



D-10937



1
001.6425
A 945

ESCUELA SUPERIOR **POLITECNICA** DEL LITORAL

FACULTAD DE INGENIERIA EN ELECTRICIDAD

“**DISEÑO** E IMPLEMENTACION DE LIBRERIAS PARA
GRAFICAS EN TRES DIMENSIONES EN PASCAL”

TESIS DE GRADO

Previa a la Obtención del Título de:

INGENIERO EN COMPUTACION

Presentada por:

Adriana Avalos Miranda

A G R A D E C I M E N T O

AL ING. SIXTO GARCIA A.

D E D I C A T O R I A

A GLADYS Y GUALBERTO

D E C L A R A C I O N E X P R E S A

"La responsabilidad por los hechos, ideas y doctrinas expuestos en esta tesis, me corresponden exclusivamente; y, el patrimonio intelectual de la misma, a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL".

(Reglamento de Exámenes y Titulos profesionales de la ESPOL)

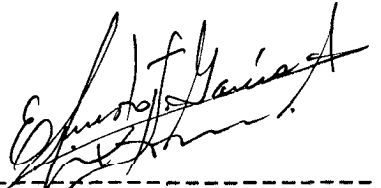


ADRIANA AVALOS MIRANDA

MIEMBROS DEL TRIBUNAL DE GRADO



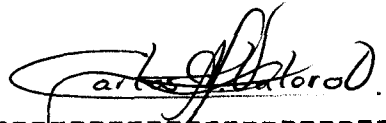
ING. JORGE FLORES M.
SUB-DECANO



ING. SIXTO GARCIA A.
DIRECTOR DE TESIS



ING. JAIME PUENTE P.
MIEMBRO PRINCIPAL



ING. CARLOS VALERO D.
MIEMBRO PRINCIPAL

RESUMEN

El presente trabajo consiste en proporcionar un sistema gráfico tridimensional que sirva como un instrumento útil para aplicaciones futuras.

Este paquete está formado por cinco librerías gráficas accesibles y de gran facilidad para el programador. Es así que se ha implementado una aplicación del sistema de gráficas la misma que utiliza una GRUA - ROBOT cuyos movimientos son controlados desde cualquier computador personal perteneciente a la familia IBM-PC, para lo cual se utilizó también una interfase digital que permite establecer la comunicación entre el computador y la GRUA.

Los movimientos del dispositivo mecánico son: girar en su propio eje, bajar y subir un gancho y contraer o expandir un brazo. Dichos movimientos físicos son controlados por el teclado del computador, a la vez se despliega la imagen de la animación de la GRUA en tres dimensiones en la pantalla de video del computador.

INDICE GENERAL

RESUMEN	Pág. VI
INDICE GENERAL	VII
INDICE DE FIGURAS	X
INDICE DE TABLAS	XV
INTRODUCCION	16
1 PANORAMA GENERAL DE LOS SISTEMAS GRAFICOS	18
1.1 GENERALIDADES	18
1.2 DISPOSITIVOS DE DESPLIEGUE	18
1.3 DISPOSITIVOS DE COPIA DURA	34
1.4 DISPOSITIVOS DE ENTRADA INTERACTIVOS	38
1.5 PROCESADOR DE DESPLIEGUES	39
1.6 SOFTWARE DE GRAFICAS	49
II TRANSFORMACIONES BIDIMENSIONALES	52
2.1 TRASLACIONES	52
2.2 ROTACIONES	54
2.3 ESCALAMIENTO	59
2.4 AFILAMIENTO	62
2.5 TRANSFORMACIONES INVERSAS	65
2.6 MATRICES	65
2.7 REPRESENTACION MATRICIAL DE LAS TRANSFORMACIONES	67
2.8 COMBINACION DE TRANSFORMACIONES	69



	Pág.
2.9 PROCEDIMIENTO DE TRANSFORMACIONES	72
III REPRESENTACIONES TRIDIMENSIONALES	74
3.1 INTRODUCCION	74
3.2 SISTEMAS DE COORDENADAS	75
3.3 SUPERFICIES DE POLIGONOS	77
3.4 SUPERFICIES CURVAS	83
IV TRANSFORMACIONES TRIDIMENSIONALES	90
4.1 TRASLACION	90
4.2 ROTACION	91
4.3 ESCALAMIENTO	94
4.4 ROTACION EN TORNO A UN EJE ARBITRARIO	95
4.5 REFLEXIONES	103
4.6 TRANSFORMACION DE SISTEMAS DE COORDENADAS	103
V VISTA TRIDIMENSIONAL	100
5.1 PROYECCIONES	100
5.2 TRANSFORMACION DE LA VISION	112
VI SUSPRESION DE SUPERFICIES Y LINEAS OCULTAS	126
6.1 GENERALIDADES	126
6.2 CLASIFICACION DE ALGORITMOS	126
6.3 SUPRESION DE LA CARA ANTERIOR	127
6.4 METODO DEL BUFFER CON PROFUNDIDAD	131
6.5 METODO DE LA LINEA DE RASTREO	136
6.6 ELIMINACION DE LINEAS OCULTAS	138

