

sombra(valor); → Efecto de sombra en la escena, donde *valor* indica si la sombra es completamente obscura (valor=1) ó si la sombra es transparente (valor=0).

anchoLinea(nPixeles); → Tamaño de las líneas medidas en pixeles, utilizado para dibujar los objetos con el modo de dibujo “linea”.

anchoPunto(nPixeles) → Tamaño de los puntos medidos en pixeles, utilizado para dibujar los objetos con el modo de dibujo “punto”.

} → Fin de la instrucción, esto con la llave de cierre. Observar que al finalizar con el último parámetro no se escribe el punto y coma “**;**” el cual indica la separación de los parámetros.

8. A continuación se declaran las instrucciones de tipo objeto, éstas tienen que ser digitadas de la siguiente forma:

Instrucción o Parámetro		**Explicación**
‘comentario	→	Comentario para la instrucción.
OBJETO{	→	Nombre del objeto a representar, éste puede ser cualquiera de los diez

definidos en la sección 4 de este manual.

dim(*valor_dimension*); → Valor de la dimensión para el objeto, éste dependerá del tipo de objeto que se esté representando.

id(*Nombre_Objeto*); → Cadena de caracteres que especifican un identificador único para el objeto.

color(*r, v, a*); → Color del objeto en términos de rojo, verde y azul, con valores en el intervalo de 0 a 1.

rotar(*angulo, x, y, z*); → Rotación del objeto dentro de la escena, donde, *angulo* es el valor del ángulo de rotación medido en grados y *x, y, z,* es el vector utilizado para rotar el objeto.

trasladar(*x, y, z*); → Traslación del objeto dentro de la escena, donde, (*x, y, z)* especifican las componentes de un vector de traslación medido desde el punto que actualmente es el origen.

modo(*modo_dibujo*); → Este parámetro define la forma en la que el objeto será dibujado, este puede tomar el valor de “lleno”, “linea” y “punto”.

imagen(*ruta_BMP*) → Textura que le será aplicada al objeto, donde *ruta_BMP,* es la ruta física del archivo de mapas de bits.

} → Fin de la instrucción.

9. Para finalizar el bloque de instrucciones también es utilizada la pleca "***/***", la cual indica el fin del archivo.
10. Guarde los cambios efectuados al archivo.
11. Ejecute la aplicación Representación de Objetos Tridimensionales, la cual fue instalada al inicio de este manual en la sección "Instalación de la aplicación".
12. En el menú "Archivo" seleccione la opción "Abrir".
13. En el tipo de archivo a abrir seleccione "Representación 3D (*.rep)".
14. Busque el archivo que creó en el bloc de notas y presione el botón "Abrir".
15. Si el archivo esta libre de errores se dibujará la escena.
16. Por medio del menú "Edicion" ó el menú desplegable al presionar el clic derecho del mouse se puede cambiar las propiedades de la escena y de cada uno de los objetos que en ella se encuentran.
17. Guarde los cambios efectuados en la escena por medio de la opción "Guardar" del menú "Archivo"

}

Ejemplo práctico

/

‘Propiedades de la escena

ambiente{

fondo(0.80, 0.50, 0.50);

escala(0.80, 0.80, 0.80);

trasladar(-0.50, 0.00, -5.00);

luz_ambiental(0.70, 0.70, 0.70, 0.50);

l**uz_difusa**(1.00, 1.00, 1.00, 1.00);

luz_especular(1.00, 1.00, 1.00, 1.00);

luz_posicion(4.00, 8.00, 4.00, 1.00);

anchoLinea(3);

anchoPunto(4)

}

‘Línea que representa el eje X

linea{

dim(0.00, 0.00, 0.00, 2.00, 0.00, 0.00);

id(linea1);

color(0.00, 0.00, 0.00);

transparencia(0.40);

trasladar(0.00, 0.00, 0.00)

}

```
‘Línea que representa el eje Z
linea{ dim(0.00, 0.00, 0.00, 0.00, 0.00, 2.00);
        id(linea2);
        color(0.00, 0.00, 0.00)
}

‘Línea que representa el eje Y
linea{ dim(0.00, 0.00, 0.00, 0.00, 2.00, 0.00);
        id(linea3);
        color(0.00, 0.00, 0.00)
}

‘Esfera con la imagen del globo terrestre
esfera{ dim(0.2500, 32, 32);
        id(esfera1);
        imagen(C:\imagenes\world.bmp);
        color(1.00, 1.00, 1.00);
        modo(lleno);
        transparencia(1.00);
        trasladar(0.20, 0.20, 0.20)
}
‘Fin del archivo
/
```

