3DEDIT es un programa para la edici¢n de escenas tridimensionales en formato B−rep (boundary representation), es decir, mediante definici¢n de los pol¡gonos que forman el contorno de los diferentes objetos.

El programa permite la creaci¢n de escenas complejas a partir de una serie de figuras b sicas, como esferas, conos, cubos, etc. mediante la edici¢n de las mismas a trav s de cuatro vistas disponibles. Las cuatro vistas est n en modo isom trico, y vienen predefinidas al arrancar el programa como planta, alzado, perfil y tridimensional. No obstante, se pueden modificar las vistas mediante la rotaci¢n y traslaci¢n de las c maras.

Para los que est is interesados en usar los ficheros generados por 3DEDIT, paso a explicar el formato que he utilizado; recordad que una expresi¢n A* indica la repetici¢n de A 0 ¢ m s veces.

escena = objeto*

objeto = N§V rtices V rtice* N§Pol¡gonos Pol¡gono*

v rtice = x y z

pol¡gono = N§Referencias Referencia*

La escena se guarda en el fichero colocando consecutivamente cada uno de los objetos que la forman; cada objeto se compone a su vez de una lista de v rtices y otra de pol¡gonos. La lista de v rtices se especifica mediante un valor (tipo word) que indica el n£mero de v rtices, y las coordenadas de cada v rtice (tres valores tipo real) de forma consecutiva. De igual forma, la lista de pol¡gonos se guarda como el n£mero de pol¡gonos (un word) y las listas de referencias a cada uno de los v rtices. Cada lista de referencias se almacena de la misma forma, es decir, una cabecera indicando el n£mero de referencias (que equivale al n£mero de v rtices del pol¡gono). Cada una de

estas referencias indica el n£mero de v rtice (tipo word) dentro de la lista anteriormente descrita. Este galimat¡as lo puedes resolver f cilmente viendo c¢mo se graba y carga los ficheros en el listado fuente.

En cuanto a posibles ampliaciones, he pensado a¤adir un sistema de c maras que utilicen perspectiva, con lo que se podr¡a usar directamente las escenas para introducirlas en alg£n iluminador. Tambi n se podr¡a implementar la ocultaci¢n de pol¡gonos a nivel general, ya que de momento s¢lamente se ocultan los pol¡gonos posteriores de cada objeto por separado (el tiempo era oro al final del curso pasado).

El lenguaje utilizado ha sido Turbo Pascal (s¡, ya lo se, deber¡a haberlo hecho en C, pero no estoy excesivamente puesto en el mismo, y de todas formas ¨qu tiene casi todo el mundo contra el −maravilloso, intuitivo, simple pero potente− Pascal?), en su versi¢n 7.0 (al compilar con la versi¢n 6.0 da error, ya que ste no incluye algunas instrucciones como continue, break, etc.).

Todo el proyecto lo he realizado mediante un 486/66MHz con 4Mb de RAM (todav¡a no me puedo comprar un Pentium, snif...), con lo que he intentado que el programa funcione medianamente bien en esta m quina. No obstante, el que tenga un bicho m s potente, tanto mejor para l, ya que el modo gr fico usado hace que el redibujado de las escenas, por ejemplo al mover objetos, sea bastante lento (no es culpa m¡a: el 3D Studio usa el mismo formato gr fico, y a la hora de mover objetos lo hace colocando una caja que lo representa para no dar la sensaci¢n de lentitud).

Puedes hacer con este programa lo que te d la gana, con la condici¢n de que incluyas todos los ficheros que se suministran; tambi n puedes extraer parte del programa fuente, si es que logras entender algo (lo siento pero no tuve tiempo de arreglarlo y mucho menos de ponerle comentarios; contacta

conmigo si quieres tener detalles sobre alg£n aspecto y quieres tener el manual t cnico que redact para la pr ctica). Naturalmente, te pido que tengas la cortes¡a de citarme en aqu llos trabajos para los que te apoyes en el presente programa.

Para comenzar a operar con el programa te sugiero que saques una copia del cutre−manual (fichero MANUAL.TXT adjunto), y comiences a trastear por tu cuenta. Tambi n tienes algunos ejemplos ya construidos (ficheros con extensi¢n .ACB). Lamento que el manual no sea m s exhaustivo, pero la revisi¢n de esta pr ctica se hizo "in situ", por lo que ¨para qu escribir un manual de cincuenta folios cuando sab¡a que probablemente el profesor no lo iba a leer?.

En cuanto a posibles ampliaciones, estoy pensando en a¤adir las siguientes caracter¡sticas; cuando tenga tiempo (espero que antes del 2.000) me pondr con ellas.

− Optimizaci¢n del c¢digo, haci ndolo m s r pido; he detectado bastantes puntos que se pueden mejorar aplicando algoritmos distintos.

− Nuevas primitivas, como semiesferas, figuras irregulares, etc. En realidad esto es bastante simple y creo que no me llevar¡a m s que unos pocos d¡as implementarlo.

− Operar con vol£menes; ste es uno de los aspectos m s £tiles en un editor de escenas, pero no est implementado de momento ya que es un punto bastante complicado.

− Animaciones; en realidad es bastante sencillo su realizaci¢n, simplemente manejando una lista de escenas en lugar de una sola. De hecho, ya tengo hechas algunas pruebas respecto a esto y funciona bastante bien; el problema es la memoria, ya que una escena medianamente compleja es bastante voluminosa, y se requiere la utilizaci¢n de memoria expandida (no se −todav¡a−

c¢mo se usa en Pascal).

− Introducci¢n de vistas con distinta perspectiva; en el programa he usado visi¢n isom trica, ya que es la m s sencilla de implementar. No obstante, se pueden introducir perspectiva; de hecho, esto lo ten¡a casi acabado, pero no funcionaba al cien por cien, por lo que decid¡ entregar la pr ctica con una vista m s simple.

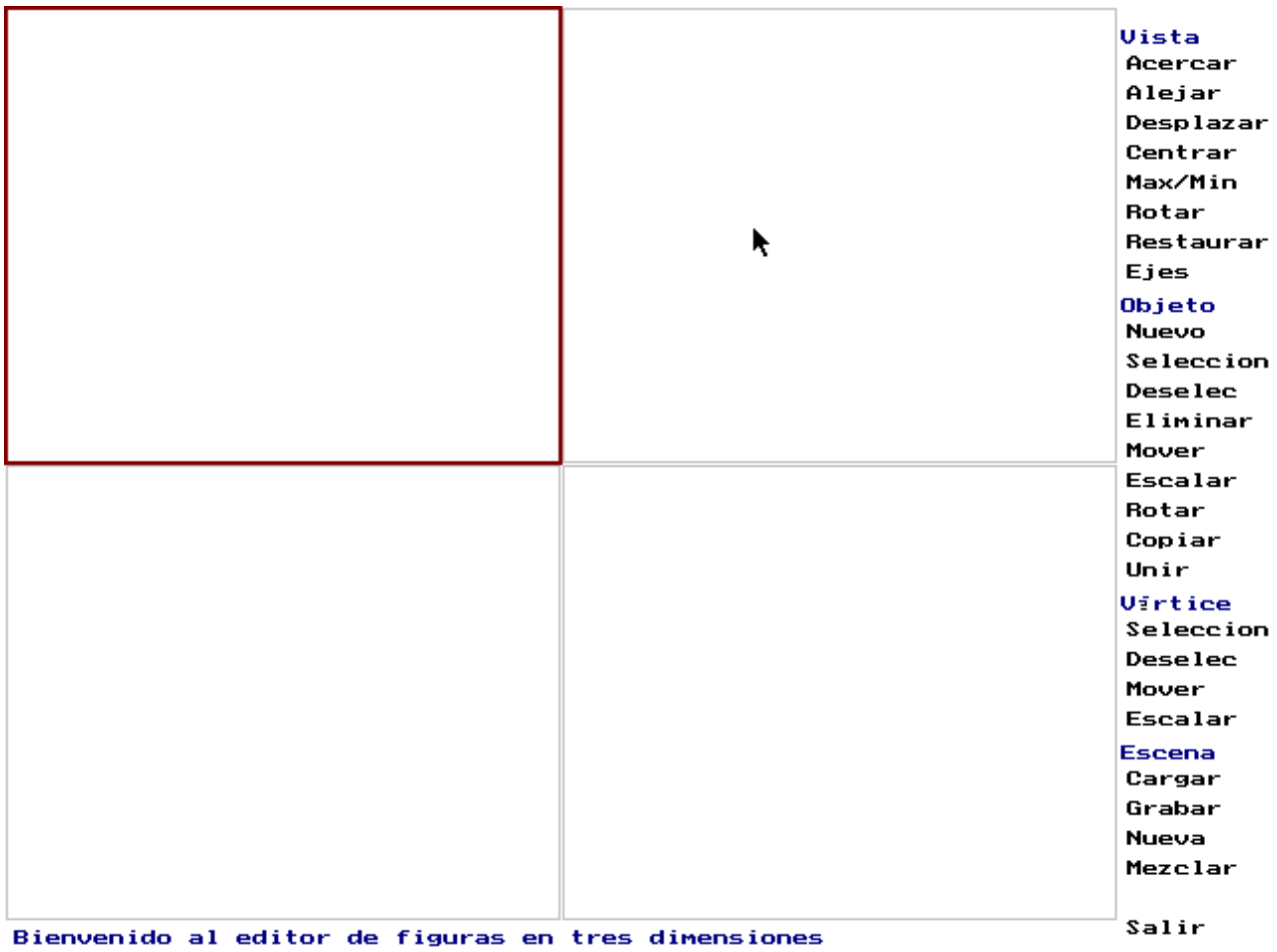
− Ocultaci¢n de pol¡gonos total; es decir, hacer que unos objetos se superpongan a otros. De momento s¢lo se ocultan los posteriores.

Práctica de Infografía

Sistema de diseño 3D

Manual de usuario

El programa arranca con la instrucción *3dedit*, y requiere para su uso de un ratón. Al arrancar, se comprueba si el ratón está efectivamente instalado y en funcionamiento; si hay algún problema, se le pedirá que solucione el problema y pruebe de nuevo. Si todo ha ido bien, aparece la pantalla siguiente:



Podemos distinguir tres zonas en esta pantalla:

− Zona de visualización. Esta zona está dividida en cuatro cuadrantes, cada una de las cuales representan una de las posibles vistas que se pueden presentar simultáneamente. Inicialmente, el primer cuadrante representa la planta, el segundo el alzado, el tercero el perfil, y el cuarto una vista tridimensional en proyección isométrica. Esquemáticamente, la configuración inicial de los cuadrantes sería la siguiente:

1 Y 2 Z

X Y

2 Z 3 Z

Y

X

X

− Zona de mensajes. Está situada en la parte inferior de la pantalla; en esta zona irán apareciendo todos los mensajes que el programa emita, tanto los de ayuda como los que requieran algún tipo de entrada por parte del usuario.

− Zona de menús. En la parte derecha de la pantalla tenemos una serie de opciones divididas en varios

submenús según a qué entidad estén referidos (vista, objeto, vértice ó escena).

El manejo del programa se realiza enteramente mediante el ratón (excepto las entradas que requieran datos numéricos o alfanuméricos, lógicamente), utilizándose los botones izquierdo y derecho.

En este momento, la vista seleccionada es la superior izquierda (primer cuadrante); esto se indica porque el marco que rodea a dicho cuadrante está en color rojo. Esta vista será sobre la que se ejecuten todos los comandos; si deseamos seleccionar cualquier otra, bastará con pulsar sobre la vista correspondiente.

Para ejecutar uno de los comandos del menú, basta con desplazar el ratón sobre la opción correspondiente, y pulsar uno de los botones. En general, el botón izquierdo se utiliza para ejecutar un comando sobre una única entidad, mientras que el derecho indica la ejecución múltiple, cuando esta opción esté disponible.

A continuación veamos el propósito de cada uno de los comandos.

Menu *Vista*

Acercar. Realiza un acercamiento del punto de vista hacia la escena, con lo que se produce el efecto de un *zoom* de ampliación de la escena.

Alejar. Aleja la posición del observador de la escena, por lo que la escena se hará más pequeña.

Desplazar. Permite desplazar el punto de observación paralelamente al plano de proyección. Para ello, tras pulsar sobre la opción, podremos realizar dicho desplazamiento simplemente moviendo el raton. Para terminar, simplemente hay que pulsar un botón.

Centrar. Ajusta la escena de forma que se visualice toda ella en la vista determinada.

Max/Min. Como antes comentamos, inicialmente aparecen cuatro cuadrantes en pantalla; esto puede ser modificado con este comando, haciendo que la vista seleccionada ocupe toda la zona de visualización, o haciendo que vuelva a su modo inicial.

Rotar. Realiza una rotación del punto de observación alrededor de la escena, manteniendo la dirección de observación. Desplazando el ratón horizontalmente, giramos respecto del eje Z, mientras que si el movimiento es vertical, la rotación se hace inclinando más o menos el plano de proyección hacia la posición del observador.

Restaurar. Restablece las características iniciales de las vistas, restaurando la posición de la cámara, los ejes, etc. Esto es útil cuando queramos regresar al esquema tradicional de planta, perfil y alzado.

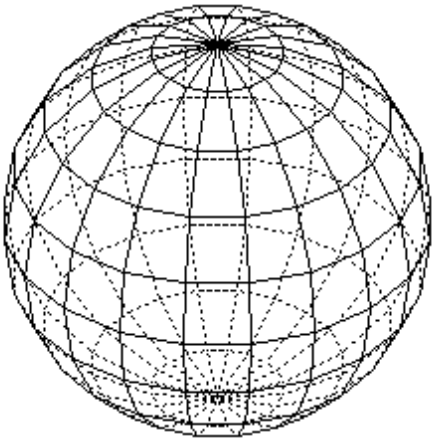
Ejes. Hace que sean visibles u ocultos los ejes X, Y y Z. Estos ejes saldrán en línea discontinua y en color amarillo.

Las opciones *Acercar, Alejar, Centrar, Restaurar* y *Ejes* admiten actuación múltiple, es decir, si ejecutamos uno de estos comandos pulsando el botón derecho del ratón, la acción se realizará sobre las cuatro vistas.

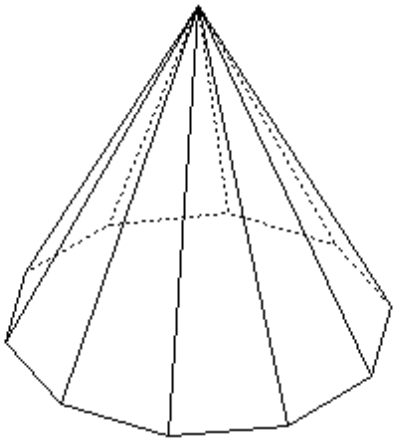
Menú *Objeto*

Nuevo. Permite la creación de un nuevo objeto, siendo éste de uno de los siguientes tipos predefinidos: esfera, pirámide, tronco, toroide y caja; se solicita al usuario que elija uno de estos tipos. Todos los objetos se crearán colocándolo sobre el plano de proyección. Veamos cómo se procede para crear un objeto de una clase:

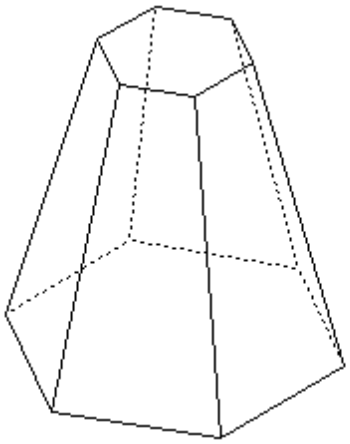
1. Esfera. El usuario debe introducir por teclado el número de vértices que tendrá cada uno de los meridianos de la esfera; posteriormente, se debe dar gráficamente el tamaño del radio de la esfera. El objeto creado consiste en una malla con forma esférica compuesta de polígonos de cuatro aristas en general y de tres aristas para aquéllos a los que pertenece cada uno de los polos. Por ejemplo, el aspecto tridimensional de una esfera de 20 vértices por circunferencia es la siguiente:



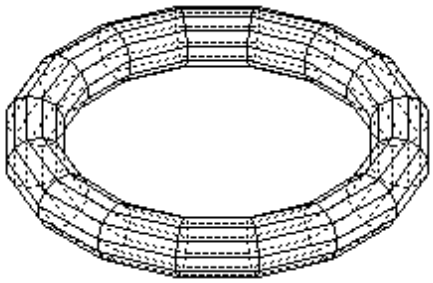
2. Pirámide. En este caso, el valor numérico que se pide al usuario es el número de vértices que componen la base, introduciendo el radio gráficamente. El objeto creado es una pirámide que tendrá por defecto una altura igual al diámetro de la base. El aspecto para el caso de 10 vértices sería el siguiente:



3. Tronco. Un tronco es un objeto resultado de eliminar de una pirámide la parte superior. Así pues, además de el número de vértices de las bases superior e inferior (lógicamente será el mismo), se deben determinar gráficamente sus respectivos radios. Veamos cómo queda un objeto de este tipo con 6 vértices en cada base:



4. Toroide. Un toroide es el resultado de realizar el recorrido de un polígono sobre el camino marcado por otro; así pues, debemos determinar la forma de ambos polígonos. Primero se introduce el número de vértices del polígono que marcará el camino, y posteriormente el del polígono que forma la sección. Gráficamente determinamos los radios máximo y mínimo del toroide, de tal forma que el diámetro del polígono de la sección será la diferencia entre ambos radios. Veamos un ejemplo de toroide con número de vértices igual a 16 para el exterior y 12 para la sección:



5. Cubo. La forma de este tipo de objetos es la obvia; el único parámetro necesario es el tamaño del mismo, que se introducirá gráficamente.