8.2. Pasos para la creación de escenas del tipo “Cuarto DOOM”

1. Abra el bloc de notas de Windows.
2. En el menú Archivo seleccione la opción Guardar como. Se desplegará el cuadro de diálogo “Guardar como” de Windows.
3. En el tipo de archivo, seleccione “Todos los archivos”.
4. Digite el nombre del archivo con la extensión *.sim, ejemplo: **cuarto.sim**.
5. Presione el botón “Guardar” para comenzar a digitar las instrucciones.
6. La estructura que debe tener el archivo es la siguiente:

poligonos *n*	⟶	Número de polígonos a ser dibujados por la escena.
imagen *ruta_BMP*	⟶	Ruta física del archivo de mapa de bits que será aplicado a todos los polígonos de la escena.
‘comentario	⟶	Comentario, puede ir en cualquier parte del archivo.
cuadro	⟶	Instrucción que indica que el polígono a dibujar es un cuadrilátero.
x1 y1 z1 px1 py1 x2 y2 z2 px2 py2 x3 y3 z3 px3 py3 x4 y4 z4 px4 py4	} ⟶	Vértices del cuadrilátero desde (x1, y1, z1) hasta (x4, y4, z4), con sus respectivas coordenadas planas para la aplicación de la textura.

triangulo → Instrucción que indica que el polígono a dibujar es un triángulo.

x1 y1 z1 px1 py1
x2 y2 z2 px2 py2
x3 y3 z3 px3 py3
→ Vértices del triángulo desde (x1, y1, z1) hasta (x3, y3, z3), con sus respectivas coordenadas planas para la aplicación de la textura.

7. Guarde los cambios efectuados al archivo.
8. Ejecute la aplicación Representación de Objetos Tridimensionales.
9. En el menú “Archivo” seleccione la opción “Abrir”.
10. En el tipo de archivo a abrir seleccione “Simulacion Doom (*.sim)”.
11. Busque el archivo que creó en el bloc de notas y presione el botón “Abrir”.
12. Si el archivo esta libre de errores se dibujará la escena.

Ejemplo práctico:

poligonos 2

imagen c:\imagenes\pared.bmp

'Piso del cuarto

cuadro

-3.0 0.0 -3.0 0.0 6.0

-3.0 0.0 3.0 0.0 0.0

3.0 0.0 3.0 6.0 0.0

3.0 0.0 -3.0 6.0 6.0

'techo del cuarto

triangulo

-3.0 0.5 -3.0 0.0 6.0

-3.0 0.5 3.0 0.0 0.0

3.0 0.5 3.0 6.0 0.0

ANEXO No. 2

MANUAL DEL PROGRAMADOR

UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERIA

ESCUELA DE COMPUTACION

MANUAL DEL PROGRAMADOR

APLICACIÓN PARA LA REPRESENTACION DE

OBJETOS EN TRES DIMENSIONES

CIUDADELA DON BOSCO MARZO 2003

INDICE

9. INTRODUCCION 1

10. Representación de objetos en tres dimensiones 2

11. OpenGL 2

12. Declaración de las funciones de la aplicación 5

 12.1. Funciones del programa principal 6

 12.2. Funciones del programa traductor 10

 12.3. Funciones del programa para dibujar objetos 15

 12.4. Funciones del programa para simular cuartos DOOM 26

 12.5. Funciones para cambiar propiedades 33

I. INTRODUCCION

El presente documento contiene las funciones utilizadas en la programación de la aplicación para la representación de objetos en tres dimensiones, en cada una de las cuales se explica su funcionamiento, su declaración, una breve descripción de los parámetros de entrada de la función así como los valores que son retornados.

Los métodos utilizados por OpenGL para la representación y las instrucciones que hacen posible la aplicación de cada una de las propiedades disponibles para los objetos dentro de la escena.

Posterior a la documentación, se presenta el programa completo de la aplicación, en código escrito para el lenguaje de programación Visual C++ 6.0.

II. REPRESENTACION DE OBJETOS EN TRES DIMENSIONES

Nuestros ojos ven en tres dimensiones; aún cuando no pensamos acerca de ello, el entorno visual en su totalidad está dominado por imágenes en tridimensionales. La visualización tridimensional tiene por objetivo representar objetos y escenas en la computadora, y nos permite crear imágenes, animaciones y escenas interactivas tan realistas o fantásticas como queramos. La representación de objetos en tres dimensiones se ha ido introduciendo en nuestra vida cotidiana quizá sin que lo notemos, desde las ilustraciones de edificios aún no construidos, pasando por los logos de canales de televisión a la hora de las noticias, hasta películas de la calidad visual.

III. OpenGL

Definida como la interfaz de software para hardware de gráficos, el principal propósito de OpenGL es dibujar objetos de dos y tres dimensiones en un buffer. Esos objetos son descritos como secuencias de vértices (los cuales defines objetos geométricos) ó pixeles (los cuales definen imágenes). OpenGL realiza varios pasos de procesamiento sobre estos datos para convertirlos en pixeles y final la imagen final deseada en el buffer.

Los siguientes puntos presentan una manera global de cómo trabaja OpenGL:

- Primitivas y Comandos

  OpenGL dibuja primitivas (puntos, segmentos de líneas o polígonos) sujetas a varios modos seleccionables. Estos modos se pueden controlar independientemente uno del otro. Lo que significa, fijar uno de estos modos no afecta la forma en que los otros modos son fijados.

  Las primitivas son definidas por un grupo de uno o más *vértices.* Un vértice define un punto, el final de una línea o la esquina de un polígono donde dos lados se unen.

  El dato (consistente de coordenadas de vértices, colores, normales, coordenadas de texturas) es asociado a un vértice, y cada vértice con su dato asociado es procesado independientemente, en orden, y de la misma forma.

  Los comandos son también procesados en el orden que son recibidos, aunque puede haber un indeterminado retardo antes que el comando tome efecto. Esto significa que cada primitiva es dibujada completamente antes que cualquier comando subsiguiente tome efecto.

- Control de Gráficos de OpenGL

  OpenGL provee un control de nivel medio sobre las operaciones fundamentales de los gráficos de dos y tres dimensiones. Esto incluye especificación de parámetros tales como matrices de transformación,

coeficientes de las ecuaciones de la iluminación y operadores de actualización de pixeles.

- Modelo de Ejecución

  El modelo de interpretación de los comandos de OpenGL es *cliente/servidor.* El código de la aplicación (cliente) ejecuta comandos, los cuales son interpretados y procesados por OpenGL (servidor). El servidor puede o no operar en la misma computadora del cliente. En este ambiente, OpenGL es transparente en la red. Un servidor puede mantener varios contextos de OpenGL, cada uno de los cuales es encapsulado en el Estado OpenGL. Un cliente puede conectarse a cualquiera de esos contextos. El protocolo de red requerido puede ser implementado argumentando un protocolo ya existente o usando un protocolo independiente. Ningún comando OpenGL esta diseñado para obtener entradas del usuario.

IV. DECLARACION DE LAS FUNCIONES DE LA APLICACION

Diagrama de funcionamiento de la aplicación

La estructura general de la aplicación puede resumirse en los procesos presentados en la figura 1, cada una de estas partes es ejecutada mediante un programa en C++.

La etapa de dibujo puede ser cualquiera de las siguientes:

- Dibujos del tipo "objetos tridimensionales"
- Dibujos del tipo "simulación de cuartos DOOM"

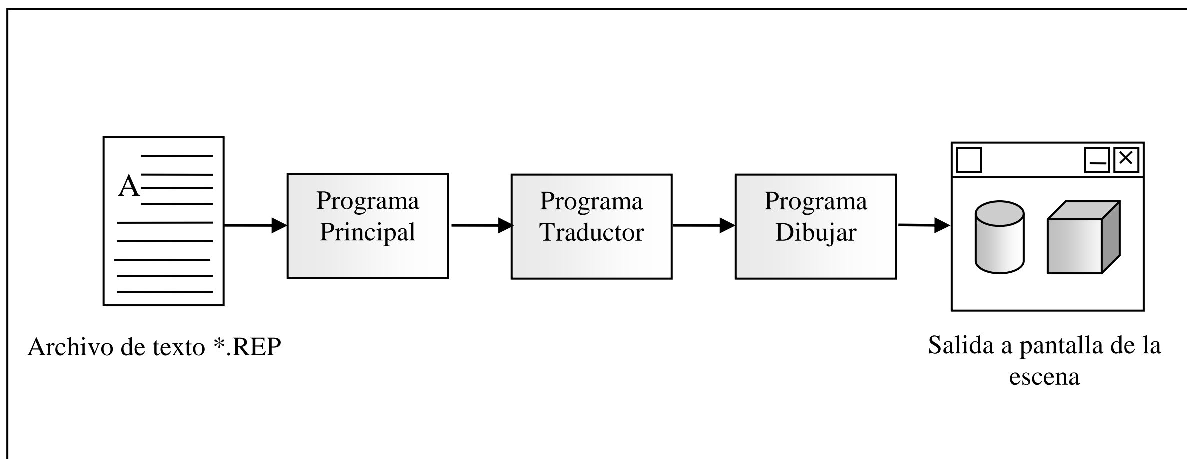


Figura 1 Proceso para dibujar una escena

4.1 Funciones del programa principal

Registrar

Definición:

Esta función es utilizada para registrar la clase de la aplicación el Sistema Operativo.

Declaración:

BOOL Registrar **(HINSTANCE** *hInst***)**

Parámetros:

- **hInst:** Identificador que tomará el manejador (HANDLE) de la instancia que se está registrando.

Valor de Retorno:

- Si la función es exitosa, el valor de retorno es un identificador único de la clase que se está registrando.
- Si la función falla, el valor de retorno es cero.

Crear

Definición:

La función Crear, es utilizada para crear la ventana principal de la aplicación.

Declaración:

BOOL Crear **(HINSTANCE** *hInstance,* **int** *nCmdShow***)**

Parámetros:

- **hInstance:** Identificador de la instancia a la cual pertenece la ventana.

- **nCmdShow:** Estado en el cual se presentará la ventana.

Valor de Retorno:

- Si la función es exitosa, el valor de retorno es un manejador a la nueva ventana que se está creando.
- Si la función falla, el valor de retorno es nulo (NULL).

ProcPpal

Definición:

Función que maneja los procedimientos de la ventana principal.