Al crear un objeto, si aumentamos el número de vértices éste se parecerá más a una figura perfecta. Así, la esfera se aproximará a una esfera perfecta, la pirámide a un cono, el tronco a un cono truncado y el toroide a un anillo circular con sección también circular. No obstante, este número no puede ser todo lo grande que se desee, por cuestiones de espacio en memoria. Para delimitar este número se realiza un cálculo aproximado de la memoria que precisará el objeto, y si no hay bastante disponible se da un mensaje por pantalla para que se disminuya el valor.

Seleccion. Permite la selección de uno o varios objetos; en caso de ser selección simple (botón izquierdo), cada vez que pulsemos el botón izquierdo seleccionaremos el objeto más cercano, entendiendo por éste aquél que contiene al vértice más cercano al puntero; para terminar se pulsa el botón derecho. Si hemos optado por selección múltiple (botón derecho), ésta se hace creando una ventana, y serán seleccionados todos los objetos que tengan algún vértice dentro de la misma. Todos los objetos seleccionados aparecerán en pantalla en color verde hasta que sea deseleccionado.

Deselec. Este comando es el opuesto al anterior, y funciona de forma completamente idéntica.

Eliminar. Con este comando, en modo sencillo iremos indicando con el botón izquierdo los objetos que se van eliminando, finalizando con el derecho. En modo múltiple, serán eliminados directamente todos los

objetos que estén seleccionados.

Mover. Si estamos en modo sencillo, elegimos los objetos a mover y los desplazamos con el ratón, depositándolos con el botón izquierdo; para terminar, se pulsa el botón derecho. En modo múltiple se moverán de forma conjunta los objetos seleccionados. Señalar que todos los movimientos se realizan lógicamente de forma paralela al plano de proyección.

Escalar. Se utiliza esta opción para cambiar el tamaño de un objeto o conjunto de objetos; este escalado se hace respecto del punto central de la caja que encierra al objeto u objetos correspondientes.

Rotar. Realiza una rotación de un objeto u objetos sobre el plano de proyección; al igual que el caso anterior, el punto respecto del cual se rota es el centro de la caja que encierra al conjunto.

Copiar. Usamos esta opción para realizar un duplicado de un objeto u objetos, permitiendo que la copia creada sea trasladada de lugar para evitar que se solape con el original.

Unir. Este comando se utiliza cuando deseamos que una serie de objetos pasen a ser únicamente uno y sean tratados como tal. En el caso del modo sencillo, se supone que queremos unir dos objetos, con lo cual los indicamos mediante el ratón y se ejecuta la acción; en caso múltiple, la unión se realiza entre todos los objetos seleccionados.

Menú *Vértice*

Seleccion. Al igual que en el caso de los objetos, podemos seleccionar vértices individualmente o mediante una ventana; en el primer caso, el vértice seleccionado es el más cercano al puntero del ratón, y en el segundo se seleccionan todos aquellos que caigan dentro de dicha ventana. Un vértice seleccionado se distingue porque aparece un pequeño círculo rojo alrededor de él.

Deselec. Realiza la acción contraria a la anterior, de forma completamente análoga.

Mover. Permite el movimiento de un vértice ó conjunto de vértices, de idéntica forma a la vista hasta ahora: bien eligiendo uno de forma individual o bien moviendo todos los seleccionados conjuntamente.

Escala. Realiza un escalado de los vértices de forma similar a la de un objeto, es decir, tomando como punto de referencia el centro del conjunto de vértices. Esta opción, como es lógico, se realiza sobre el conjunto de vértices seleccionados.

Menú *Escena*

Cargar. Lee un fichero de disco y carga la escena en memoria. Para ello, se solicita el nombre del fichero a cargar y se saca por pantalla la escena. Respecto al nombre del fichero, éste puede tener cualquier nombre válido para M.S.−D.O.S.; se puede especificar cualquier extensión e incluso la ruta del fichero. En el manual técnico se explica en profundidad el formato de los ficheros.

Grabar. Salva la escena actual a disco, pidiendo por teclado el nombre destino; en caso de que el fichero ya exista, se pide confirmación para sobreescribirlo.

Nueva. Elimina de memoria la escena devolviendo el programa al estado inicial, pidiendo previamente confirmación por teclado.

Mezclar. Realiza la misma función que *cargar*, con la diferencia de que no elimina la escena actual, sino que la mantiene y une ambas escenas. Esta opción es muy útil cuando queremos diseñar una escena

descomponiéndola en varias subescenas más pequeñas.

Cuando entramos en una opción que permite movimiento, rotación, etc. mediante el ratón, existe la posibilidad de fijar el movimiento del mismo. Esto se puede variar pulsando la tecla *Ctrl*; cada vez que se pulse, aparecerá en el centro del cuadrante un pequeño icono que indica el modo actual de funcionamiento. Veamos el significado de cada uno de ellos:

Movimiento libre

Movimiento vertical. El movimiento horizontal

está bloqueado.

Movimiento horizontal. El movimiento vertical

está bloqueado.

Movimiento en diagonal. El movimiento se hace

igual en horizontal y en vertical.

El estado inicial del ratón es el de movimiento libre, y sucesivamente se va pasando cada uno de los siguientes. Esta limitación del ratón se puede utilizar cuando por ejemplo queremos mover un objeto en horizontal. La utilidad de la última opción se manifiesta cuando cambiamos de tamaño un objeto y queremos que mantengan la proporción entre el eje horizontal y el vertical.

Manual técnico

La práctica realizada consiste en un editor tridimensional de escenas compuestas por un conjunto de sólidos. Cada objeto se crea a partir de una serie de primitivas, que concretamente son esferas, pirámides, troncos de pirámide, toroides y cubos. Cada una de estas figuras básicas admite una serie de transformaciones hasta darle la forma deseada; se pueden desplazar, rotar, duplicar, unir, etc.

La edición se realiza a partir de cuatro vistas distintas ,cada una de las cuales representa la proyección paralela de la escena desde un determinado punto, y que inicialmente serán las típicas de los programas de diseño (planta, alzado, perfil y una vista tridimensional), y que podrán ser variadas a base de desplazar el punto de observación o rotando el plano de proyección.

Veamos a continuación una serie de cuestiones básicas respecto de la práctica, tal y como se plantearon al comienzo de la realización de la misma.

Representación interna

El modelo más adecuado para la representación de sólidos es el conocido bajo el nombre de $B-rep$ (Boundary Representation). Éste consiste en guardar para cada sólido los datos de los polígonos que forman su contorno.

Por tanto para cada objeto tendremos que guardar, además de las coordenadas de los vértices que lo componen, una lista de polígonos que indique qué vértices forman cada uno de los mismos.

Ya que el lenguaje en el que está implementada la práctica es el Turbo Pascal, las listas de objetos, polígonos y vértices se crean mediante listas enlazadas a base de punteros. Podemos representar una escena mediante el siguiente esquema.

Escena

Objeto 1 Objeto 2 Objeto N

Así, una escena será una lista enlazada de objetos; cada objeto está compuesto a su vez de listas: una de vértices, en la que cada nodo almacena la posición en el espacio de dicho vértice, y otra de polígonos en la que cada polígono es a su vez otra lista enlazada de apuntadores a los vértices que forman el polígono.

La implementación de los tipos en el programa es la siguiente:

− Lista de vértices:

lista_vertices=^tlvert;

tlvert=record

pos : vector;

sig : lista_vertices;

end;

− Lista de referencias a un vértice en un polígono:

l_vertpoli=^tp;

tp=record

vert: p_vertice;

sig: l_vertpoli;

end;

− Lista de polígonos:

lista_poligonos=^lpol;

lpol=record

lvert: l_vertpoli;

sig: lista_poligonos;

end;

− Lista de objetos:

lista_objetos=^lo;

lo=

record

lvert: lista_vertices;

lpoli: lista_poligonos;

sig: lista_objetos;

end;

Este tipo de representación hace que cuando queremos realizar una transformación baste con modificar la posición de los vértices.

Especificación de las vistas

En este apartado vamos a ver la forma en que se define cada una de las vistas que se pueden representar en los cuatro cuadrantes operativos en el programa. Para esto he optado en la práctica por la utilización de los siguientes parámetros.

− Ejes que definen el plano de proyección. Estos ejes nos dan un nuevo sistema de referencia, y los denominaremos en lo subsiguiente *u*, *v* y *n*. Los dos primeros son dos vectores unitarios contenidos en el plano de proyección, y el tercero es un vector también unitario perpendicular al plano de proyección.

− Centro de la pantalla. Especifica, en coordenadas respecto del sistema de referencia anteriormente comentado, la posición del plano de proyección que coincidirá con el centro de la pantalla (o de la vista).

− Escala. Es un valor numérico que indica cuántos pixels equivalen a una unidad de medida en coordenadas reales.

Hay que señalar que, por estar en un sistema de proyección paralela, no precisamos de un origen para el sistema de referencia, puesto que no es relevante la posición de dicho plano, sólamente su orientación; por ello, podemos suponer que éste pasa por el origen.

Todo esto podemos verlo teniendo en cuenta la configuración inicial de las vistas. Gráficamente podemos verlo de esta forma:

1 Y 2 Z

X Y

2 Z 3 Z

Y

X

X

El primer cuadrante representará inicialmente la planta, por lo que el eje *u* será paralelo al eje *x*, el *v* paralelo al *y* y el *n* paralelo al *z*. Sucede algo parecido con los cuadrantes segundo y tercero. El cuarto es especial, ya que inicialmente representa una vista isométrica; por tanto, el eje *v* será paralelo a la bisectriz de los ejes *x* e *y*, y como el eje *n* debe ser paralelo a (1,−1,1), obtendremos el que nos falta, que es el *v*, mediante el producto

vectorial de los anteriores. Por otra parte, la vista estará centrada en todos los casos sobre el punto (0,0,0); por tanto, podemos obtener la siguiente tabla:

	1	2	3	4
Eje u	(1, 0, 0)	(0, 1, 0)	(1, 0, 0)	(1/"2, 1/"2, 0)
Eje v	(0, 1, 0)	(0, 0, 1)	(0, 0, 1)	(−1/"6, 1/"6, 2/"6)
Eje n	(0, 0, 1)	(1, 0, 0)	(0,−1, 0)	(1/"3,−1/"3, 1/"3)

Configuración de las ventanas

El usuario tiene a su disposición cuatro cuadrantes sobre los que definir cada una de las cuatro posibles vistas. Cada ventana viene determinada por el cuadro que las enmarca, es decir, por cuatro parámetros: *cmin*, *fmin,cmax* y *fmax*, que representan las columnas y filas máximas y mínimas que acotan a la ventana.

Inicialmente, el tamaño de las cuatro ventanas es idéntico; podemos ver esto mediante el siguiente gráfico:

2 278 282 558

2

228

232

458

Disponemos de una variable global, *lim_cuad*, que indica en todo momento la disposición de cada cuadrante. También existe un vector, *lim_cuad_ini*, que contiene la configuración inicial, y otro, *maxima*, que hace que uno de los cuadrantes ocupe todo el espacio. Esto se utiliza al maximizar la pantalla.

Generación de las figuras primitivas

Como se ha mencionado anteriormente, la construcción de objetos se realiza modificando una serie de figuras básicas, por lo que debemos conocer la forma de generar cada una y almacenarla en memoria de la forma en que antes se menciona; veámos esto una por una.

− Esfera.− La forma de una esfera puede ser aproximada mediante una malla de puntos unidos por aristas, que al ser desplegada adopta una forma como la de la figura abajo representada. En ella vemos que la figura generada es parecida a la forma de meridianos y paralelos en un globo terráqueo.

La posición de cada vértice vendrá dada por un ángulo en horizontal y otro en vertical, de tal modo que dividimos los 360 grados en partes iguales y vamos calculando las posiciones e insertando los vértices en una lista.

(N/2)−1

Por la forma en que se han calculado las posiciones de los vértices, éstos se encuentran dentro de la lista de derecha a izquierda y de abajo hacia arriba, teniendo en cuenta la malla. Con esto deducimos que cada polígono estará compuesto por un vértice, su sucesor, y los dos que se diferencian de éstos en (N/2)−1 posiciones dentro de la lista, con lo cual podemos crear directamente la lista de polígonos. Además hay que crear los polígonos de la parte superior e inferior, lo que se hace relacionando los vértices del polo sur con el primer vértice de la lista, y a partir de ahí avanzando (N/2)−1 posiciones. Se procede análogamente con el

polo norte, comenzando por el vértice superior izquierdo.

− Pirámide.− La forma de la malla resultado de una pirámide es obvia; en este caso, primero creamos los vértices de la base, conectándolos entre sí. Posteriormente, simplemente se conecta cada uno de estos vértices también con su sucesor en la base y con el que forma la cima de la pirámide; así generamos los polígonos triangulares de cada una de las caras laterales.

− Tronco.− En este caso, la malla que vamos a crear es de la siguiente forma (recordemos que N es el número de vértices por circunferencia y que los vértices de la derecha conectan con los de la izquierda).

N

Para generar la lista de vértices, vamos insertando en ella alternativamente un vértice de cada base, aumentando progresivamente el ángulo en horizontal respecto del centro de las dos bases. De forma parecida a la esfera, en este caso cada polígono está formado por un vértice y sus tres sucesores.

− Toroide.− Recordemos que un toroide tiene dos valores para el número de vértices: uno para el perímetro y otro para la sección. Los llamaremos N1 y N2, respectivamente. De esta manera, la forma de la malla al ser desplegada es la de la figura siguiente, teniendo en cuenta que en este caso los nodos de la derecha conectan con los de la izquierda y los de arriba con los de abajo.

N2

N1

La posición de los vértices se calcula mediante dos ángulos, uno que expresa el desplazamiento respecto del perímetro y otro dentro de la sección. Teniendo en cuenta la disposición de la lista de vértices, tenemos que un vértice estará en el mismo polígono que su sucesor en la lista, y los dos que disten de éstos N2 posiciones dentro de la misma. La diferencia respecto de la esfera es que en este caso hay que considerar la conectividad de la parte superior de la malla con la parte inferior.

− Cubo.− Aunque este tipo de figura aparece como primitiva en el programa, en realidad es generada a partir de un tronco de cuatro vértices y el mismo radio en cada base.

Hay que señalar que en todas las creaciones de objetos, se ha tenido en cuenta el sentido de definición de los polígonos; en este caso, el criterio ha sido el de seguir el sentido de las agujas del reloj, con lo que el vector normal de los polígonos siempre va hacia fuera del sólido.

Selecciones

Durante la realización de esta práctica, se hizo evidente la necesidad que hay en cualquier programa de edición de poder seleccionar una serie de entidades para realiza operaciones con ellas sin tener que hacerlas individualmente sobre cada uno de sus componentes. En este caso, se manifiesta en el manejo de grupos de objetos y de vértices.

La solución adoptada consiste en tener en todo momento una lista de objetos y otra de vértices seleccionados, de tal manera que cada nodo de la misma apunte a un objeto o vértice. De esta forma, todas las rutinas se han diseñado para actuar sobre estas selecciones; en el caso de que deseemos realizar una acción sobre un elemento concreto, con el fin de poder utilizar la misma rutina, simplemente creamos una selección temporal que sólo contendrá a dicho elemento.

El tipo de datos construido para la lista de objetos seleccionados es la siguiente:

l_obj_sel=^los;

los=record

obj: objeto;

sig: l_obj_sel;

end;

El tipo de datos para la lista de vértices es exactamente igual que el de una lista de referencias, que es la que compone cada polígono, ya que contiene punteros a una serie de vértices.

Formato externo

La forma en que se graban las escenas en disco intenta seguir aproximadamente la organización que se sigue en memoria. Para cada objeto, tendremos un conjunto de vértices, especificando su posición, y una lista de polígonos, que harán referencia a esos vértices. Ahora bien, la forma de referirse a esos vértices debe ser distinta; en este caso, a cada vértice se le asigna un número según la posición que ocupa dentro de la lista correspondiente, con lo que la referencia consiste precisamente en ese número.

Veamos más en concreto el formato; los objetos se colocan uno detrás de otro dentro del fichero, teniendo cada uno la siguiente estructura.