Declaración:

LRESULT CALLBACK ProcPpal **(HWND** *hWnd***, UINT** *Message***, WPARAM** *wParam***, LPARAM** *lParam***)**

Parámetros:

- **hWnd:** Manejador de la ventana al cual pertenece el procedimiento.
- **Message:** Mensaje que se le está enviando al procedimiento, ya sea comandos del menú, creación de la ventana, finalización de la aplicación, verificación de teclas presionadas.
- **wParam:** Parámetro del mensaje de 32 bits.
- **lParam:** Parámetro del mensaje de 32 bits.

Valor de Retorno:

- El valor retornado es el resultado del mensaje procesado y este depende del mensaje.

ObtenerNomArch

Definición:

Función que muestra el cuadro de diálogo abrir archivo de Windows.

Declaración:

LPSTR ObtenerNomArch **(HWND** *hwnd***, char** * *szFile***, int** *StringSize***)**

Parámetros:

- **hwnd:** Manejador de la ventana principal.
- **szFile:** Nombre del archivo.
- **StringSize:** Especifica el tamaño del nombre del archivo.

Valor de Retorno:

- Si la función es exitosa, se devuelve el nombre del archivo.
- Si la función falla, se devuelve el valor nulo

CargarArchivo

Definición:

Función que selecciona el tipo de archivo a cargar por la aplicación, el cual es obtenido de la función **ObtenerArchivo.** Esta función no devuelve ningún valor, solamente llama a la función respectiva para dibujar ya sea objetos o cuartos tipo DOOM.

Declaración:

void CargarArchivo **(HWND** *hwnd***)**

Parámetros:

- **hwnd:** Manejador de la ventana principal.

Valor de Retorno:

- No retorna valor.

WinMain

Definición:

Función que es llamada por el sistema como punto de entrada inicial en una aplicación basada sobre Win32.

Declaración:

int WINAPI WinMain**(**

HINSTANCE *hInst***,**

HINSTANCE *hPrevInstance***,**

LPSTR *lpszCmdParam***,**

int *nCmdShow* **)**

Parámetros:

- **hInst:** Manejador de la instancia actual.
- **hPrevinstance:** Manejador a la instancia previa.
- **lpszCmdParam:** Puntero a la línea de comando.
- **nCmdShow:** Muestra el estado de la ventana.

Valor de Retorno:

- Si la función es exitosa, termina cuando recibe el mensaje WM_QUIT, debe retornar el valor de salida contenido en el parámetro *wParam.*

- Si la función termina antes que todo el lazo de mensajes, el valor de retorno es cero.

4.2 Funciones del programa traductor

AbrirArchRep

Definición:

Esta función se encarga de abrir los archivos de tipo *.rep, los almacena en cadenas temporales para verificar si las instrucciones son de tipo ambiente o de tipo objeto. No devuelve ningún valor, si todas las instrucciones son correctas llama al programa para dibujar objetos tridimensionales.

Declaración:

void AbrirArchRep**(** **char** **NombreArch,* **HWND** *hwnd***)**

Parámetros:

- **NombreArch:** Nombre del archivo a ser traducido.
- **hwnd:** Manejador de la ventana principal.

Valor de Retorno:

- No retorna valor, si todas las instrucciones son correctas se llama al programa dibujar objetos tridimensionales. Caso contrario se genera un error de traducción de instrucciones.

TradObjeto

Definición:

Función que traduce las instrucciones de tipo objeto, divide cada instrucción de sus parámetros para su posterior evaluación.

Declaración:

bool TradObjeto **(char** * *instruccion,* **char** * *comp,* **HWND** *hwnd***)**

Parámetros:

- **instruccion:** Cadena de la instrucción a ser traducida.
- **comp:** Cadena con la cual se comparará la instrucción.
- **hwnd:** Manejador de la ventana principal.

Valor de Retorno:

- Si todos los parámetros son correctos, se devuelve el valor TRUE.
- Si los parámetros tienen error, se devuelve el valor FALSE.

TradAmbiente

Definición:

Función que traduce las instrucciones de tipo ambiente, divide cada instrucción de sus parámetros para su posterior evaluación.

Declaración:

bool TradAmbiente **(char** * *cadena,* **HWND** *hwnd***)**

Parámetros:

- **cadena:** Cadena de la instrucción la cual debe ser “ambiente{}”.
- **hwnd:** Manejador de la ventana principal.

Valor de Retorno:

- Si todos los parámetros son correctos, se devuelve el valor TRUE.

- Si los parámetros tienen error, se devuelve el valor FALSE.

TradParamObj

Definición:

Función que traduce los parámetros de cada una de las instrucciones de tipo objeto.

Declaración:

bool TradParamObj **(char *** *params,* **char** * *comp,* **HWND** *hwnd***)**

Parámetros:

- **params:** Nombre del parámetro a evaluar.
- **comp:** Utilizado para distinguir entre las dimensiones de cada una de las instrucciones.
- **hwnd:** Manejador de la ventana principal.

Valor de Retorno:

- Si el parámetro a evaluar es correcto, se devuelve el valor TRUE.
- Si el parámetro tiene error, se devuelve el valor FALSE.

TradParamAmb

Definición:

Función que traduce los parámetros de la instrucción de tipo ambiente.

Declaración:

bool TradParamAmb **(char *** *params,* **HWND** *hwnd***)**

Parámetros:

- **params:** Nombre del parámetro a evaluar.
- **hwnd:** Manejador de la ventana principal.

Valor de Retorno:

- Si el parámetro a evaluar es correcto, se devuelve el valor TRUE.
- Si el parámetro tiene error, se devuelve el valor FALSE.

RevDim|Objeto|

Definición:

Donde:

|Objeto| = tres primeras letras del nombre del objeto que se está evaluando. Por ejemplo: **RevDimEsf**

Esta función verifica que la dimensión del objeto este correcta.

Declaración:

bool RevDim|Objeto| **(char** * *parDim,* **HWND** *hwnd***)**

Parámetros:

- **parDim:** Cadena con los valores de la dimensión del objeto.
- **hwnd:** Manejador de la ventana principal.

Valor de Retorno:

- Si todos los valores de la dimensión del objeto son correctos, se devuelve el valor TRUE.
- Caso contrario, se devuelve el valor FALSE.

Rev|Parametro|

Definición:

Donde:

|Parametro| = nombre del parámetro a evaluar. Por ejemplo: **RevColor, RevRotar, RevTrasladar.**

Esta función verifica que todos los valores del parámetro a ser evaluado estén correctos.

Declaración:

bool Rev|Parametro| **(char** * *par|Parametro|,* **HWND** *hwnd***)**

Parámetros:

- **par|Parametro|:** Cadena con los valores del parámetro a ser evaluado.
- **hwnd:** Manejador de la ventana principal.

Valor de Retorno:

- Si todos los valores del parámetro son correctos, se devuelve el valor TRUE.
- Caso contrario, se devuelve el valor FALSE.

TradError

Definición:

Función encargada de concatenar los mensajes de error en la traducción de las instrucciones.

Declaración:

void TradError **(char** * *error,* **HWND** *hwnd,* **int** *b***)**

Parámetros:

- **error:** Cadena con el mensaje de error a ser concatenada.

- **hwnd:** Manejador de la ventana principal.
- **b:** Bandera que indica si el mensaje de error proviene de una instrucción de tipo objeto o de tipo ambiente.

Valor de Retorno:

- No retorna valor, solo modifica una variable global de tipo bool la cual indica si existe error o no.

4.3 Funciones del programa dibujar "Objetos Tridimensionales"

CargarValoresAmb

Definición:

Inicializa una estructura de datos de tipo AmbienteGL para ser utilizada en el programa dibujar.cpp.

Declaración:

void CargarValoresAmb **(AmbienteGL** *amb***)**

Parámetros:

- **amb:** Estructura que contiene los valores originales encontrados en el archivo de texto.

Valor de Retorno:

- No retorna valor.

CargarValoresObj

Definición:

Inicializa un arreglo del tipo estructura de datos ObjetoGL para ser utilizada en el programa dibujar.cpp.

Declaración:

void CargarValoresObj **(ObjetoGL** *e[],* **int** *nObjetos***)**

Parámetros:

- **amb:** Arreglo de tipo ObjetoGL que contiene los valores originales de cada uno de los objetos encontrados en el archivo de texto.
- **nObjetos:** Número de objetos.

Valor de Retorno:

- No retorna valor.

DibujarPpal

Definición:

Función encargada de la creación de la ventana para dibujar objetos tridimensionales.

Declaración:

int WINAPI DibujarPpal **(HWND** *parentWnd,* **char** * *title,* **int** *nObjetos***)**

Parámetros:

- **parentWnd:** Manejador a la ventana principal, la cual es padre de la ventana de dibujo.

- **title:** Nombre de la ventana, el cual es la ruta del archivo de tipo *.rep.
- **nObjetos:** Referencia al número de objetos que se dibujarán en la escena.

Valor de Retorno:

- Si la función es exitosa, termina cuando recibe el mensaje WM_QUIT, debe retornar el valor de salida contenido en el parámetro *wParam.*
- Si la función termina antes que todo el lazo de mensajes, el valor de retorno es cero.

ProcDibujar

Definición:

Función que maneja los procesos de la ventana de dibujo de objetos.

Declaración:

int LRESULT CALLBACK ProcDibujar **(HWND** *hWndDib*, **UINT** *iMsg*, **WPARAM** *wParam*, **LPARAM** *lParam***)**

Parámetros:

- **hWndDib:** Manejador de la ventana al cual pertenece el procedimiento.
- **iMsg:** Mensaje que se le está enviando al procedimiento, ya sea comandos del menú, creación de la ventana, finalización de la aplicación, verificación de teclas presionadas.
- **wParam:** Parámetro del mensaje de 32 bits.
- **lParam:** Parámetro del mensaje de 32 bits.