Número de vértices

Vértice 1

Vértice 2

...

Vértice V

Número de Polígonos=P

Número de referencias polígono 1

Posición del vértice

...

Posición del vértice

...

Número de referencias polígono P

Posición del vértice

...

Posición del vértice

Supongamos que tenemos un tetraedro, que está formado por cuatro polígonos cada uno compuesto de tres vértices, con esta forma:

A

D

C B

Lista de vértices: A B C D

Lista de polígonos: P1: A B C

P2: A D B

P3: A C D

P4: B D C

La representación en el fichero de este objeto sería la siguiente:

4 (vértices)

Ax Ay Az

Bx By Bz

Cx Cy Cz

Dx Dy Dz

4 (polígonos)

3 (el polígono P1 tiene tres vértices)

1 2 3 (vértices A, B y C)

3 (el polígono P2 tiene tres vértices)

1 4 2 (vértices A, D y B)

3 (el polígono P2 tiene tres vértices)

1 3 4 (vértices A, C y D)

3 (el polígono P2 tiene tres vértices)

2 4 3 (vértices B, D y C)

Si la escena tuviera más objetos, éstos se dispondrían uno detrás de otro sin tener que incluir ninguna

información extra.

Por último debemos mencionar los tipos utilizados para representar cada dato en el fichero; en la tabla se da tanto el tipo como el tamaño en bytes del mismo.

	Tipo de dato	Tamaño en bytes
Nº de vértices	Word	2
Posición vértice	Vector	18
Nº de polígonos	Word	2
Nº vert del políg.	Word	2
Referencia a vért.	Word	2

Funciones auxiliares

El programa incluye una serie de rutinas que no están directamente relacionadas con la base de la práctica, pero que han sido utilizadas para la implementación de la misma, como son funciones matemáticas, manejo de mensajes, etc. Veamos simplemente lo que hace cada una de ellas.

− *Procedure Mensaje(x,y: integer;s: string)*

Presenta el mensaje *s* en la posición (x,y) de la pantalla.

− *Procedure Info(s: string)*

Presenta el mensaje *s* en la parte inferior de la pantalla.

− *Procedure Limita_raton*

Limita el movimiento del ratón a la zona de pantalla demarcada por la vista activa.

− *Procedure Libera_raton*

Permite el movimiento del ratón por toda la pantalla.

− *Procedure Poner_menus*

Presenta en la parte izquierda pantalla todos los menús de acciones.

− *Procedure Cambia_opcion(op,nop: byte)*

Desactiva la opción *op* y activa la *nop*.

− *Function Cad_Ent(s: string): integer*

Transforma la cadena de caracteres *s* a entero.

− *Function Cad_Real(s: string): real*

Transforma la cadena de caracteres *s* a número real.

− *Function Leer_cadena(men: string): string*

Lee una cadena por teclado, apareciendo los sucesivos caracteres en la zona inferior de la pantalla.

− *Procedure Inicia_graf*

Inicializa la pantalla a modo gráfico de 640x480 y 16 colores.

− *Procedure Escala_vector(v1: vector; esc: real;*

var v2: vector);

Multiplica el vector *v1* por el escalar *esc*, devolviendo el resultado en *v2*; es decir, v2 = esc·v1.

− *Procedure Dif_vectores(A,B: vector; var C: vector);*

Realiza la operación vectorial C = A − B.

− *Procedure Suma_vectores(A,B: vector; var C: vector);*

Realiza la operación vectorial C = A + B.

− *Function Prod_escalar(a,b: vector): real;*

Devuelve el producto escalar de los vectores *a* y *b*.

− *Procedure Prod_matrices(a,b: matriz; var c: matriz);*

Multiplica las matrices *a* y *b*, devolviendo el resultado en la matriz *c*.

− *Procedure MatrizXvector(M: matriz; vi: vector;*

var vf: vector)

Multiplica la matriz *M* por el vector *vi*.

− *Procedure VectorXmatriz(vi: vector; M: matriz;*

var vf: vector)

Multiplica el vector *vi* por la matriz *M*. Sería la operación vf = *vitM*

− *Procedure Intro_coord(var A: vector;c1,c2,c3: real)*

Da a las coordenadas del vector *A* los valores *c1, c2* y *c3*.

− *Function Distancia_3d(A,B: vector): real*

Calcula la distancia entre los puntos dados por los vectores *A* y *B.*

− *Function Distancia_2d(A,B: pixel): real*

Calcula la distancia entre los pixels cuya posición determinan *A* y *B*.

− *Function Minimo(a,b: real): real*

Devuelve el valor mínimo entre *a* y *b*.

Rutinas principales

A continuación vamos a analizar las rutinas que intervienen en la representación gráfica de la escena, transformación de los objetos, etc. No entraremos en demasiados detalles respecto de la implementación final de las rutinas, sino que nos quedaremos en una descripción general de las mismas.

Conversión entre sistemas de referencia

En el sistema de diseño actual, disponemos de tres sistemas de referencia distintos: el mundo real, el plano de proyección y la pantalla. Disponemos de una matriz llamada *ejes* para cada vista que contiene los vectores unitarios que determinan los ejes *u*, *v* y *n*. El hecho de definirlo como matriz nos permite acceder a ella de forma global y también de forma individual a cada uno de los vectores que la forman. Además, se han definido unas constantes *u*, *v* y *n* que equivalen respectivamente a 1, 2 y 3, con lo que nos podremos referir a un eje concreto como ej[u], ej[v] y ej[n], ganándose así en claridad. Sucede igual con *x, y* y *z*.

Estos tres sistemas admiten transformaciones de unos a otros, existiendo una serie de rutinas implementadas para cumplir con dicha función:

Trans_wc_pp: Convierte un vector de coordenadas reales a coordenadas respecto del plano de proyección. Para realizar esta conversión, se calcula la proyección del vector sobre cada uno de los ejes del sistema de referencia final. Esto equivale a realizar el producto escalar del vector por los vectores *u*, *v* y *n* correspondientes, o dicho de otra forma, multiplicar el vector por la matriz *ejes* asociada a la vista seleccionada.

Trans_pp_wc: Realiza la función contraria a la anterior, partiendo de un vector en coordenadas respecto al plano de proyección. En este caso habrá que multiplicar cada coordenada del vector a convertir por el vector correspondiente de los ejes *u*, *v* y *n*. Dicho de otro modo, llamando pp al vector original y wc al resultado:

wc = pp[u]·u + pp[v]·v + pp[n]·n

Esto es equivalente a multiplicar por delante el vector *pp* por la matriz de los ejes.

Trans_pp_2d: Convierte un vector de coordenadas respecto del plano de proyección a coordenadas respecto de la ventana activa. Para aclarar esto veamos la siguiente figura:

Puesto que el centro de la ventana está situado en el punto indicado por CW (contenido en una matriz del mismo nombre), debemos calcular primero las coordenadas del punto respecto de dicho centro, multiplicar dicha diferencia por la escala de la vista y sumar el resultado al centro de la ventana en coordenadas de pantalla. Las coordenadas de pantalla se expresan mediante el tipo *pixel*, que contiene un número de fila y otro de columna; puesto que las filas crecen en sentido opuesto al eje *v*, este término debe apaecer con signo negativo.

Trans_2d_pp. Realiza la función opuesta a la anterior, y su deducción es simple si se deshacen las operaciones realizadas en el punto anterior, es decir, calculamos las coordenadas respecto del centro de la ventana en pantalla y dividimos por la escala y sumamos al vector CW. Naturalmente, la coordenada *n* no puede ser deducida, por lo que se le da un valor arbitrario igual a cero.

Trans_wc_2d. Esta rutina consiste tan sólo en la composición de *Trans_wc_pp* y *Trans_pp_2d*.

Dibujar una escena

Para sacar por pantalla una escena, lo que se hace es recorrer toda la lista de objetos, transformando las coordenadas de cada vértice a coordenadas de pantalla y dibujando la correspondiente línea.

El color de la línea varía según el objeto esté seleccionado o no; para ello, se busca el objeto en la lista de objetos seleccionados y se pone el color blanco ó verde.

También puede cambiar el tipo de línea, ya que cuando el polígono es posterior se dibuja una línea discontinua. Para ello, se utiliza el método del vector normal, tomando tres vértices del polígono y calculando dicho vector. Si el producto escalar de dicho vector por el vector normal al plano es menor que cero, el polígono será posterior, en caso contrario será visible. Esto es así porque el vector normal va hacia fuera del sólido y además el vector de proyección es opuesto a la dirección de *n*.

N1 N2

n Dp

$N1 \cdot n < 0$ −−> Posterior

$N2 \cdot n > 0$ −−> Visible

Operaciones sobre vistas

Existe una serie de transformaciones que se pueden hacer sobre una vista; veamos cuáles son y en qué consisten:

Acercar o alejar. Puesto que estamos en proyección paralela, la forma de acercar o de alejar el punto de observación a la escena es aumentar o disminuir la escala. Esto se hace mediante la rutina *Zoom_vista*, multiplicando la escala por un factor determinado.

Desplazar. Se utiliza para cambiar la posición del punto desde el que queremos ver la escena. Esto se consigue en el procedimiento *desplaza_camara*, que simplemente va leyendo la variación que se produce en la posición del ratón y realiza el cambio en las componentes *u* y *v* sumándole la variación horizontal y vertical, respectivamente.

Centrar. Para hacer que la escena coincida con los bordes de la ventana, lo primero que se hace es delimitar las coordenadas *u* y *v* máxima y mínima de la misma, transformando todos los puntos a coordenadas del plano de proyección. Posteriormente, se calcula la escala dividiendo el tamaño de la ventana entre el tamaño en el plano de proyección de la escena, y se fija el centro de la vista en las coordenadas del plano que dan el centro de la escena.

Maximizar y minimizar. En todo momento disponemos de una variable llamada *maximizado* que indica si la ventana actual ocupa todo el espacio de visualización. De este modo, sabemos si debemos restaurar los límites originales de los cuadrantes o si debemos cargar los datos del vector *maxima* (antes comentado).

Rotar. Esta función tiene como objetivo el poder ofrecer una visión de la escena desde cualquier punto de vista; al mover el ratón horizontalmente, generamos una rotación sobre el eje *z*, mientras que si lo hacemos verticalmente, estamos cambiando la inclinación del plano de proyección. En el primer caso, usamos una matriz de transformación típica mediante el seno y el coseno del ángulo de rotación. En el segundo, debemos rotar respecto del eje *u*, con lo que en lugar de multiplicar por detrás multiplicamos por delante. Este razonamiento es análogo al usado al transformar entre sistemas de referencia, solo que en lugar de transformar

un vector lo hacemos con tres vectores, es decir, con una matriz. Todo esto está implementado en el procedimiento *Rotar_camara*.

Restaurar. Se utiliza para recuperar el estado inicial de las vistas; para esto, tan sólo hay que hacer que los vectores *lim_cuad*, *cw* y *ejes* contengan los valores iniciales, definidos en la sección de constantes.

Ejes. Para ocultar ó hacer visibles los ejes, se cambia el valor del vector *ej_vis*, que cuando es *true* indica que los ejes son visibles; este vector se usa en el proceso de representar la escena para dibujar los ejes *x*, *y* y *z*.

Operaciones sobre objetos

Para realizar las transformaciones, el criterio seguido ha sido el del cálculo vectorial, en lugar del clásico de usar matrices cuadradas de tamaño 4. Esto es debido a que, una vez implementado este último método, se pone de manifiesto que la mayoría de las casillas de las matrices usadas están a cero, con lo que estamos haciendo cálculos innecesarios. Por ejemplo, para desplazar un vértice, sólo necesitaríamos los valores de la última columna, pero aun así estamos haciendo gran cantidad de productos (recordemos que el producto de matrices tiene complejidad de orden n3). De aquí se deduce que el uso de matrices de este tipo está más justificado en el caso de transformaciones muy complejas.

Por ello, he preferido deducir analíticamente las operaciones vectoriales y aplicarlas a los puntos a transformar.

Crear un objeto. La rutina *Nuevo_objeto* permite introducir un nuevo objeto en la escena, mediante la elección de uno de los tipos básicos y ejecutando los procedimientos *esfera*, *piramide*, *tronco* y *toroide*, que crean un objeto del tipo correspondiente, tal y como se comentó anteriormente.

En este procedimiento usamos el procedimiento *polígono*, que dibuja un polígono regular de un número determinado de lados. Con esto permitimos que al introducir gráficamente el tamaño de un objeto se vea la forma del mismo.

Elegir un objeto o vértice. La rutina *Seleccionar* incluye estas dos posibilidades, ya que devuelve el vértice más cercano al puntero del ratón y el objeto al que este vértice pertenece. El proceso es simple: primero se lee la posición del ratón, y a continuación se consultan todos los vértices de la escena, transformándolos a coordenadas de pantalla; en cada momento recordamos el vértice más cercano y el objeto en el que está incluido éste. Hay que señalar que esta rutina no modifica la lista de objetos ni de vértices seleccionados, simplemente señala a los más cercanos. Serán las acciones más complejas las que la utilicen (por ejemplo, antes de mover un objeto lo seleccionamos).

Seleccionar un objeto o vértice. En el caso de un objeto, basta con insertar el objeto en la lista *obj_sel*, que contiene los objetos seleccionados; esto se hace con la rutina *selec_objeto*. El caso de un vértice es idéntico, pero con la lista *vert_sel*, usando *selec_vert*.

Deseleccionar un objeto o vértice. Para conseguir esto basta con eliminar de la lista de objetos ó vértices seleccionados el objeto en cuestión.

Consulta sobre selección de un objeto ó vértice. Para saber si un objeto ó vértice está seleccionado, recorremos la lista hasta que lo encontramos. Esto se implementa en *Esta_selec_obj* y *Esta_selec_vert*.

Seleccionar un conjunto de objetos ó vértices. Todas las acciones descritas anteriormente se utilizan para seleccionar un único elemento; pero existe otro modo de hacer selecciones: mediante cuadro, con la rutina *Selec_cuadro*. En este caso, se leen las esquinas del cuadro, y se comienza a recorrer la lista de objetos; todos los vértices que caigan dentro del cuadro y los objetos a los que éstos pertenecen son candidatos. Entonces,

según la acción que queramos realizar (seleccionar/deseleccionar objeto/vértice), se aplica la rutina correspondiente, teniendo cuidado de no seleccionar un objeto ya seleccionado y de no deseleccionar uno que no lo esté.

Eliminar un objeto. La eliminación de un objeto consiste en recorrer todas las listas que forman su estructura e ir liberando la memoria correspondiente.

Las siguientes acciones sobre objetos actuan siempre sobre una lista de objetos seleccionados; recordemos que cuando se actúa sobre un sólo objeto se habrá creado anteriormente una selección virtual conteniendo sólamente a ese objeto.

Mover un conjunto de objetos. Cuando movemos un objeto en una vista determinada queremos que lo haga en paralelo con el plano de proyección, por lo que el movimiento del ratón lo convertimos a un vector en coordenadas respecto a dicho plano, y al convertirlo a coordenadas reales, tendremos el desplazamiento real. Para modificar los objetos, recorremos la lista de objetos seleccionados y desplazamos cada vértice de cada objeto. Todo este proceso se realiza dentro de *Mover_selec*.

Escalar un conjunto de objetos. Para la realización de esta transformación disponemos de dos rutinas: *Escalar_selec*, que lee el ratón y determina el grado de escalado respecto de cada eje, y *Escalar_objeto*, que realiza el escalado del objeto propiamente dicho.

En la primera rutina se procede de forma parecida al anterior apartado, es decir, se capta la variación en el ratón, y la variación horizontal se traduce como un escalado en el eje *u* y la vertical en el eje *v*. Tras convertir este vector a coordenadas reales, se recorre la lista de objeto seleccionados y se aplica *Escalar_objeto* sobre cada uno de ellos. Este escalado se realiza calculando la posición de cada vértice respecto del punto de referencia, y multiplicando cada coordenada del vector resultado por el vector de escalado.

Rotar un conjunto de objetos. Básicamente la estructura es la misma que el caso anterior, con la salvedad de que el movimiento del ratón es interpretad ahora como el ángulo en que se desea rotar el objeto.

Para rotar un objeto, puesto que esta rotación es sobre el plano de proyección, se debe expresar cada vértice en relación a dicho sistema de referencia, calcular su posición respecto del centro de rotación, y realizar la transformación sobre el vector obtenido. Una vez rotado el vector lo volvemos a convertir a coordenadas reales.

Copiar un conjunto de objetos. Este proceso consiste en recorrer la lista de objetos seleccionados y realizar un *clon* del cada objeto. Para esto, en la rutina *Copiar_selec* se recorre la lista de vértices y se hace un calco de la misma. Posteriormente, para cada polígono, se busca los vértices a los que referencia el original; la posición que ocupe un vértice en la lista original nos dará la posición que ocupa su imagen en la nueva vista, con lo que el nuevo polígono debe contener a este nuevo vértice en su lista de referencias.

Unir un conjunto de objetos. La unión básica se hace entre dos objetos, en el proceso *Unir_objetos*; el proceso es bastante sencillo: basta con unir las dos listas de vértices en una única lista, y unir de la misma forma las de polígonos, además de borrar uno de los objetos.

Existe otra rutina, *Unir seleccion*, que une un conjunto de objetos seleccionados. Para ello, simplemente va uniendo todos los objetos con el primero de ellos. Además debe borrar de la selección todos los objetos excepto el primero, ya que éstos han dejado de existir como tales.

Operaciones sobre una escena

Las operaciones relativas a escenas son las siguientes:

Nueva escena. Antes de eliminar la escena actual, se pide confirmación, para posteriormente utilizar la rutina *Eliminar_escena*, que consiste en recorrer la lista de objetos e ir eliminándolos.

Cargar_escena. Siguiendo el esquema antes comentado del formato de un fichero, carga en memoria una escena, eliminándo la actual, previa confirmación. Lógicamente se comprueba que el fichero a cargar realmente existe, abriéndolo y consultando el error devuelto por el sistema operativo. El procedimiento asociado es *Cargar_escena*.

Grabar_escena. Se pide por teclado el nombre del fichero al que se desea volcar la escena, y si este no existe, se procede a su grabación siguiendo el formato anteriormente explicado.

Mezclar. Utiliza también el procedimiento *Cargar_escena*, con la diferencia de que no elimina la escena actual, sino que los objetos se van incorporando a la misma conforme son leidos.

```
{$M 65000,0,655350}

Program Infografia;

uses graph,crt,mouse,drivers,dos;

const x=1;y=2;z=3;

u=1;v=2;n=3;

type pixel=

record

c,f: integer;

end;

vector=array[1..3] of real;

matriz=array[1..3] of vector;

lista_vertices=^tlvert;

p_vertice=lista_vertices;

tlvert=record

pos : vector;

sig : lista_vertices;

end;

l_vertpoli=^tp;

tp=record
```

```
vert: p_vertice;
sig: l_vertpoli;
end;
lista_poligonos=^lpol;
lpol=record
lvert: l_vertpoli;
sig: lista_poligonos;
end;
lista_objetos=^lo;
objeto=lista_objetos;
lo=
record
lvert: lista_vertices;
lpoli: lista_poligonos;
sig: lista_objetos;
end;
l_obj_sel=^los;
los=record
obj: lista_objetos;
sig: l_obj_sel;
end;
l_vert_sel=l_vertpoli;
type num_cuadrante=0..3;
t_lim_cuad=
record
cmin,fmin,cmax,fmax: integer;
```

```
end;
tlimites=array[num_cuadrante] of t_lim_cuad;
tejes=array[num_cuadrante] of matriz;
torig=array[num_cuadrante] of vector;
tescala=array[num_cuadrante] of real;
tej_vis=array[num_cuadrante] of boolean;
modo_raton=(libre,horiz,vertic,isomet);
alfabeto=set of char;
var izq,der: boolean;
escena: lista_objetos;
maximizado: boolean;
obj_sel: l_obj_sel;
vert_sel: l_vert_sel;
ejes: tejes;
cw: torig;
escala: tescala;
lim_cuad: tlimites;
ej_vis: tej_vis;
cuad_act: num_cuadrante;
mraton: modo_raton;
const raiz2_1=0.707106781186548;
raiz3_1=0.577350269189626;
raiz6_1=0.408248290463863;
const ejes_ini: tejes=((( 1, 0, 0), ( 0, 1, 0), ( 0, 0, 1)),
(( 0, 1, 0), ( 0, 0, 1), ( 1, 0, 0)),
(( 1, 0, 0), ( 0, 0, 1), ( 0,−1, 0)),
```

((raiz2_1, raiz2_1, 0),

(−raiz6_1, raiz6_1, 2*raiz6_1),

(raiz3_1, −raiz3_1, raiz3_1)));

orig_ini: torig=((0, 0, 0),(0, 0, 0),(0, 0, 0),(0, 0, 0));

lim_cuad_ini: tlimites=

((cmin: 2; fmin: 2; cmax: 278; fmax: 228),

(cmin: 282; fmin: 2; cmax: 558; fmax: 228),

(cmin: 2; fmin: 232; cmax: 278; fmax: 458),

(cmin: 282; fmin: 232; cmax: 558; fmax: 458));

Maxima: t_lim_cuad=(cmin: 2; fmin: 2; cmax: 558; fmax: 458);

escala_ini: tescala=(100, 100, 100, 100);

ej_vis_ini: tej_vis=(false, false, false, false);

maxreal: real=1.7e38;

cos30=0.866025403784439;

sen30=0.5;

coef_zoom= 0.8;

final_lista=nil;

color_oselec=green;

color_vselec=brown;

color_nosel=white;

color_ejes=yellow;

modo2D=true;

modo3D=false;

caracteres=[#0..#255];

digitos=['0'..'9'];