Valor de Retorno:

- El valor retornado es el resultado del mensaje procesado y este depende del mensaje.

Dibujar_Crear

Definición:

Crea el entorno para la ventana de dibujo.

Declaración:

BOOL Dibujar_Crear **(HWND** *hWndDib***)**

Parámetros:

- **hWndDib:** Manejador de la ventana de dibujo.

Valor de Retorno:

- Si la función es exitosa, se retona valor TRUE.
- Si la función falla, se devuelve el valor FALSE.

Dibujar_Finalizar

Definición:

Cierra la ventana de dibujo.

Declaración:

void Dibujar_Finalizar **(HWND** *hWndDib***)**

Parámetros:

- **hWndDib:** Manejador de la ventana de dibujo.

Valor de Retorno:

- La función no retorna valor.

Dibujar_Ajustar

Definición:

Función encargada de redibujar la escena cuando cambia el tamaño de la ventana de dibujo.

Declaración:

void Dibujar_Ajustar **(HWND** *hWndDib***)**

Parámetros:

- **hWndDib:** Manejador de la ventana de dibujo.

Valor de Retorno:

- La función no retorna valor.

Dibujar_BotonDer

Definición:

Función que crea un menú desplegable para cambiar las propiedades de la escena al presionar el botón derecho del mouse.

Declaración:

void Dibujar_BotonDer **(HWND** *hWndDib,* **BOOL** *fDoubleClick,*
int *x,* **int** *y,* **UINT** *KeyFlags***)**

Parámetros:

- **hWndDib:** Manejador de la ventana de dibujo.
- **fDoubleClick:** Doble clic presionado.

- **x, y:** Posición del mouse en la pantalla.
- **KeyFlags:** Indica si una tecla virtual ha sido presionada.

Valor de Retorno:

- La función no retorna valor.

Dibujar_Comando

Definición:

Función que verifica la ejecución de comandos en la ventana de dibujo referentes al menú desplegable.

Declaración:

void Dibujar_Comando **(HWND** *hWndDib,* **int** *id,*
HWND *hwndCtl,* **UINT** *CodeNotify***)**

Parámetros:

- **hWndDib:** Manejador de la ventana de dibujo.
- **id:** Identificador del menú seleccionado.
- **hwndCtl:** Manejador al control que envía el parámetro cuando el mensaje es enviado de un control, en otro caso el valor es NULL.
- **CodeNotify:** Especifica el código de notificación si el mensaje es de un control. Si el mensaje es de un acelerador, este parámetro es uno. Si el parámetro es de un menú, toma el valor de cero.

Valor de Retorno:

- La función no retorna valor.

Dibujar_Pintar

Definición:

Función que inicializa el ambiente OpenGL para la escena, entre los parámetros están: traslación, rotación, escala.

Declaración:

void Dibujar_Pintar **(HWND** *hWndDib***)**

Parámetros:

- **hWndDib:** Manejador de la ventana de dibujo.

Valor de Retorno:

- La función no retorna valor.

DibujarEscena

Definición:

Dibuja cada uno de los objetos especificados en el archivo.rep con sus propiedades, esta función es llamada cada vez que se redibuja la ventana.

Declaración:

void DibujarEscena **(HWND** *hWndDib,* **BOOL** *shadow***)**

Parámetros:

- **hWndDib:** Manejador de la ventana de dibujo.
- **shadow:** Verifica si la escena se dibuja con sombra o no.

Valor de Retorno:

- La función no retorna valor.

Dibujar_BotonIzqPres

Definición:

Función encargada de verificar si se presiono el botón izquierdo del mouse, por medio de esta se toma el valor de la posición del mouse y se rotan los objetos.

Declaración:

void Dibujar_BotonIzqPres **(HWND** *hWndDib,* **int** *x,* **int** *y***)**

Parámetros:

- **hWndDib:** Manejador de la ventana de dibujo.
- **x, y:** Posición actual del puntero del mouse.

Valor de Retorno:

- La función no retorna valor.

IniciarEscena

Definición:

Los valores de la iluminación y colores de fondo son fijados por esta función, así como la verificación de la existencia de las texturas para cada uno de los objetos.

Declaración:

int IniciarEscena **(HWND** *hWndDib***)**

Parámetros:

- **hWndDib:** Manejador de la ventana de dibujo.

Valor de Retorno:

- Si la función es exitosa, se retorna el valor TRUE.
- Si la función falla, se retorna el valor FALSE.

CargarBMP

Definición:

Verifica la existencia del archivo de imagen BMP en la ruta especificada por FileName.

Declaración:

AUX_RGBImageRec * CargarBMP **(char** * *FileName***)**

Parámetros:

- **FileName:** Ruta del archivo BMP a ser aplicado al objeto.

Valor de Retorno:

- Si la función es exitosa, se retorna el archivo BMP.
- Si la función falla, se retorna valor NULL.

CargarImagen

Definición:

Genera un arreglo de tipo AUX_RGBImageRect en el cual se almacena cada una de las texturas a ser aplicadas a los objetos que poseen esa propiedad.