const numitems=27;

```
const menu: array[1..numitems] of
record
texto: string[10];
ayuda: string[60];
fila: byte;
end=
((texto:'Acercar'; ayuda:'Acercar al observador a la escena'; fila: 1),
(texto:'Alejar' ; ayuda:'Alejar al observador de la escena'; fila: 2),
(texto:'Desplazar'; ayuda:'Desplazar el punto de observaci¢n'; fila: 3),
(texto:'Centrar'; ayuda:'Centrar la escena en la vista seleccionada'; fila: 4),
(texto:'Max/Min'; ayuda:'Maximizar o minimizar la vista seleccionada'; fila: 5),
(texto:'Rotar'; ayuda:'Realizar una rotaci¢n de la c mara'; fila: 6),
(texto:'Nuevo'; ayuda:'Crear un nuevo objeto'; fila:10),
(texto:'Seleccion'; ayuda:'Seleccionar un objeto'; fila:11),
(texto:'Deselec'; ayuda:'Deseleccionar un objeto'; fila:12),
(texto:'Eliminar'; ayuda:'Eliminar un objeto/conjunto existentes'; fila:13),
(texto:'Mover'; ayuda:'Mover un objeto/conjunto'; fila:14),
(texto:'Escala 2D'; ayuda:'Cambiar el tama¤o en 2D de un objeto/conjunto'; fila:15),
(texto:'Escala 3D'; ayuda:'Cambiar el tama¤o en 3D de un objeto/conjunto'; fila:16),
(texto:'Rotar'; ayuda:'Rotar sobre s¡ mismo un objeto/conjunto'; fila:17),
(texto:'Copiar'; ayuda:'Realizar un duplicado de un objeto/conjunto'; fila:18),
(texto:'Unir'; ayuda:'Unir dos ¢ m s objetos para formar uno s¢lo'; fila:19),
(texto:'Seleccion'; ayuda:'Seleccionar un v rtice'; fila:21),
(texto:'Mover'; ayuda:'Mover un v rtice/conjunto de v rtices'; fila:23),
(texto:'Cargar'; ayuda:'Cargar una escena almacenada en un fichero'; fila:26),
(texto:'Grabar'; ayuda:'Salvar la escena en un fichero'; fila:27),
```

```
(texto:'Nueva'; ayuda:'Crear una escena nueva'; fila:28),
(texto:'Restaurar'; ayuda:'Restaurar la vista inicial'; fila: 7),
(texto:'Deselec'; ayuda:'Deseleccionar un objeto'; fila:22),
(texto:'Escalar'; ayuda:'Escala v rtices respecto del centro'; fila:24),
(texto:'Ejes'; ayuda:'Muestra u oculta los ejes X, Y y Z'; fila: 8),
(texto:'Mezclar'; ayuda:'Mezcla la escena cargada con la actual'; fila:29),
(texto:'Salir'; ayuda:'Salir del programa'; fila:30));
const acer=1;alej=2;desp=3;cent=4;mxmn=5;roca=6;
nuev=7;slob=8;dsob=9;elim=10;mvob=11;es2D=12;es3D=13;rota=14;copi=15;unio=16;
slvt=17;mvvt=18;carg=19;grab=20;nues=21;rest=22;dsvt=23;esvt=24;eje=25;mezc=26;
salir=27;
Function Leertecla: char;
begin
mem[$40:$1A]:= mem[$40:$1C];
Leertecla:= readkey;
end;
Procedure Mensaje(x,y: integer;s: string);
var antcol: byte;
vp: viewporttype;
begin
getviewsettings(vp);
setviewport(0,0,getmaxx,getmaxy,clipon);
antcol:= getcolor;
setcolor(yellow);
setfillstyle(1,lightblue);
bar(x,y,x+textwidth(s),y+textheight(s));
```

```
outtextxy(x,y,s);
setcolor(antcol);
setviewport(vp.x1,vp.y1,vp.x2,vp.y2,clipon);
end;
Procedure Info(s: string);
var vp: viewporttype;
begin
getviewsettings(vp);
setviewport(0,0,639,479,clipon);
setfillstyle(1,lightblue);
bar(0,465,550,479);
Mensaje(5,465,s);
setviewport(vp.x1,vp.y1,vp.x2,vp.y2,clipon);
end;
Procedure Limita_raton;
var vp: viewporttype;
begin
getviewsettings(vp);
with vp do begin
Situarraton((x1+x2) div 2,(y1+y2) div 2);
zonaraton(x1,y1,x2,y2);
end;
end;
Procedure Libera_raton;
begin
zonaraton(0,0,639,479);
```

```
end;
Procedure Poner_menus;
var i: byte;
vp: viewporttype;
begin
getviewsettings(vp);
setviewport(0,0,639,479,clipon);
setcolor(yellow);
outtextxy(562,12,'Vista');
setcolor(white);
for i:= 1 to numitems do
outtextxy(565,10+menu[i].fila*15,menu[i].texto);
setcolor(yellow);
outtextxy(562,147,'Objeto');
outtextxy(562,312,'V rtice');
outtextxy(562,387,'Escena');
setviewport(vp.x1,vp.y1,vp.x2,vp.y2,clipon);
end;
Procedure Cambia_opcion(op,nop: byte);
var vp: viewporttype;
begin
punterooff;
getviewsettings(vp);
setviewport(0,0,639,479,clipon);
if op>0 then begin
setfillstyle(1,lightblue);
```

```
bar(563,9+menu[op].fila*15,639,20+menu[op].fila*15);
setcolor(white);
outtextxy(565,10+menu[op].fila*15,menu[op].texto);
end;
if nop>0 then begin
setfillstyle(1,white);
bar(563,10+menu[nop].fila*15,639,20+menu[nop].fila*15);
setcolor(blue);
outtextxy(565,12+menu[nop].fila*15,menu[nop].texto);
setviewport(vp.x1,vp.y1,vp.x2,vp.y2,clipon);
info(menu[nop].ayuda)
end
else
info('');
punteroon;
end;
Function Cad_Ent(s: string): integer;
var i: integer;
c: integer;
begin
val(s,i,c);
Cad_Ent:= i;
end;
Function Cad_Real(s: string): real;
var i: real;
c: integer;
```

```
begin
val(s,i,c);
Cad_Real:= i;
end;
Function Leer_cadena(men: string; conjcar: alfabeto): string;
var car: char;
x,y: integer;
vp: viewporttype;
s: string;
begin
getviewsettings(vp);
setviewport(0,0,getmaxx,getmaxy,clipon);
setcolor(yellow);
info(men);
x:= textwidth(men);
y:= 465;
s:= '';
car:= #0;
while car<>#13 do begin
car:= Leertecla;
if car in ['0'..'9','A'..'Z','a'..'z','.','\',':'] then begin
s:= s+car;
outtextxy(x,y,s);
end;
if (car=#8) and (s<>'') then begin
s[0]:= chr(length(s)−1);
```

```
info(men);
outtextxy(x,y,s);
end;
end;
Leer_cadena:= s;
setviewport(vp.x1,vp.y1,vp.x2,vp.y2,clipon);
end;
Procedure Inicia_graf;
var gd,gm: integer;
begin
gd:= detect;
initgraph(gd,gm,'');
end;
Procedure Escala_vector( v1: vector; esc: real; var v2: vector);
var i: 1..3;
begin
for i:= 1 to 3 do
v2[i]:= v1[i]*esc;
end;
Procedure Dif_vectores(A,B: vector; var C: vector);
var i: 1..3;
begin
for i:= 1 to 3 do
C[i]:= A[i]−B[i];
end;
Procedure Suma_vectores( A,B: vector; var C: vector);
```

```
var i: 1..3;
begin
for i:= 1 to 3 do
C[i]:= A[i]+B[i];
end;
Function Prod_escalar(a,b: vector): real;
begin
Prod_escalar:= a[1]*b[1]+a[2]*b[2]+a[3]*b[3];
end;
Procedure Prod_matrices(a,b: matriz; var c: matriz);
var i,j,k: 1..3;
begin
for i:= 1 to 3 do
for j:= 1 to 3 do begin
c[i,j]:= 0;
for k:= 1 to 3 do
c[i,j]:= c[i,j]+a[i,k]*b[k,j];
end;
end;
Procedure Trans_PP_2d(pp: vector; var pix: pixel; cuad: num_cuadrante);
var c,f: real;
begin
with lim_cuad[cuad] do begin
c:= (cmax−cmin)/2 + (pp[u] − cw[cuad][u])/escala[cuad];
f:= (fmax−fmin)/2 − (pp[v] − cw[cuad][v])/escala[cuad];
{ if c>cmax−cmin then c:= cmax−cmin+1;
```

```
if c<0 then c:= −1;
if f>fmax−fmin then f:= fmax−fmin+1;
if f<0 then f:= −1;}
pix.c:= round(c);
pix.f:= round(f);
end;
end;
Procedure Trans_2d_PP(pix: pixel; var pp: vector; cuad: num_cuadrante);
var cv: pixel;
begin
with lim_cuad[cuad] do begin
pp[u]:= cw[cuad][u]+(pix.c − (cmax−cmin)/2)*escala[cuad];
pp[v]:= cw[cuad][v]−(pix.f − (fmax−fmin)/2)*escala[cuad];
pp[n]:= 0;
end;
end;
Procedure MatrizXvector(M: matriz; vi: vector; var vf: vector);
begin
vf[1]:= Prod_escalar(M[1],vi);
vf[2]:= Prod_escalar(M[2],vi);
vf[3]:= Prod_escalar(M[3],vi);
end;
Procedure VectorXmatriz(vi: vector; M: matriz; var vf: vector);
begin
vf[1]:= vi[1]*M[1,1]+vi[2]*M[2,1]+vi[3]*M[3,1];
vf[2]:= vi[1]*M[1,2]+vi[2]*M[2,2]+vi[3]*M[3,2];
```

```
vf[3]:= vi[1]*M[1,3]+vi[2]*M[2,3]+vi[3]*M[3,3];
end;
Procedure Trans_PP_WC(pp: vector; var wc: vector; cuad: num_cuadrante);
begin
VectorXMatriz( pp, ejes[cuad], wc);
end;
Procedure Trans_WC_PP(wc: vector; var pp: vector; cuad: num_cuadrante);
var esc: real;
begin
MatrizXvector( ejes[cuad], wc, pp);
end;
Procedure Trans_WC_2d(wc: vector; var pix: pixel; cuad: num_cuadrante);
var pp: vector;
begin
Trans_WC_PP(wc,pp,cuad);
Trans_PP_2d(pp,pix,cuad);
end;
Procedure Intro_coord(var A: vector;c1,c2,c3: real);
begin
A[1]:= c1; A[2]:= c2; A[3]:= c3;
end;
Function Distancia_3d(A,B: vector): real;
var o,p,q,temp: real;
begin
o:= sqr(B[1]−A[1]);
p:= sqr(B[2]−A[2]);
```

```
q:= sqr(B[3]−A[3]);
temp:= o+p+q;
Distancia_3d:= sqrt(temp);
end;
Function Distancia_2d(A,B: pixel): real;
var o,p,temp: real;
dc,df: real;
begin
dc:= B.c−A.c; o:= sqr(dc);
df:= B.f−A.f; p:= sqr(df);
temp:= o+p;
Distancia_2d:= sqrt(temp);
end;
Function Minimo(a,b: real): real;
begin
if a<=b then Minimo:= a
else Minimo:= b;
end;
Procedure Activa_cuadrante(cuad: num_cuadrante);
begin
setviewport(lim_cuad[cuad].cmin,lim_cuad[cuad].fmin,
lim_cuad[cuad].cmax,lim_cuad[cuad].fmax,clipon);
end;
Function Esta_selec_vert(vert: p_vertice): boolean;
var p_vs: l_vert_sel;
begin
```

```
p_vs:= vert_sel;
while (p_vs<>nil) and (p_vs^.vert<>vert) do
p_vs:= p_vs^.sig;
Esta_selec_vert:= p_vs<>nil;
end;
Procedure Selec_vertice(vert: p_vertice; var vert_s: l_vert_sel);
var l_vs: l_vert_sel;
begin
new(l_vs);
l_vs^.vert:= vert;
l_vs^.sig:= vert_s;
vert_s:= l_vs;
end;
Procedure Desel_vertice(vert: p_vertice; var vert_s: l_vert_sel);
var l_vs: l_vert_sel;
cabeza: l_vert_sel;
temp: l_vert_sel;
begin
new(cabeza);
cabeza^.vert:= nil;
cabeza^.sig:= vert_s;
l_vs:= cabeza;
while (l_vs^.sig^.vert<>vert) do
l_vs:= l_vs^.sig;
temp:= l_vs^.sig;
l_vs^.sig:= l_vs^.sig^.sig;
```

```
dispose(temp);
vert_s:= cabeza^.sig;
dispose(cabeza);
end;
Function Esta_selec_obj(obj: objeto): boolean;
var p_os: l_obj_sel;
begin
p_os:= obj_sel;
while (p_os<>nil) and (p_os^.obj<>obj) do
p_os:= p_os^.sig;
Esta_selec_obj:= p_os<>nil;
end;
Procedure Selec_objeto(obj: objeto; var obj_s: l_obj_sel);
var l_os: l_obj_sel;
begin
new(l_os);
l_os^.obj:= obj;
l_os^.sig:= obj_s;
obj_s:= l_os;
end;
Procedure Desel_objeto(obj: objeto;var obj_s: l_obj_sel);
var l_os: l_obj_sel;
cabeza: l_obj_sel;
temp: l_obj_sel;
begin
new(cabeza);
```

```
cabeza^.obj:= nil;
cabeza^.sig:= obj_s;
l_os:= cabeza;
while (l_os^.sig^.obj<>obj) do
l_os:= l_os^.sig;
temp:= l_os^.sig;
l_os^.sig:= l_os^.sig^.sig;
dispose(temp);
obj_s:= cabeza^.sig;
dispose(cabeza);
end;
Procedure Eliminar_objeto(obj: objeto);
var pobj,pobj2: objeto;
pvert1,pvert2: p_vertice;
plvert1,plvert2: l_vertpoli;
pol1,pol2: lista_poligonos;
begin
pobj:= obj;
pvert1:= obj^.lvert;
while pvert1<>nil do begin
pvert2:= pvert1;
pvert1:= pvert1^.sig;
if esta_selec_vert(pvert2) then
desel_vertice(pvert2,vert_sel);
dispose(pvert2);
end;
```

```
pol1:= obj^.lpoli;
while pol1<>nil do begin
pol2:= pol1;
plvert1:= pol1^.lvert;
while plvert1<>nil do begin
plvert2:= plvert1;
plvert1:= plvert1^.sig;
dispose(plvert2);
end;
pol1:= pol1^.sig;
dispose(pol2);
end;
if obj=escena then begin
escena:= obj^.sig;
dispose(obj);
end
else begin
pobj2:= escena;
while pobj2^.sig<>pobj do
pobj2:= pobj2^.sig;
pobj2^.sig:= pobj2^.sig^.sig;
dispose(obj);
end;
end;
Procedure Eliminar_escena(var esc: lista_objetos);
var ob1,ob2: objeto;
```

```
begin
while obj_sel<>nil do
Desel_objeto(obj_sel^.obj, obj_sel);
while vert_sel<>nil do
Desel_vertice(vert_sel^.vert, vert_sel);
ob1:= escena;
while ob1<>nil do begin
ob2:= ob1^.sig;
Eliminar_objeto(ob1);
ob1:= ob2;
end;
end;
Procedure Ins_obj_esc(var o: objeto);
begin
new(o);
o^.lvert:= final_lista;
o^.lpoli:= final_lista;
o^.sig:= escena;
escena:= o;
end;
Procedure Ins_vert_obj(var pv: p_vertice;var obj: objeto; c: vector);
begin
new(pv);
pv^.sig:= obj^.lvert;
obj^.lvert:= pv;
pv^.pos:= c;
```

```
end;
Procedure Ins_vert_poli(var pv: p_vertice; var pol: l_vertpoli);
{A¤adimos el v rtice pv al principio de la lista de v rtices}
var lv: l_vertpoli;
begin
new(lv);
lv^.vert:= pv;
lv^.sig:= pol;
pol:= lv;
end;
Procedure Ins_poli_obj(var lvert: l_vertpoli; var obj: objeto);
var l_poli: lista_poligonos;
begin
new(l_poli);
l_poli^.lvert:= lvert;
l_poli^.sig:= obj^.lpoli;
obj^.lpoli:= l_poli;
end;
Procedure Centro_caja(l_os: l_obj_sel; var cc: vector);
var minx,miny,minz,maxx,maxy,maxz: real;
p_os: l_obj_sel;
pv: p_vertice;
begin
minx:= maxreal;miny:= maxreal;minz:= maxreal;
maxx:= −maxreal;maxy:= −maxreal;maxz:= −maxreal;
p_os:= l_os;
```

```
while p_os<>nil do begin
pv:= p_os^.obj^.lvert;
while pv<>nil do begin
minx:= Minimo(minx,pv^.pos[x]);maxx:= −Minimo(−maxx, −pv^.pos[x]);
miny:= Minimo(miny,pv^.pos[y]);maxy:= −Minimo(−maxy, −pv^.pos[y]);
minz:= Minimo(minz,pv^.pos[z]);maxz:= −Minimo(−maxz, −pv^.pos[z]);
pv:= pv^.sig;
end;
p_os:= p_os^.sig;
end;
cc[x]:= (minx+maxx)/2; cc[y]:= (miny+maxy)/2; cc[z]:= (minz+maxz)/2;
end;
Procedure Dibuja_escena( cuad: num_cuadrante);
var p1,p2: pixel;
obj: objeto;
pol: lista_poligonos;
vertpol: l_vertpoli;
i: integer;
colant: byte;
Norm: vector;
v1,v2,v3: vector;
ej: vector;
begin
punterooff;
setfillstyle(1,0);
setcolor(color_nosel);
```

```
Activa_cuadrante(cuad);
clearviewport;
obj:= escena;
while obj<>final_lista do begin
if Esta_selec_obj(obj) then
setcolor(color_oselec)
else
setcolor(color_nosel);
pol:= obj^.lpoli;
while pol<>final_lista do begin
vertpol:= pol^.lvert;
v1:= vertpol^.vert^.pos;
v2:= vertpol^.sig^.vert^.pos;
v3:= vertpol^.sig^.sig^.vert^.pos;
Dif_vectores( v2, v1, v1);
Dif_vectores( v3, v2, v2);
Norm[x]:= v1[y]*v2[z]−v1[z]*v2[y];
Norm[y]:= v1[z]*v2[x]−v1[x]*v2[z];
Norm[z]:= v1[x]*v2[y]−v1[y]*v2[x];
if Prod_escalar(ejes[cuad][n],Norm)>0 then setlinestyle(0,0,1)
else setlinestyle(3,0,1);
Trans_WC_2d(vertpol^.vert^.pos,p1,cuad);
moveto(p1.c,p1.f);
vertpol:= vertpol^.sig;
while vertpol<>nil do begin
trans_WC_2d(vertpol^.vert^.pos,p2,cuad);
```

```
Lineto(p2.c,p2.f);
if Esta_selec_vert(vertpol^.vert) then begin
colant:= getcolor;
setcolor(color_vselec);
pieslice(p2.c,p2.f,0,360,2);
setcolor(colant);
end;
vertpol:= vertpol^.sig;
end;
lineto(p1.c,p1.f);
pol:= pol^.sig;
end;
obj:= obj^.sig;
end;
if ej_vis[cuad] then begin
setlinestyle(1,0,0);
setcolor(color_ejes);
ej[x]:= 0;ej[y]:= 0;ej[z]:= 0;
Trans_wc_2d(ej,p1,cuad);
ej[x]:= 5000;ej[y]:= 0;ej[z]:= 0;
Trans_wc_2d(ej,p2,cuad);
Line(p1.c,p1.f,p2.c,p2.f);
ej[x]:= 0;ej[y]:= 5000;ej[z]:= 0;
Trans_wc_2d(ej,p2,cuad);
Line(p1.c,p1.f,p2.c,p2.f);
ej[x]:= 0;ej[y]:= 0;ej[z]:= 5000;
```

```
Trans_wc_2d(ej,p2,cuad);
Line(p1.c,p1.f,p2.c,p2.f);
end;
punteroon;
setlinestyle(0,0,1);
end;
Procedure Dibuja_todo( cuad: num_cuadrante);
var i: byte;
begin
if not maximizado then
for i:= 0 to 3 do
Dibuja_escena(i)
else
Dibuja_escena(cuad);
Activa_cuadrante(cuad);
end;
Procedure Sentido_raton;
begin
setcolor(red);
setlinestyle(0,0,3);
with lim_cuad[cuad_act] do
moveto((cmax−cmin) div 2,(fmax−fmin) div 2);
if mraton in [libre,horiz] then begin
linerel(−15,0); linerel(5,−3);linerel(−5,3);
linerel(5,3);linerel(−5,−3);
linerel(30,0); linerel(−5,−3);linerel(5,3);
```

```
linerel(−5,3);linerel(5,−3);
linerel(−15,0);
end;
if mraton in [libre,vertic] then begin
linerel(0,−15); linerel(−3,5);linerel(3,−5);
linerel(3,5);linerel(−3,−5);
linerel(0,30); linerel(−3,−5);linerel(3,5);
linerel(3,−5);linerel(−3,5);
linerel(0,−15);
end;
if mraton=isomet then begin
linerel(−10,−10); linerel(5,0);linerel(−5,0);
linerel(0,5);linerel(0,−5);
linerel(20,20); linerel(−5,0);linerel(5,0);
linerel(0,−5);linerel(0,5);
end;
setlinestyle(0,0,1);
end;
Procedure Zoom_vista(cuad: num_cuadrante; coef: real);
begin
escala[cuad]:= escala[cuad]/coef;
end;
Procedure Varraton(var dc,df: integer; var izq,der: boolean);
var nposrat: pixel;
begin
Libera_raton;
```

```
Situarraton(100,100);
repeat
leerraton(nposrat.c,nposrat.f,izq,der);
if mem[$40:23] and 04<>0 then begin {Cambiamos el modo del raton}
if mraton=isomet then mraton:= libre
else mraton:= succ(mraton);
sentido_raton;
while mem[$40:23] and 04<>0 do;
dibuja_escena(cuad_act);
end;
if keypressed then
case Leertecla of
'+': begin
Zoom_vista(cuad_act,1/coef_zoom);
dibuja_escena(cuad_act);
end;
'−': begin
Zoom_vista(cuad_act,coef_zoom);
dibuja_escena(cuad_act);
end;
end;
until (nposrat.c<>100) or (nposrat.f<>100) or izq or der;
dc:= nposrat.c−100;
df:= nposrat.f−100;
case mraton of
horiz : df:= 0;
```

```