Declaración:

int CargarImagen**(void)**

Parámetros:

- La función no tiene parámetros.

Valor de Retorno:

- Si la función es exitosa, se retorna el valor TRUE.
- Si la función falla, se retorna el valor FALSE.

ReducUnitario

Definición:

Reduce un vector cualquiera a vector unitario.

Declaración:

void ReducUnitario **(float** *vector[3]***)**

Parámetros:

- **vector[3]:** Arreglo que contiene los componentes del vector a ser reducido a vector unitario.

Valor de Retorno:

- La función no retorna valor, solo altera los componentes de *vector[3]*

CalcNormal

Definición:

Calcula la normal de tres puntos.

Declaración:

void CalcNormal **(float** *v[3][3],* **float** *out[3]***)**

Parámetros:

- **v[3][3]:** Puntos a los cuales se le calculará la normal.
- **out[3]:** Arreglo en el cual se almacenan los componentes del vector normal calculado.

Valor de Retorno:

- La función no retorna valor, solo altera los componentes de *out[3].*

GenerarSombra

Definición:

Genera una matriz de 4x4 en la cual se almacena el plano donde se proyectará la sombra de los objetos.

Declaración:

void GenerarSombra **(GLfloat** *points[3][3],* **GLfloat** *lightPos[3],*
GLfloat *destMat[4][4]* **)**

Parámetros:

- **points[3][3]:** Puntos en el espacio sobre los cuales se proyectará la sombra.
- **lightPos[3]:** Coordenadas de la posición de la luz, con esta se hacen los cálculos para poder simular la sombra.
- **destMat[4][4]:** Matriz de destino donde se almacenan los valores que serán multiplicados por la matriz actual utilizada por OpenGL.

Valor de Retorno:

- La función no retorna valor, solo altera los componentes de *destMat[4][4]*.

GuardarCambios

Definición:

Esta función es la encargada de guardar los cambios que se hayan generado en la escena.

Declaración:

void GuardarCambios **(void)**

Parámetros:

- Esta función no posee parámetros, toma la estructura de datos actual con los cambios que se le hayan efectuado y guarda los cambios en el archivo

Valor de Retorno:

- La función no retorna valor.

4.4 Funciones del programa dibujar "Cuartos del tipo DOOM"

CargarDoom

Definición:

Lee el archivo de tipo *.sim, y las instrucciones son almacenadas en una estructura de datos para la creación de la escena.

Declaración:

int CargarDoom **(void)**

Parámetros:

- No recibe parámetros, el programa inicia en este punto con la lectura del archivo.

Valor de Retorno:

- Si las instrucciones leídas del archivo son correctas se devuelve el valor TRUE.
- Si las instrucciones están incorrectas, se devuelve el valor FALSE.

CargarBMPDoom

Definición:

Verifica la existencia del archivo de imagen BMP en la ruta especificada por NomArch.

Declaración:

AUX_RGBImageRec * CargarBMPDoom **(char** * *NomArch***)**

Parámetros:

- **NomArch:** Ruta del archivo BMP a ser aplicado a los polígonos de la escena.

Valor de Retorno:

- Si la función es exitosa, se retorna el archivo BMP.
- Si la función falla, se retorna valor NULL.

TexturasDoom

Definición:

Esta función es la encargada de generar la textura para los polígonos de la escena.

Declaración:

int TexturasDoom**(void)**

Parámetros:

- La función no tiene parámetros.

Valor de Retorno:

- Si la función es exitosa, se retorna el valor TRUE.
- Si la función falla, se retorna el valor FALSE.

IniciarDoom

Definición:

Por medio de esta función se fijan los valores de la iluminación y colores de fondo, así como la verificación de la existencia de las texturas para los polígonos que se dibujarán.

Declaración:

int IniciarEscena **(void)**

Parámetros:

- La función no recibe parámetros.

Valor de Retorno:

- Si la función es exitosa, se retorna el valor TRUE.
- Si la función falla, se retorna el valor FALSE.

EscenaDoom

Definición:

Los polígonos que han sido definidos en el archivo se dibujan mediante esta función, la cual se encarga de verificar si son triángulos o rectángulos.

Declaración:

int EscenaDoom **(void)**

Parámetros:

- La función no recibe parámetros.

Valor de Retorno:

- Si la función es exitosa, se retorna el valor TRUE.
- Si la función falla, se retorna el valor FALSE.

TerminarDoom

Definición:

Función encargada de cerrar la ventana de los cuartos DOOM y de liberar los recursos que se han utilizado.

Declaración:

void TerminarDoom **(void)**

Parámetros:

- La función no recibe parámetros.

Valor de Retorno:

- La función no retorna valor.

CrearDoom

Definición:

Función encargada de la creación de la ventana para el dibujo de los polígonos que conformaran los cuartos, esta ventana es hija de la ventana principal.