vertic : dc:= 0;
isomet : begin
dc:= dc+df;
df:= dc;
end;
end;
Limita_raton;
end;
Function Pos_vert(pv: p_vertice; lvert: lista_vertices): word;
{Devuelve la posici¢n de un v rtice dentro de una lista}
var n: word;
punt: p_vertice;
begin
n:= 1;
punt:= lvert;
while punt<>pv do begin
n:= n+1;
punt:= punt^.sig;
end;
pos_vert:= n;
end;
Function Vert_pos(n: word; lvert: lista_vertices): p_vertice;
var punt: p_vertice;
i: word;
begin
punt:= lvert;
```

```
for i:= 1 to n−1 do
punt:= punt^.sig;
Vert_pos:= punt;
end;
Procedure Copiar(l_os: l_obj_sel);
var pv,npv,ultvert: p_vertice;
nlvert: lista_vertices;
ultpol,nlpol,npol,pol: lista_poligonos;
ultref,nlref,ref,nref: l_vertpoli;
ob: objeto;
begin
while l_os<>nil do begin
nlvert:= nil;
pv:= l_os^.obj^.lvert;
while pv<>nil do begin
new(npv);
npv^.pos:= pv^.pos;
if nlvert=nil then nlvert:= npv
else ultvert^.sig:= npv;
ultvert:= npv;
pv:= pv^.sig;
end;
ultvert^.sig:= nil;
nlpol:= nil;
pol:= l_os^.obj^.lpoli;
while pol<>nil do begin
```

```
new(npol);
nlref:= nil;
ref:= pol^.lvert;
while ref<>nil do begin
new(nref);
nref^.vert:= Vert_pos( Pos_vert(ref^.vert,l_os^.obj^.lvert), nlvert);
if nlref=nil then nlref:= nref
else ultref^.sig:= nref;
ultref:= nref;
ref:= ref^.sig;
end;
ultref^.sig:= nil;
npol^.lvert:= nlref;
if nlpol=nil then nlpol:= npol
else ultpol^.sig:= npol;
ultpol:= npol;
pol:= pol^.sig;
end;
ultpol^.sig:= nil;
new(ob);
ob^.lvert:= nlvert;
ob^.lpoli:= nlpol;
ob^.sig:= escena;
escena:= ob;
l_os:= l_os^.sig;
end;
```

```
end;
Procedure Escalar_selec(var l_os: l_obj_sel; ref: vector; cuad: num_cuadrante; modo: boolean);
var posrat: pixel;
dc,df,dp: integer;
p_os: l_obj_sel;
pvert: p_vertice;
posrel: vector;
pp: vector;
begin
punterooff;
Posraton(posrat.c,posrat.f);
Trans_WC_PP( ref, ref, cuad);
repeat
dibuja_escena(cuad);
varraton(dc,df,izq,der);
if izq or der then begin
situarraton(posrat.c,posrat.f);
punteroon;
exit;
end;
p_os:= l_os;
while p_os<>nil do begin
pvert:= p_os^.obj^.lvert;
while pvert<>final_lista do begin
Trans_WC_PP( pvert^.pos, pp, cuad);
Dif_vectores( pp, ref, posrel);
```

```
if modo=modo2D then
dp:= 0
else begin
df:= dc;
dp:= dc;
end;
posrel[u]:= posrel[u]*(1+dc/100);
posrel[v]:= posrel[v]*(1+df/100);
posrel[n]:= posrel[n]*(1+dp/100);
Suma_vectores( ref, posrel, pp);
Trans_PP_WC( pp, pvert^.pos, cuad);
pvert:= pvert^.sig;
end;
p_os:= p_os^.sig;
end;
until false;
end;
Procedure Escalar_vertices(l_vs: l_vert_sel; cuad: num_cuadrante);
var dc,df: integer;
vrel: vector;
minx,maxx,miny,maxy,minz,maxz: real;
ref: vector;
p_vs: l_vert_sel;
begin
minx:= maxreal; miny:= maxreal; minz:= maxreal;
maxx:= −maxreal; maxy:= −maxreal; maxz:= −maxreal;
```

```
p_vs:= l_vs;
while p_vs<>nil do begin
with p_vs^.vert^ do begin
minx:= Minimo(minx,pos[x]);maxx:= −Minimo(−maxx, −pos[x]);
miny:= Minimo(miny,pos[y]);maxy:= −Minimo(−maxy, −pos[y]);
minz:= Minimo(minz,pos[z]);maxz:= −Minimo(−maxz, −pos[z]);
end;
p_vs:= p_vs^.sig;
end;
ref[x]:= (minx+maxx)/2;
ref[y]:= (miny+maxy)/2;
ref[z]:= (minz+maxz)/2;
Trans_wc_pp(ref,ref,cuad);
repeat
Dibuja_escena(cuad);
varraton(dc,df,izq,der);
p_vs:= l_vs;
while p_vs<>nil do begin
Trans_wc_pp(p_vs^.vert^.pos, vrel, cuad);
Dif_vectores(vrel, ref, vrel);
vrel[u]:= vrel[u]*(1+dc/100);
vrel[v]:= vrel[v]*(1+df/100);
Suma_vectores(vrel, ref, vrel);
Trans_pp_wc(vrel, p_vs^.vert^.pos,cuad);
p_vs:= p_vs^.sig;
end;
```

```
until izq;
end;
Procedure Rotar_objeto(var o: objeto; ref: vector;alfa: real; cuad: num_cuadrante);
var pvert: p_vertice;
coseno,seno: real;
npp,pp: vector;
begin
coseno:= cos(alfa*Pi/180);
seno:= sin(alfa*Pi/180);
Trans_WC_PP( ref, ref, cuad);
pvert:= o^.lvert;
while pvert<>final_lista do begin
Trans_WC_PP( pvert^.pos, pp,cuad);
Dif_vectores( pp, ref, pp);
Intro_coord( npp, pp[u]*coseno−pp[v]*seno, pp[u]*seno+pp[v]*coseno, pp[n]);
Suma_vectores( npp, ref, npp);
Trans_PP_WC( npp, pvert^.pos,cuad);
pvert:= pvert^.sig;
end;
end;
Procedure Rotar_selec(var l_os: l_obj_sel; ref: vector; cuad: num_cuadrante);
var posrat: pixel;
alfa: real;
pvert: p_vertice;
dc,df: integer;
p_os: l_obj_sel;
```

```
begin
alfa:= 0;
punterooff;
Posraton(posrat.c,posrat.f);
repeat
dibuja_escena(cuad);
Varraton(dc,df,izq,der);
alfa:= dc/5;
p_os:= l_os;
while p_os<>nil do begin
Rotar_objeto(p_os^.obj,ref,alfa,cuad);
p_os:= p_os^.sig;
end;
until izq or der;
situarraton(posrat.c,posrat.f);
punteroon;
end;
Procedure Unir_objetos(var ob1,ob2: objeto);
var pv: p_vertice;
pol: lista_poligonos;
pob: objeto;
begin
pv:= ob1^.lvert;
while pv^.sig<>nil do
pv:= pv^.sig;
pv^.sig:= ob2^.lvert;
```

```
pol:= ob1^.lpoli;
while pol^.sig<>nil do
pol:= pol^.sig;
pol^.sig:= ob2^.lpoli;
if escena=ob2 then begin
escena:= ob2^.sig;
dispose(ob2);
end
else begin
pob:= escena;
while pob^.sig<>ob2 do
pob:= pob^.sig;
pob^.sig:= ob2^.sig;
dispose(ob2);
end;
end;
Procedure Unir_selec(l_os: l_obj_sel);
var p_os,temp: l_obj_sel;
begin
p_os:= l_os^.sig; {Unimos todos los objetos con el primero}
while p_os<>nil do begin
unir_objetos(l_os^.obj,p_os^.obj);
p_os:= p_os^.sig;
end;
p_os:= l_os^.sig; {Deseleccionamos todos los objetos excepto}
while p_os<>nil do begin {el primero }
```

```
temp:= p_os^.sig;
desel_objeto(p_os^.obj,obj_sel);
p_os:= temp;
end;
end;
Procedure Rotar_camara(cuad: num_cuadrante);
var nposrat,posrat: pixel;
dc,df: integer;
alfa,beta: real;
coseno,seno: real;
rot: matriz;
orwc: vector;
begin
punterooff;
repeat
dibuja_escena(cuad);
Trans_pp_wc(cw[cuad],orwc,cuad);
varraton(dc,df,izq,der);
alfa:= −dc*pi/180;
coseno:= cos(alfa);
seno:= sin(alfa);
Intro_coord(rot[x], coseno, −seno, 0);
Intro_coord(rot[y], seno, coseno, 0);
Intro_coord(rot[z], 0, 0, 1);
Prod_matrices(ejes[cuad], rot, ejes[cuad]);
beta:= −df*pi/180;
```

```
coseno:= cos(beta);
seno:= sin(beta);
Intro_coord(rot[u], 1, 0, 0);
Intro_coord(rot[v], 0, coseno, −seno);
Intro_coord(rot[n], 0, seno, coseno);
Prod_matrices(rot, ejes[cuad], ejes[cuad]);
Trans_wc_pp(orwc,cw[cuad],cuad);
until izq or der;
punteroon;
end;
Procedure Desplaza_camara(cuad: num_cuadrante);
var dc,df: integer;
begin
punterooff;
repeat
dibuja_escena(cuad);
varraton(dc,df,izq,der);
cw[cuad][u]:= cw[cuad][u]+dc*escala[cuad];
cw[cuad][v]:= cw[cuad][v]−df*escala[cuad];
until izq or der;
punteroon;
end;
Procedure Centrar_vista(cuad: num_cuadrante; l_obj: lista_objetos);
var pvert: p_vertice;
pp: vector;
pobj: objeto;
```

```
minu,minv,maxu,maxv: real;
eschor,escver: real;
begin
minu:= maxreal;maxu:= −maxreal;minv:= maxreal;maxv:= −maxreal;
pobj:= l_obj;
while pobj<>nil do begin
pvert:= pobj^.lvert;
while pvert<>final_lista do begin
Trans_WC_PP(pvert^.pos,pp,cuad);
minu:= Minimo( minu, pp[u]); maxu:= −Minimo( −maxu, −pp[u]);
minv:= Minimo( minv, pp[v]); maxv:= −Minimo( −maxv, −pp[v]);
pvert:= pvert^.sig;
end;
pobj:= pobj^.sig;
end;
with lim_cuad[cuad] do begin
eschor:= (maxu−minu)/(cmax−cmin−10);
escver:= (maxv−minv)/(fmax−fmin−10);
end;
escala[cuad]:= −Minimo( −eschor, −escver); {escala = maximo (hor,ver) }
cw[cuad][u]:= (maxu + minu)/2;
cw[cuad][v]:= (maxv + minv)/2;
end;
Procedure Restaurar_vista(cuad: num_cuadrante);
begin
escala[cuad]:= escala_ini[cuad];
```

```
ejes[cuad]:= ejes_ini[cuad];
cw[cuad]:= orig_ini[cuad];
end;
Procedure Desplaza_vertice(var pvert: p_vertice; desp: vector);
begin
Suma_vectores(pvert^.pos, desp, pvert^.pos);
end;
Procedure Mover_vertices(var l_vs: l_vert_sel; cuad: num_cuadrante);
var posrat: pixel;
dc,df: integer;
desp: vector;
pvs: l_vert_sel;
begin
punterooff;
Posraton(posrat.c,posrat.f);
repeat
dibuja_escena(cuad);
varraton(dc,df,izq,der);
Intro_coord( desp, dc*escala[cuad], −df*escala[cuad], 0);
Trans_pp_wc( desp, desp, cuad);
pvs:= l_vs;
while pvs<>nil do begin
Desplaza_vertice(pvs^.vert,desp);
pvs:= pvs^.sig;
end;
until izq or der;
```

```
situarraton(posrat.c,posrat.f);
punteroon;
end;
Procedure Mover_selec(var l_os: l_obj_sel; cuad: num_cuadrante);
var posrat: pixel;
pvert: p_vertice;
dc,df: integer;
desp: vector;
pos: l_obj_sel;
begin
punterooff;
Posraton(posrat.c,posrat.f);
repeat
dibuja_escena(cuad);
varraton(dc,df,izq,der);
pos:= l_os;
while pos<>nil do begin
pvert:= pos^.obj^.lvert;
while pvert<>nil do begin
Intro_coord( desp, dc*escala[cuad], −df*escala[cuad], 0);
Trans_pp_wc( desp, desp, cuad);
Desplaza_vertice(pvert,desp);
pvert:= pvert^.sig;
end;
pos:= pos^.sig;
end;
```

```
until izq or der;
punteroon;
end;
Procedure Cambia_cuadrante(var cuad: num_cuadrante);
var xr,yr: integer;
i: num_cuadrante;
begin
posraton(xr,yr);
for i:= 0 to 3 do begin
if (xr>lim_cuad[i].cmin) and (xr<lim_cuad[i].cmax) and
(yr>lim_cuad[i].fmin) and (yr<lim_cuad[i].fmax) then begin
cuad:= i;
exit;
end;
end;
end;
Procedure Selec_cuadro(cuad: num_cuadrante; op: byte);
var p1,p2: pixel;
ct,ft: integer;
obj: objeto;
pv: p_vertice;
pix: pixel;
begin
punteroon;
setcolor(white);
info('Sit£e esquina superior izquierda');
```

```
repeat
leerraton(p1.c,p1.f,izq,der);
until izq or der;
p1.c:= p1.c−lim_cuad[cuad].cmin;
p1.f:= p1.f−lim_cuad[cuad].fmin;
punterooff;
Evita_repet;
info('Sit£e esquina inferior derecha');
repeat
repeat
ct:= p2.c;
ft:= p2.f;
leerraton(p2.c,p2.f,izq,der);
p2.c:= p2.c−lim_cuad[cuad].cmin;
p2.f:= p2.f−lim_cuad[cuad].fmin;
until (ct<>p2.c) or (ft<>p2.f) or izq or der;
dibuja_escena(cuad);
setcolor(white);
rectangle(p1.c,p1.f,p2.c,p2.f);
until izq or der;
if p2.c<p1.c then begin
ct:= p1.c;
p1.c:= p2.c;
p2.c:= ct;
end;
if p2.f<p1.f then begin
```

```
ft:= p1.f;

p1.f:= p2.f;

p2.f:= ft;

end;

obj:= escena;

while obj<>nil do begin

pv:= obj^.lvert;

while pv<>nil do begin

Trans_WC_2d(pv^.pos,pix,cuad);

if (pix.c>p1.c) and (pix.c<p2.c) and (pix.f>p1.f) and (pix.f<p2.f) then begin

if (op=slob) and (not esta_selec_obj(obj)) then begin

selec_objeto(obj,obj_sel);

break;

end;

if (op=dsob) and (esta_selec_obj(obj)) then begin

desel_objeto(obj,obj_sel);

break;

end;

if (op=slvt) and (not esta_selec_vert(pv)) then

selec_vertice(pv,vert_sel);

if (op=dsvt) and (esta_selec_vert(pv)) then

desel_vertice(pv,vert_sel);

end;

pv:= pv^.sig;

end;

obj:= obj^.sig;
```

```
end;

end;

Procedure Selecciona(var pv: p_vertice; var obj: objeto;

cuad: num_cuadrante; var ok: boolean);

var dvert: real;

p: pixel;

pobj: objeto;

ppol: lista_poligonos;

pvert: l_vertpoli;

dminvert: real;

pmin: pixel;

xr,yr: integer;

p0: pixel;

nvert: byte;

pp: vector;

wc: vector;

begin

punteroon;

repeat

leerraton(xr,yr,izq,der);

until izq or der;

Evita_repet;

if der then begin

ok:= false;

exit;

end;
```

```
punterooff;
p0.c:= xr−lim_cuad[cuad].cmin; {Hacemos el rat¢n relativo a la vista}
p0.f:= yr−lim_cuad[cuad].fmin;
dminvert:= maxreal;
pobj:= escena;
while pobj<>nil do begin
ppol:= pobj^.lpoli;
while ppol<>nil do begin
pvert:= ppol^.lvert;
while pvert<>final_lista do begin
Trans_WC_2d(pvert^.vert^.pos,p,cuad);
dvert:= Distancia_2d(p,p0);
if dvert<dminvert then begin
pv:= pvert^.vert;
pmin:= p;
dminvert:= dvert;
obj:= pobj;
end;
pvert:= pvert^.sig;
end;
ppol:= ppol^.sig;
end;
pobj:= pobj^.sig;
end;
punteroon;
if escena=nil then ok:= false
```

```
else ok:= true;
end;
Procedure Presenta_cuadrantes(sel: num_cuadrante);
var i: num_cuadrante;
lim: t_lim_cuad;
begin
setviewport(0,0,639,479,clipon);
punterooff;
setcolor(red);
for i:= 0 to 3 do begin
lim:= lim_cuad[i];
setlinestyle(0,0,1);
setcolor(lightgray);
rectangle(lim.cmin−1,lim.fmin−1,lim.cmax+1,lim.fmax+1);
setcolor(0);
rectangle(lim.cmin−2,lim.fmin−2,lim.cmax+2,lim.fmax+2);
end;
setcolor(red);
rectangle(lim_cuad[sel].cmin−2,lim_cuad[sel].fmin−2,
lim_cuad[sel].cmax+2,lim_cuad[sel].fmax+2);
rectangle(lim_cuad[sel].cmin−1,lim_cuad[sel].fmin−1,
lim_cuad[sel].cmax+1,lim_cuad[sel].fmax+1);
Activa_cuadrante(sel);
punteroon;
end;
Procedure Max_Min(cuad: num_cuadrante);
```

```
var df,dc: integer;
e1,e2: real;
esc: real;
begin
e1:= (maxima.cmax−maxima.cmin)/(lim_cuad_ini[cuad].cmax−lim_cuad_ini[cuad].cmin);
e2:= (maxima.fmax−maxima.fmin)/(lim_cuad_ini[cuad].fmax−lim_cuad_ini[cuad].fmin);
esc:= Minimo( e1, e2);
if maximizado then esc:= 1/esc;
escala[cuad]:= escala[cuad]/esc;
if not maximizado then begin
lim_cuad[cuad]:= maxima;
Presenta_cuadrantes(cuad);
maximizado:= true;
Dibuja_escena(cuad);
end
else begin
lim_cuad[cuad]:= lim_cuad_ini[cuad];
clearviewport;
Presenta_cuadrantes(cuad);
maximizado:= false;
Dibuja_todo(cuad);
end;
end;
Procedure Toroide(cent: vector; R1,R2: integer; n1,n2: byte; cuad: num_cuadrante);
var alfa,beta: real;
i,j: byte;
```

```
pp: vector;
wc: vector;
ult,pv1,pv2: p_vertice;
cara: l_vertpoli;
cont: word;
pva1,pva2: p_vertice;
tor: objeto;
centpp: vector;
begin
Ins_obj_esc(tor);
Trans_WC_PP(cent,centpp,cuad);
for i:= 0 to n1−1 do begin
alfa:= 2*Pi*i/n1+Pi/n1;
for j:= 0 to n2−1 do begin
beta:= 2*Pi*j/n2+Pi/n2;
pp[u]:= centpp[u]+(r1+r2*cos(beta))*cos(alfa);
pp[v]:= centpp[v]+(r1+r2*cos(beta))*sin(alfa);
pp[n]:= centpp[n]+r2*sin(beta);
Trans_PP_WC(pp,wc,cuad);
Ins_vert_obj(pv1,tor,wc);
end;
end;
ult:= tor^.lvert;
while ult^.sig<>nil do
ult:= ult^.sig;
ult^.sig:= tor^.lvert;
```

```
pv1:= tor^.lvert;
pv2:= pv1;
for i:= 1 to n2 do
pv2:= pv2^.sig;
for i:= 1 to n1 do begin
pva1:= pv1;
pva2:= pv2;
for j:= 1 to n2 do begin
cara:= nil;
Ins_vert_poli(pv1,cara);
if j<n2 then begin
Ins_vert_poli(pv1^.sig,cara);
Ins_vert_poli(pv2^.sig,cara);
end
else begin
Ins_vert_poli(pva1,cara);
Ins_vert_poli(pva2,cara);
end;
Ins_vert_poli(pv2,cara);
Ins_poli_obj(cara,tor);
pv1:= pv1^.sig;
pv2:= pv2^.sig;
end;
end;
ult^.sig:= nil;
end;
```

```
Procedure Tronco(cent: vector; R1,R2: integer; h: integer; nv: byte; cuad: num_cuadrante);
var cara,base,tapa: l_vertpoli;
i: byte;
alfa: real;
npv,pv: p_vertice;
tro: objeto;
ptapa: l_vertpoli;
pp: vector;
wc: vector;
centpp: vector;
begin
Ins_obj_esc(tro);
Trans_WC_PP(cent,centpp,cuad);
tapa:= nil;
base:= nil;
for i:= 0 to nv−1 do begin
alfa:= 2*Pi*i/nv+Pi/nv;
pp[u]:= centpp[u]+r1*cos(alfa);
pp[v]:= centpp[v]+r1*sin(alfa);
pp[n]:= centpp[n];
Trans_PP_WC(pp,wc,cuad);
Ins_vert_obj(pv,tro,wc);
Ins_vert_poli(pv,base);
pp[u]:= centpp[u]+r2*cos(alfa);
pp[v]:= centpp[v]+r2*sin(alfa);
pp[n]:= centpp[n]+h;
```

```
Trans_PP_WC(pp,wc,cuad);
Ins_vert_obj(pv,tro,wc);
Ins_vert_poli(pv,tapa);
end;
cara:= nil;
while tapa<>nil do begin
ins_vert_poli(tapa^.vert,cara);
tapa:= tapa^.sig;
end;
ptapa:= tapa;
while ptapa<>nil do begin
tapa:= tapa^.sig;
dispose(ptapa);
ptapa:= tapa;
end;
tapa:= cara;
Ins_poli_obj(tapa,tro);
Ins_poli_obj(base,tro);
pv:= tro^.lvert;
while pv<>nil do begin
cara:= nil;
Ins_vert_poli(pv,cara);
Ins_vert_poli(pv^.sig,cara);
npv:= pv^.sig^.sig;
if npv<>nil then begin
Ins_vert_poli(npv^.sig,cara);
```

```
Ins_vert_poli(npv,cara);
end
else begin
Ins_vert_poli(tro^.lvert^.sig,cara);
Ins_vert_poli(tro^.lvert,cara);
end;
Ins_poli_obj(cara,tro);
pv:= npv;
end;
end;
Procedure Piramide(cent: vector; r: integer; h: integer; nv: byte; cuad: num_cuadrante);
var i: integer;
cara: l_vertpoli;
pp: vector;
wc: vector;
alfa: real;
pv: p_vertice;
base: l_vertpoli;
cima: p_vertice;
pol: lista_poligonos;
j: integer;
pir: objeto;
centpp: vector;
begin
Trans_WC_PP(cent,centpp,cuad);
Ins_obj_esc(pir);
```

```
for i:= nv−1 downto 0 do begin
alfa:= 2*Pi*i/nv+Pi/nv;
pp[u]:= centpp[u]+r*cos(alfa);