Declaración:

BOOL CrearDoom**(HWND** *hwnd1,* **char** * *title,* **int** *ancho,* **int** *alto,* **int** *bits***)**

Parámetros:

- **hwnd1:** Manejador de la ventana principal, la cual será padre de la ventana de dibujo DOOM.
- **title:** Título de la ventana, el cual es el nombre del archivo que se esta dibujando.
- **ancho:** Ancho de la ventana medido en pixeles.
- **ato:** Alto de la ventana medido en pixeles.
- **bits:** Número de bits que se utilizará en los colores de la escena.

Valor de Retorno:

- Si la función es exitosa, se retorna el valor TRUE.

- Si la función falla, se retorna el valor FALSE.

DoomProc

Definición:

Los valores de la iluminación y colores de fondo son fijados por esta función, así como la verificación de la existencia de las texturas para cada uno de los objetos.

Declaración:

LRESULT CALLBACK DoomProc**(HWND** *hWnd***,**
UINT *uMsg***,**
WPARAM *wParam***,**
LPARAM *lParam***)**

Parámetros:

- **hwnd:** Manejador de la ventana a la cual pertenece el procedimiento.
- **uMsg:** Mensaje que se le está enviando al procedimiento, ya sea creación de la ventana, finalización de la aplicación, verificación de teclas presionadas.
- **wParam:** Parámetro del mensaje de 32 bits.
- **lParam:** Parámetro del mensaje de 32 bits.

Valor de Retorno:

- El valor retornado es el resultado del mensaje procesado y este depende del mensaje.

✦ DoomPpal

Definición:

Esta función es utilizada para abrir el archivo que se obtiene del programa principal así como para verificar las teclas presionadas y moverse dentro de la escena.

Declaración:

int WINAPI DoomPpal **(char** * *ArchDoom,* **HWND** *parentWnd***)**

Parámetros:

- **ArchDoom:** Nombre del archivo a abrir, el cual también será utilizado como nombre de la ventana.
- **parentWnd:** Manejador a la ventana principal, la cual es padre de la ventana de dibujo.

Valor de Retorno:

- Si la función es exitosa, termina cuando recibe el mensaje WM_QUIT, debe retornar el valor de salida contenido en el parámetro *wParam.*
- Si la función termina antes que todo el lazo de mensajes, el valor de retorno es cero.

4.5 Funciones del programa propiedades

➔ InciarPropiedades

Definición:

Carga las estructuras locales de tipo objeto y ambiente utilizadas para efectuar los cambios a los objetos y a la escena respectivamente.

Declaración:

void IniciarPropiedades **(ObjetoGL** *e[ ],* **int** *nObjetos,* **AmbienteGL** *amb1***)**

Parámetros:

- **e[ ]:** Arreglo que contiene la descripción y los parámetros de cada uno de los objetos.
- **nObjetos:** Número de objetos dibujados por la escena.
- **amb1:** Estructura que contiene los parámetros generales de la escena.

Valor de Retorno:

- La función no retorna valor, solamente inicializa estructuras a ser utilizadas en todas las funciones de las propiedades.

➔ Procedimientos de cada una de las propiedades

Definición:

Cada una de estas funciones inicializa los objetos que aparecen en los cuadros de diálogo, así como de aplicar los cambios a la escena.

Declaraciones:

- **BOOL APIENTRY** ProcEscena **(HDLG** *hDlg,* **UINT** *message,* **UINT** *wParam,* **LONG** *lParam***)**
- **BOOL APIENTRY** ProcAmbiental **(HDLG** *hDlg,* **UINT** *message,* **UINT** *wParam,* **LONG** *lParam***)**
- **BOOL APIENTRY** ProcEspecular **(HDLG** *hDlg,* **UINT** *message,* **UINT** *wParam,* **LONG** *lParam***)**
- **BOOL APIENTRY** ProcDifusa **(HDLG** *hDlg,* **UINT** *message,* **UINT** *wParam,* **LONG** *lParam***)**
- **BOOL APIENTRY** ProcEspecular **(HDLG** *hDlg,* **UINT** *message,* **UINT** *wParam,* **LONG** *lParam***)**
- **BOOL APIENTRY** ProcPosicion **(HDLG** *hDlg,* **UINT** *message,* **UINT** *wParam,* **LONG** *lParam***)**
- **BOOL APIENTRY** ProcObjetos **(HDLG** *hDlg,* **UINT** *message,* **UINT** *wParam,* **LONG** *lParam***)**

Parámetros:

- **hDlg:** Manejador del cuadro de diálogo.
- **message:** Mensaje a ser procesado por el cuadro de diálogo.
- **wParam:** Especifica información adicional del mensaje.

- **lParam:** Especifica información adicional del mensaje.

Valor de Retorno:

- Excepto en respuesta al mensaje WM_INITDIALOG el procedimiento retorna un valor diferente de cero si el mensaje es procesado, y cero si este no es procesado.
- En respuesta al mensaje WM_INITDIALOG, el cuadro de diálogo retorna cero si este llama a la función SetFocus para fijar el foco en uno de los controles del cuadro de diálogo. En otro caso, este retorna cero, caso en el cual el sistema fija el foco al primer control del cuadro de diálogo al cual puede ser fijado el foco.