pp[v]:= centpp[v]+r*sin(alfa);
pp[n]:= centpp[n];
Trans_PP_WC(pp,wc,cuad);
Ins_vert_obj(pv,pir,wc);
end;
centpp[n]:= centpp[n]+h;
Trans_PP_WC(centpp,wc,cuad);
Ins_vert_obj(cima,pir,wc);
pv:= cima^.sig;
base:= nil;
while pv<>nil do begin
Ins_vert_poli(pv,base);
cara:= nil;
if pv^.sig<>nil then
Ins_vert_poli(pv^.sig,cara)
else
Ins_vert_poli(cima^.sig,cara);
Ins_vert_poli(pv,cara);
Ins_vert_poli(cima,cara);
Ins_poli_obj(cara,pir);
pv:= pv^.sig;
end;
Ins_poli_obj(base,pir);
```

```
end;
Procedure Esfera(cent: vector; r: integer; nv: byte; cuad: num_cuadrante);
var i,j,k: integer;
cosa,sena,cosb,senb: real;
alfa,beta: real;
pp: vector;
wc: vector;
pv,pv2: p_vertice;
lvert: l_vertpoli;
m: byte;
ult: p_vertice;
pol: lista_poligonos;
lv: l_vertpoli;
lvert1,lvert2: l_vertpoli;
pvi,pvs: p_vertice;
pvt1,pvt2: p_vertice;
esf: objeto;
centpp: vector;
begin
Trans_WC_PP(cent,centpp,cuad);
Ins_obj_esc(esf);
m:= (nv div 2) − 1;
for i:= 0 to nv−1 do begin
alfa:= 2*Pi*i/nv;
cosa:= cos(alfa);
sena:= sin(alfa);
```

```
for j:= 1 to m do begin
beta:= 2*Pi*j/nv−Pi/2;
cosb:= cos(beta);
senb:= sin(beta);
pp[u]:= r*cosb*cosa+centpp[u];
pp[v]:= r*cosb*sena+centpp[v];
pp[n]:= r*senb+centpp[n];
Trans_PP_WC(pp,wc,cuad);
Ins_vert_obj(pv,esf,wc);
end;
end;
pvt2:= esf^.lvert;
while pvt2^.sig<>nil do
pvt2:= pvt2^.sig;
ult:= esf^.lvert;
while ult^.sig<>nil do
ult:= ult^.sig;
ult^.sig:= esf^.lvert;
new(pvi);
pp[u]:= centpp[u];
pp[v]:= centpp[v];
pp[n]:= centpp[n]−r;
Trans_PP_WC(pp,wc,cuad);
pvi^.pos:= wc;
new(pvs);
pp[u]:= centpp[u];
```

```
pp[v]:= centpp[v];

pp[n]:= centpp[n]+r;

Trans_PP_WC(pp,wc,cuad);

pvs^.pos:= wc;

lvert:= nil;

pv:= esf^.lvert;

for i:= 0 to nv−1 do begin

pvt1:= pv;

for j:= 1 to m−1 do begin

lvert:= nil;

Ins_vert_poli(pv,lvert);

Ins_vert_poli(pv^.sig,lvert);

pv2:= pv;

for k:= 1 to m do

pv2:= pv2^.sig;

Ins_vert_poli(pv2^.sig,lvert);

Ins_vert_poli(pv2,lvert);

Ins_poli_obj(lvert,esf);

pv:= pv^.sig;

end;

lvert2:= nil;

ins_vert_poli(pvi,lvert2);

ins_vert_poli(pv,lvert2);

ins_vert_poli(pvt2,lvert2);

ins_poli_obj(lvert2,esf);

pvt2:= pv;
```

```
pv:= pv^.sig;
lvert1:= nil;
ins_vert_poli(pvt1,lvert1);
ins_vert_poli(pv,lvert1);
ins_vert_poli(pvs,lvert1);
ins_poli_obj(lvert1,esf);
end;
pvs^.sig:= esf^.lvert;
esf^.lvert:= pvs;
pvi^.sig:= esf^.lvert;
esf^.lvert:= pvi;
ult^.sig:= nil;
end;
Procedure Nuevo_objeto(cuad: num_cuadrante);
var centpp: vector;
centwc: vector;
op: char;
r: integer;
rm: integer;
df,dc: integer;
posrat,nposrat: pixel;
cent2d: pixel;
N,N2: longint;
pvert: p_vertice;
ratio: real;
membaja: boolean;
```

```
mem_nec: longint;
Procedure Poligono(cent: pixel; rad: integer; n: integer);
var alfa: real;
coseno,seno: real;
i: integer;
c,f,nc,nf: real;
begin
setcolor(white);
rad:= abs(rad);
alfa:= Pi*2/n;
coseno:= cos(alfa);
seno:= sin(alfa);
c:= rad*cos(alfa/2);
f:= rad*sin(alfa/2);
for i:= 1 to n do begin
nc:= c*coseno−f*seno;
nf:= c*seno+f*coseno;
line(cent.c+round(c),cent.f+round(f),cent.c+round(nc),cent.f+round(nf));
c:= nc;f:= nf;
end;
{ line(c.c+p.c,c.f+p.f,c.c+pini.c,c.f+pini.f);}
end;
begin
info('1. Esfera 2. Pir mide 3. Tronco 4. Toroide 5. Cubo 6. Cancelar');
repeat
op:= Leertecla;
```

```
until op in ['1'..'6'];
if op='6' then exit;
case op of
'1' : begin
repeat
N:= Cad_Ent(Leer_cadena('N£mero de v rtices por circunferencia (ò4) ',digitos));
mem_nec:= 31*N*N+2*N+56+10240;
membaja:= memavail<mem_nec;
if (N<4) or membaja then write(#7);
if membaja then begin
Info('No hay suficiente memoria: pruebe con menos v rtices. Pulse una tecla');
Leertecla;
end;
until (N>=4) and (not membaja);
end;
'2' : begin
repeat
N:= Cad_Ent(Leer_cadena('N£mero de v rtices de la base: ',digitos));
mem_nec:= 62*N+42+10240;
membaja:= memavail<mem_nec;
if (N<3) or membaja then write(#7);
if membaja then begin
Info('No hay suficiente memoria: pruebe con menos v rtices. Pulse una tecla');
Leertecla;
end;
until (N>=3) and (not membaja);
```

```
end;
'3' : begin
repeat
N:= Cad_Ent(Leer_cadena('N£mero de v rtices de las caras superior e inferior: ',digitos));
mem_nec:= 100*N+28+10240;
membaja:= memavail<mem_nec;
if (N<3) or membaja then write(#7);
if membaja then begin
Info('No hay suficiente memoria: pruebe con menos v rtices. Pulse una tecla');
Leertecla;
end;
until (N>=3) and (not membaja);
end;
'4' : begin
repeat
repeat
N:= Cad_Ent(Leer_cadena('N£mero de v rtices del per¡metro: ',digitos));
if N<3 then write(#7);
until N>=3;
repeat
N2:= Cad_Ent(Leer_cadena('N£mero de v rtices de la secci¢n: ',digitos));
if N2<3 then write(#7);
until N2>=3;
mem_nec:= 182*N*N2+12+10240;
membaja:= memavail<mem_nec;
if membaja then begin
```

```
write(#7);
Info('No hay suficiente memoria: pruebe con menos v rtices. Pulse una tecla');
Leertecla;
end;
until not membaja;
end;
'5' : begin
N:= 4;
mem_nec:= 100*N+28+10240;
membaja:= memavail<mem_nec;
if membaja then begin
Info('No hay suficiente memoria');
exit;
end;
end;
end;
info('Sit£e el centro de la figura');
Punteroon;
repeat
leerraton(posrat.c,posrat.f,izq,der);
until izq or der;
Evita_repet;
cent2d.c:= posrat.c−lim_cuad[cuad].cmin;
cent2d.f:= posrat.f−lim_cuad[cuad].fmin;
Trans_2d_PP(cent2d,centpp,cuad);
punterooff;
```

```
info('Determine el tama¤o base');
r:= 0;
repeat
Dibuja_escena(cuad);
Poligono(cent2d,r,N);
Varraton(dc,df,izq,der);
r:= r+dc;
until izq or der;
Evita_repet;
r:= abs(round(r*escala[cuad])); {Radio en el coordenadas reales}
Trans_pp_wc(centpp,centwc,cuad);
case op of
'1': Esfera(centwc,r,N,cuad);
'2': Piramide(centwc,r,round(2*r),N,cuad);
'3','4': begin
rm:= 0;
repeat
Dibuja_escena(cuad);
Poligono(cent2d,round(r/escala[cuad]),N);
Poligono(cent2d,rm,N);
Varraton(dc,df,izq,der);
rm:= rm+dc;
until izq or der;
Evita_repet;
rm:= abs(round(rm*escala[cuad]));
if op='3' then Tronco(centwc,r,rm,2*r,N,cuad)
```

```
else Toroide(centwc,round((r+rm)/2),round((r−rm)/2),N,N2,cuad);
end;
'5': Tronco(centwc,r,r,trunc(r*sqrt(2)),4,cuad);
end;
Dibuja_todo(cuad);
Libera_raton;
punteroon;
end;
Procedure Cargar_escena(var esc: lista_objetos);
var obj: objeto;
ultvert,vert: p_vertice;
nobj: word;
npol: word;
nvert: word;
f: file;
pos_nvert,pos_npol: longint;
i,j: word;
com_vert: longint;
punt: longint;
fic: string;
lvert: lista_vertices;
pol,ultpol,lpol: lista_poligonos;
vertpol,ultvertpol,lvertpol: l_vertpoli;
n: word;
error: byte;
begin
```

```
repeat
fic:= Leer_cadena('Nombre del fichero a cargar (Enter=cancelar): ',caracteres);
if fic='' then exit;
assign(f,fic);
{$I−}
reset(f,1);
{$I+}
error:= ioresult;
if error<>0 then begin
write(#7);
info('El fichero no existe. Pulse una tecla');
Leertecla;
end;
until error=0;
reset(f,1);
while not eof(f) do begin
Ins_obj_esc(obj);
lvert:= nil;
blockread(f,nvert,sizeof(nvert));
com_vert:= filepos(f);
for i:= 1 to nvert do begin
new(vert);
if lvert=nil then lvert:= vert
else ultvert^.sig:= vert;
ultvert:= vert;
blockread(f,vert^.pos,sizeof(vert^.pos));
```

```
end;
ultvert^.sig:= nil;
obj^.lvert:= lvert;
lpol:= nil;
blockread(f,npol,sizeof(npol));
for i:= 1 to npol do begin
new(pol);
if lpol=nil then lpol:= pol
else ultpol^.sig:= pol;
ultpol:= pol;
lvertpol:= nil;
blockread(f,nvert,sizeof(nvert));
for j:= 1 to nvert do begin
new(vertpol);
if lvertpol=nil then lvertpol:= vertpol
else ultvertpol^.sig:= vertpol;
ultvertpol:= vertpol;
blockread(f,n,sizeof(n));
vertpol^.vert:= vert_pos(n,lvert);
end;
ultvertpol^.sig:= nil;
pol^.lvert:= lvertpol;
end;
ultpol^.sig:= nil;
obj^.lpoli:= lpol;
end;
```

```
close(f);
for i:= 0 to 3 do
centrar_vista(i,esc);
end;
Procedure Grabar_escena(esc: lista_objetos);
var obj: objeto;
vertpol: l_vertpoli;
vert: p_vertice;
pol: lista_poligonos;
nobj: word;
npol: word;
nvert: word;
f: file;
pos_nvert,pos_npol: longint;
fic: string;
error: byte;
n: word;
begin
repeat
fic:= Leer_cadena('Nombre del fichero a grabar: ',caracteres);
assign(f,fic);
{$I−}
reset(f,1);
{$I+}
error:= ioresult;
if error=0 then begin
```

```
write(#7);
info('El fichero ya existe. ¨Desea reemplazarlo?');
if upcase(Leertecla)='S' then error:= 1;
end;
until error<>0;
rewrite(f,1);
obj:= esc;
while obj<>nil do begin
nvert:= 0;
pos_nvert:= filepos(f);
blockwrite(f,nvert,sizeof(nvert));
vert:= obj^.lvert;
while vert<>nil do begin
blockwrite(f,vert^.pos,sizeof(vert^.pos));
nvert:= nvert+1;
vert:= vert^.sig;
end;
seek(f,pos_nvert);
blockwrite(f,nvert,sizeof(nvert));
seek(f,filesize(f));
pos_npol:= filepos(f);
npol:= 0;
blockwrite(f,npol,sizeof(npol));
pol:= obj^.lpoli;
while pol<>nil do begin
pos_nvert:= filepos(f);
```

```
nvert:= 0;
blockwrite(f,nvert,sizeof(nvert));
vertpol:= pol^.lvert;
while vertpol<>nil do begin
nvert:= nvert+1;
n:= Pos_vert(vertpol^.vert,obj^.lvert);
blockwrite(f,n,sizeof(n));
vertpol:= vertpol^.sig;
end;
seek(f,pos_nvert);
blockwrite(f,nvert,sizeof(nvert));
seek(f,filesize(f));
npol:= npol+1;
pol:= pol^.sig;
end;
seek(f,pos_npol);
blockwrite(f,npol,sizeof(npol));
seek(f,filesize(f));
obj:= obj^.sig;
end;
end;
var i: byte;
obj,obj2: objeto;
opcion,nop: byte;
posrat,nposrat,postemp: pixel;
pvert: p_vertice;
```

```
ok: boolean;
cm: vector;
selo_temp: l_obj_sel;
selv_temp: l_vert_sel;
conj_vistas: set of num_cuadrante;
begin
writeln('3DEDIT. Realizado por Agust¡n Caballero Belda');
writeln;
Initevents;
if not mouseevents then begin
write(#7);
writeln;
writeln('Rat¢n no detectado');
writeln('Por favor, corrija el problema e intente de nuevo');
doneevents;
halt;
end;
inicia_graf;
setbkcolor(blue);
setfillstyle(1,lightblue);
bar(maxima.cmax,maxima.fmin,640,480);
bar(0,maxima.fmax,maxima.cmax,480);
punteroon;
maximizado:= false;
lim_cuad:= lim_cuad_ini;
ejes:= ejes_ini;
```

```
escala:= escala_ini;

cw:= orig_ini;

escena:= nil;

Poner_menus;

obj_sel:= nil;

vert_sel:= nil;

cuad_act:= 0;

Presenta_cuadrantes(cuad_act);

Activa_cuadrante(cuad_act);

{ for i:= 0 to 3 do

zoom_vista( i, 100);}

info('3DEDIT Programado por Agust¡n Caballero Belda');

repeat

setviewport(0,0,639,479,clipon);

libera_raton;

mraton:= libre;

Evita_repet;

punteroon;

repeat

leerraton(posrat.c,posrat.f,izq,der);

if posrat.c>560 then begin

nop:= 0;

for i:= 1 to numitems do

if (posrat.f−10) div 15 = menu[i].fila then

nop:= i;

end
```

```
else nop:= 0;
if (nop<>opcion) then
cambia_opcion(opcion,nop);
opcion:= nop;
until ( (izq or der) and ((opcion<>0) or ((posrat.c<560) and (posrat.f<460))));
Activa_cuadrante(cuad_act);
Evita_repet;
punterooff;
if (izq or der) then begin
if posrat.c>560 then begin {Si es men£}
posraton(posrat.c,posrat.f);
if ((escena=nil) and (not (opcion in [alej,acer,roca,desp,rest,eje,mxmn,nuev,carg,salir]))) then begin
info('Comando no ejecutable sobre una escena vac¡a');
write(#7);
end
else if (obj_sel=nil) and der and (opcion in [elim,mvob,es2D,es3D,rota,copi,unio]) then begin
info('Comando no ejecutable. No se ha seleccionado ning£n objeto');
write(#7);
end
else if (vert_sel=nil) and ((der and (opcion=mvvt)) or (opcion=esvt)) then begin
info('Comando no ejecutable. No se ha seleccionado ning£n v rtice');
write(#7);
end
else
case opcion of
acer,alej,cent,rest,eje: begin
```

```
if izq then conj_vistas:= [cuad_act]
else conj_vistas:= [0..3];
for i:= 0 to 3 do
if i in conj_vistas then begin
case opcion of
acer: Zoom_vista(i,1/coef_zoom);
alej: Zoom_vista(i,coef_zoom);
cent: Centrar_vista(i,escena);
rest: Restaurar_vista(i);
eje: ej_vis[i]:= not ej_vis[i];
end;
if maximizado then begin
if i=cuad_act then
dibuja_escena(cuad_act);
end
else
dibuja_escena(i);
end;
end;
mxmn: Max_min(cuad_act);
nuev: begin
Limita_raton;
Nuevo_objeto(cuad_act);
end;
esvt: Escalar_vertices( vert_sel, cuad_act);
mvvt: begin
```

```
Limita_raton;
if not der then
repeat
Evita_repet;
info('Izq: Elegir v rtice Der: Cancelar');
Selecciona(pvert,obj,cuad_act,ok);
if ok then begin
new(selv_temp);
selv_temp^.sig:= nil;
selv_temp^.vert:= pvert;
Mover_vertices(selv_temp,cuad_act);
Dibuja_todo(cuad_act);
Desel_vertice(pvert,selv_temp);
end
until not ok
else begin
Mover_vertices(vert_sel,cuad_act);
Dibuja_todo(cuad_act);
end;
end;
mvob,es2D,es3D,rota,copi,elim,slob,dsob : begin
Limita_raton;
if izq then
repeat
Evita_repet;
info('Izq: Elegir objeto Der: Cancelar');
```

```
Selecciona(pvert,obj,cuad_act,ok);
if ok then begin
new(selo_temp);
selo_temp^.sig:= nil;
selo_temp^.obj:= obj;
Centro_caja(selo_temp,cm);
case opcion of
mvob: Mover_selec(selo_temp,cuad_act);
es2D: Escalar_selec(selo_temp,cm,cuad_act,modo2D);
es3D: Escalar_selec(selo_temp,cm,cuad_act,modo3D);
rota: Rotar_selec(selo_temp,cm,cuad_act);
copi: begin
Copiar(selo_temp);
Mover_selec(selo_temp,cuad_act);
end;
elim: begin
Eliminar_objeto(obj);
if Esta_selec_obj(obj) then Desel_objeto(obj,obj_sel);
end;
slob: if not Esta_selec_obj(obj) then Selec_objeto(obj,obj_sel);
dsob: if Esta_selec_obj(obj) then Desel_objeto(obj,obj_sel);
end;
Desel_objeto(obj,selo_temp);
dibuja_todo(cuad_act);
end
until not ok
```

```
else begin
Centro_caja(obj_sel,cm);
case opcion of
mvob: Mover_selec(obj_sel,cuad_act);
es2D: Escalar_selec(obj_sel,cm,cuad_act,modo2D);
es3D: Escalar_selec(obj_sel,cm,cuad_act,modo3D);
rota: Rotar_selec(obj_sel,cm,cuad_act);
copi: begin
Copiar(obj_sel);
Mover_selec(obj_sel,cuad_act);
end;
elim: begin
info('Se eliminar n todos los objetos seleccionados. ¨Confirma?');
if upcase(Leertecla)='S' then
while obj_sel<>nil do begin
Eliminar_objeto(obj_sel^.obj);
Desel_objeto(obj_sel^.obj,obj_sel);
end;
end;
dsob,slob: Selec_cuadro(cuad_act,opcion);
end;
dibuja_todo(cuad_act);
end;
end;
unio: begin
if not der then
```

```
Repeat
Evita_repet;
Limita_raton;
info('Izq: Elegir primer objeto Der: Cancelar');
Selecciona(pvert,obj,cuad_act,ok);
if ok then begin
info('Izq: Elegir segundo objeto Der: Cancelar');
repeat
Selecciona(pvert,obj2,cuad_act,ok);
if obj=obj2 then
info('No se puede unir un objeto consigo mismo');
until (obj<>obj2) or (not ok);
if ok then Unir_objetos(obj,obj2);
end;
until not ok
else begin
Unir_selec(obj_sel);
end;
end;
slvt,dsvt: begin
Limita_raton;
if izq then
repeat
Evita_repet;
info('Izq: Elegir v rtice Der: Cancelar');
Selecciona(pvert,obj,cuad_act,ok);
```

```
if ok then begin
if (opcion=slvt) and (not Esta_selec_vert(pvert)) then
Selec_vertice(pvert,vert_sel);
if (opcion=dsvt) and (Esta_selec_vert(pvert)) then
Desel_vertice(pvert,vert_sel);
Dibuja_todo(cuad_act);
end;
until not ok
else
Selec_cuadro(cuad_act,opcion);
end;
carg: begin
info('La escena actual ser  eliminada. ¨Confirma?');
if (escena=nil) or (upcase(Leertecla)='S') then begin
eliminar_escena(escena);
for i:= 0 to 3 do
Restaurar_vista(i);
Cargar_escena(escena);
Dibuja_todo(cuad_act);
end;
end;
mezc: begin
Cargar_escena(escena);
Dibuja_todo(cuad_act);
end;
grab: Grabar_escena(escena);
```

```
nues: begin
info('La escena actual ser  eliminada. ¨Confirma?');
if upcase(Leertecla)='S' then begin
Eliminar_escena(escena);
end;
for i:= 0 to 3 do
Restaurar_vista(i);
dibuja_todo(cuad_act);
end;
desp: Desplaza_camara(cuad_act);
roca: rotar_camara(cuad_act);
salir: begin
info('¨Seguro que desea salir?');
if upcase(Leertecla)='S' then begin
Eliminar_escena(escena);
Doneevents;
restorecrtmode;
halt;
end;
end;
end;
Libera_raton;
situarraton(posrat.c,posrat.f);
end
else if not maximizado then
begin
```

Cambia_cuadrante(cuad_act);

Presenta_cuadrantes(cuad_act);

end;

end;

cambia_opcion(opcion,opcion);

until false;

end.

3D EDIT

Calasparra (Murcia)

acebe@ctv.es

Manual de usuario

El programa arranca con la instrucci¢n 3dedit, y

requiere para su uso de un rat¢n. Al arrancar, se comprueba

si el rat¢n est efectivamente instalado y en funcionamiento;

si hay alg£n problema, se le pedir que solucione el problema

y pruebe de nuevo. Si todo ha ido bien, aparece la pantalla

siguiente:

Podemos distinguir tres zonas en esta pantalla:

− Zona de visualizaci¢n. Esta zona est dividida en

cuatro cuadrantes, cada una de las cuales representan una de

las posibles vistas que se pueden presentar simult neamente.

Inicialmente, el primer cuadrante representa la planta, el

segundo el alzado, el tercero el perfil, y el cuarto una

vista tridimensional en proyecci¢n isom trica.

− Zona de mensajes. Est situada en la parte inferior de

la pantalla; en esta zona ir n apareciendo todos los mensajes

que el programa emita, tanto los de ayuda como los que requieran alg£n tipo de entrada por parte del usuario.

− Zona de men£s. En la parte derecha de la pantalla tenemos una serie de opciones divididas en varios submen£s seg£n a qu entidad est n referidos (vista, objeto, v rtice ¢ escena).

El manejo del programa se realiza enteramente mediante el rat¢n (excepto las entradas que requieran datos num ricos o alfanum ricos, l¢gicamente), utiliz ndose los botones izquierdo y derecho.

Inicialmente, la vista seleccionada es la superior izquierda (primer cuadrante); esto se indica porque el marco que rodea a dicho cuadrante est en color rojo. Esta vista ser sobre la que se ejecuten todos los comandos; si deseamos seleccionar cualquier otra, bastar con pulsar sobre la vista correspondiente.

Para ejecutar uno de los comandos del men£, basta con desplazar el rat¢n sobre la opci¢n correspondiente, y pulsar uno de los botones. En general, el bot¢n izquierdo se utiliza para ejecutar un comando sobre una £nica entidad, mientras que el derecho indica la ejecuci¢n m£ltiple, cuando esta opci¢n est disponible.

A continuaci¢n veamos el prop¢sito de cada uno de los comandos.

Menu Vista

Acercar. Realiza un acercamiento del punto de vista hacia la

escena, con lo que se produce el efecto de un zoom de ampliaci¢n de la escena.

Alejar. Aleja la posici¢n del observador de la escena, por lo que la escena se har m s peque¤a.

Desplazar. Permite desplazar el punto de observaci¢n paralelamente al plano de proyecci¢n. Para ello, tras pulsar sobre la opci¢n, podremos realizar dicho desplazamiento simplemente moviendo el raton. Para terminar, simplemente hay que pulsar un bot¢n.

Centrar. Ajusta la escena de forma que se visualice toda ella en la vista determinada.

Max/Min. Como antes comentamos, inicialmente aparecen cuatro cuadrantes en pantalla; esto puede ser modificado con este comando, haciendo que la vista seleccionada ocupe toda la zona de visualizaci¢n, o haciendo que vuelva a su modo inicial.

Rotar. Realiza una rotaci¢n del punto de observaci¢n alrededor de la escena, manteniendo la direcci¢n de observaci¢n. Desplazando el rat¢n horizontalmente, giramos respecto del eje Z, mientras que si el movimiento es vertical, la rotaci¢n se hace inclinando m s o menos el plano de proyecci¢n hacia la posici¢n del observador.

Restaurar. Restablece las caracter¡sticas iniciales de las vistas, restaurando la posici¢n de la c mara, los ejes, etc. Esto es £til cuando queramos regresar al esquema tradicional de planta, perfil y alzado.

Ejes. Hace que sean visibles u ocultos los ejes X, Y y Z.

Estos ejes saldr n en l¡nea discontinua y en color amarillo.

Las opciones Acercar, Alejar, Centrar, Restaurar y Ejes admiten actuaci¢n m£ltiple, es decir, si ejecutamos uno de estos comandos pulsando el bot¢n derecho del rat¢n, la acci¢n se realizar sobre las cuatro vistas.

Men£ Objeto

Nuevo. Permite la creaci¢n de un nuevo objeto, siendo ste de uno de los siguientes tipos predefinidos: esfera, pir mide, tronco, toroide y caja; se solicita al usuario que elija uno de estos tipos. Todos los objetos se crear n coloc ndolo sobre el plano de proyecci¢n. Veamos c¢mo se procede para crear un objeto de una clase:

1. Esfera. El usuario debe introducir por teclado el n£mero de v rtices que tendr cada uno de los "meridianos" de la esfera; posteriormente, se debe dar gr ficamente el tama¤o del radio de la esfera. El objeto creado consiste en una malla con forma esf rica compuesta de pol¡gonos de cuatro aristas en general y de tres aristas para aqu llos a los que pertenece cada uno de los "polos".

2. Pir mide. En este caso, el valor num rico que se pide al usuario es el n£mero de v rtices que componen la base, introduciendo el radio gr ficamente. El objeto creado es una pir mide que tendr por defecto una altura igual al di metro de la base.

3. Tronco. Un tronco es un objeto resultado de eliminar

de una pir mide la parte superior. As¡ pues, adem s de el n£mero de v rtices de las bases superior e inferior (l¢gicamente ser el mismo), se deben determinar gr ficamente sus respectivos radios.

4. Toroide. Un toroide es el resultado de realizar el recorrido de un pol¡gono sobre el camino marcado por otro; as¡ pues, debemos determinar la forma de ambos pol¡gonos. Primero se introduce el n£mero de v rtices del pol¡gono que marcar el camino, y posteriormente el del pol¡gono que forma la secci¢n. Gr ficamente determinamos los radios m ximo y m¡nimo del toroide, de tal forma que el di metro del pol¡gono de la secci¢n ser la diferencia entre ambos radios.

5. Cubo. La forma de este tipo de objetos es la obvia; el £nico par metro necesario es el tama¤o del mismo, que se introducir gr ficamente.

Al crear un objeto, si aumentamos el n£mero de v rtices ste se parecer m s a una figura perfecta. As¡, la esfera se aproximar a una esfera perfecta, la pir mide a un cono, el tronco a un cono truncado y el toroide a un anillo circular con secci¢n tambi n circular. No obstante, este n£mero no puede ser todo lo grande que se desee, por cuestiones de espacio en memoria. Para delimitar este n£mero se realiza un c lculo aproximado de la memoria que precisar el objeto, y si no hay bastante disponible se da un mensaje por pantalla para que se disminuya el valor.

Selecci¢n. Permite la selecci¢n de uno o varios objetos; en

caso de ser selecci¢n simple (bot¢n izquierdo), cada vez que pulsemos el bot¢n izquierdo seleccionaremos el objeto m s cercano, entendiendo por ste aqu l que contiene al v rtice m s cercano al puntero; para terminar se pulsa el bot¢n derecho. Si hemos optado por selecci¢n m£ltiple (bot¢n derecho), sta se hace creando una ventana, y ser n seleccionados todos los objetos que tengan alg£n v rtice dentro de la misma. Todos los objetos seleccionados aparecer n en pantalla en color verde hasta que sea deseleccionado.

Deselec. Este comando es el opuesto al anterior, y funciona de forma completamente id ntica.

Eliminar. Con este comando, en modo sencillo iremos indicando con el bot¢n izquierdo los objetos que se van eliminando, finalizando con el derecho. En modo m£ltiple, ser n eliminados directamente todos los objetos que est n seleccionados.

Mover. Si estamos en modo sencillo, elegimos los objetos a mover y los desplazamos con el rat¢n, deposit ndolos con el bot¢n izquierdo; para terminar, se pulsa el bot¢n derecho. En modo m£ltiple se mover n de forma conjunta los objetos seleccionados. Se¤alar que todos los movimientos se realizan l¢gicamente de forma paralela al plano de proyecci¢n.

Escalar. Se utiliza esta opci¢n para cambiar el tama¤o de un objeto o conjunto de objetos; este escalado se hace respecto del punto central de la caja que encierra al objeto u objetos

correspondientes. En modo 2D cambiamos el tama¤o de la proyecci¢n seleccionada £nicamente, mientras que en 3D se cambia el tama¤o en todas las dimensiones.

Rotar. Realiza una rotaci¢n de un objeto u objetos sobre el plano de proyecci¢n; al igual que el caso anterior, el punto respecto del cual se rota es el centro de la caja que encierra al conjunto.

Copiar. Usamos esta opci¢n para realizar un duplicado de un objeto u objetos, permitiendo que la copia creada sea trasladada de lugar para evitar que se solape con el original.

Unir. Este comando se utiliza cuando deseamos que una serie de objetos pasen a ser £nicamente uno y sean tratados como tal. En el caso del modo sencillo, se supone que queremos unir dos objetos, con lo cual los indicamos mediante el rat¢n y se ejecuta la acci¢n; en caso m£ltiple, la uni¢n se realiza entre todos los objetos seleccionados.

Men£ V rtice

Selecci¢n. Al igual que en el caso de los objetos, podemos seleccionar v rtices individualmente o mediante una ventana; en el primer caso, el v rtice seleccionado es el m s cercano al puntero del rat¢n, y en el segundo se seleccionan todos aquellos que caigan dentro de dicha ventana. Un v rtice seleccionado se distingue porque aparece un peque¤o c¡rculo rojo alrededor de l.

Deselec. Realiza la acci¢n contraria a la anterior, de forma

completamente an loga.

Mover. Permite el movimiento de un v rtice ¢ conjunto de v rtices, de id ntica forma a la vista hasta ahora: bien eligiendo uno de forma individual o bien moviendo todos los seleccionados conjuntamente.

Escala. Realiza un escalado de los v rtices de forma similar a la de un objeto, es decir, tomando como punto de referencia el centro del conjunto de v rtices. Esta opci¢n, como es l¢gico, se realiza sobre el conjunto de v rtices seleccionados.

Men£ Escena

Cargar. Lee un fichero de disco y carga la escena en memoria. Para ello, se solicita el nombre del fichero a cargar y se saca por pantalla la escena. Respecto al nombre del fichero, ste puede tener cualquier nombre v lido para M.S.−D.O.S.; se puede especificar cualquier extensi¢n e incluso la ruta del fichero. En el manual t cnico se explica en profundidad el formato de los ficheros.

Grabar. Salva la escena actual a disco, pidiendo por teclado el nombre destino; en caso de que el fichero ya exista, se pide confirmaci¢n para sobreescribirlo.

Nueva. Elimina de memoria la escena devolviendo el programa al estado inicial, pidiendo previamente confirmaci¢n por teclado.

Mezclar. Realiza la misma funci¢n que cargar, con la diferencia de que no elimina la escena actual, sino que la

mantiene y une ambas escenas. Esta opci¢n es muy £til cuando queremos dise¤ar una escena descomponi ndola en varias subescenas m s peque¤as.

Cuando entramos en una opci¢n que permite movimiento, rotaci¢n, etc. mediante el rat¢n, existe la posibilidad de fijar el movimiento del mismo. Esto se puede variar pulsando la tecla Ctrl; cada vez que se pulse, aparecer en el centro del cuadrante un peque¤o icono que indica el modo actual de funcionamiento. Los modos son libre, vertical, horizontal y diagonal. El estado inicial del rat¢n es el de movimiento libre, y sucesivamente se va pasando cada uno de los siguientes. Esta limitaci¢n del rat¢n se puede utilizar cuando por ejemplo queremos mover un objeto en horizontal. La utilidad de la £ltima opci¢n se manifiesta cuando cambiamos de tama¤o un objeto y queremos que mantengan la proporci¢n entre el eje horizontal y el vertical.

Lista de polígonos

Lista de vértices

N

6

4

2

5

3

1

v

CW

PP

u