VII	DISEÑO DE LIBRERIAS	140
7.1	OBJETIVOS DE LA PLANEACION	140
7.2	METODOLOGIA PARA LA CLASIFICACION DE APLICACIONES	142
7.3	PROGRAMACION DE VISTAS TRIDIMENSIONALES	148
7.4	EXTENSIONES DEL MODELO TRIDIMENSIONAL	149
VIII	DISEÑO DE LA INTERFASE DEL USUARIO	155
8.1	COMPONENTES DE LA INTERFAZ DEL USUARIO	155
8.2	MODELO DEL USUARIO	156
8.3	LENGUAJE DE COMANDO	157
8.4	DISEÑO DEL MENU	158
8.5	FORMATO DE SALIDA	159
IX	APLICACION DEL SISTEMA DE ANIMACION DE GRAFICAS	161
9.1	OBJETIVOS DEL DISEÑO	161
9.2	DESCRIPCION DEL SISTEMA QUE CONTROLA EL DISPOSITIVO EXTERNO MOVIL	162
9.3	MODELO DEL USUARIO	164
	CONCLUSIONES Y RECOMENDACIONES	166
	APENDICE A	168
	APENDICE B	203
	BIBLIOGRAFIA	245

INDICE DE FIGURAS

Pág.

CAPITULO 1

1.2.1	DISEÑO BASICO DE UN CRT	20
1.2.2	OPERACION DE UN DISPARADOR DE ELECTRONES EN UN CRT.....	20
1.2.3	SISTEMA DE RASTREO AL AZAR	24
1.2.4	SISTEMA DE RASTREO CON RASTREADOR	24
1.2.5	PROGRAMACION DEL MONITOR DE RAS - TREO CON RASTREADOR	26
1.2.6	OPERACION DE UN CRT CON MASCARA DE SOMBRA	29
1.2.7	OPERACION DE UN DVST	32
1.2.8	OPERACION DE UN SISTEMA TRIDI - MENSIONAL	35
1.5.1	DIAGRAMA DE HARDWARE DE UN CIRCUITO DE DESPLIEGUE	41
1.5.2	DIAGRAMA DE LAS OPERACIONES LOGICAS QUE REALIZA UN SISTEMA DE RASTREO AL AZAR	43
1.5.3	DIAGRAMA DE BLOQUES DE LAS FUNCIO- NES QUE REALIZA UN SISTEMA DE RAS- TREO AL AZAR	44
1.5.4	GENERADOR DE CARACTERES DEFINIDO POR	

Pág.

	UNA RETICULA DE PUNTOS RECTANGULAR	46
1.5.5	DIAGRAMA DE BLOQUES SIMPLIFICADO DE UN SISTEMA DE RASTREO CON RAS- TREADOR	48
1.6	DIAGRAMA DE BLOQUES DE LA TRANSFORMACION DE LA DEFINICION DE IMAGEN DE UN USUARIO PARA DAR CABIDA A VARIOS DISPOSITIVOS	50
CAPITULO II		
2.1.1	TRASLACION DE UN PUNTO	52
2.1.2	DESPLAZAMIENTO HORIZONTAL Y VERTICAL	53
2.1.3	TRASLACION DE UN OBJETO	54
2.2.1	ROTACION EN TORNO A UN PIVOTE	56
2.2.2	ROTACION EN TORNO AL ORIGEN	56
2.2.3	ROTACION EN TORNO A UN PUNTO (Px,Py)	56
2.3.1	ESCALAMIENTO	59
2.3.2	ESCALAMIENTO CAMBIA LA DISTANCIA A PARTIR DE UN PUNTO FIJO	61
2.3.3	ESCALMIENTO	61
2.4.1	AFILAMIENTO DE UNA LINEA VERTICAL Y UNA HORIZONTAL	63
2.4.2	AFILAMIENTO Y	63
2.4.3	AFILAMIENTO X	64
2.8	CAMBIO DE ORDEN DE UNA TRASLACION Y UNA ROTACION	71

CAPITULO III

Pág.

3.2.1	SISTEMA DE COORDENADAS TRIDIMENSIONALES	75
3.2.2	SISTEMA DE COORDENADAS DEL OBSERVADOR	77
3.3.1	VECTOR NORMAL A UN PLANO	81
3.3.2	PLANO DE UN CUBO UNITARIO	81
3.4.1	EJEMPLO DE TRES ORDENES DE CONTINUIDAD	87
3.4.2	SECCION ESFERICA	88
3.4.3	CURVA DESPLEGADA QUE PASA A TRAVES DE PUNTOS DE CONTROL	88
3.4.4	CURVA DESPLEGADA QUE PASA CERCA DE LOS PUNTOS DE CONTROL	88

CAPITULO IV

4.2.1	TRAYECTORIA DE LAS ROTACIONES TRIDIMENSIONALES POSITIVOS	91
4.2.2	EJE DE ROTACION PERPENDICULAR AL PAPEL	94
4.4.1	REPRESENTACION VECTORIAL DE LA RECTA	97
4.4.2	ROTACION EN TORNO A UN EJE ARBITRARIO	100
4.6.1	SEGUNDO SISTEMA DE COORDENADAS	104
4.6.1.a	ROTACION	104
4.6.1.b	OBJETO TRASLADADO Y ROTADO CON RESPECTO AL SISTEMA DE COORDENADAS INICIAL	104

CAPITULO V

5.1.1	PROYECCION DEL SISTEMA DE COORDENADAS
-------	---------------------------------------

5.1.2	PROYECCION OBLICUA DEL PUNTO SOBRE EL PLANO DE PROYECCION	109
5.2.1	SISTEMA DE COORDENADAS DE VISION	113
5.2.2	ORIENTACION DEL SISTEMA DE COORDE- NADAS DE VISION	114
5.2.3	SISTEMA DE VISION	116
5.2.4	SECUENCIA DE TRANSFORMACIONES PARA ALINEAR UN SISTEMA DE VISION	118
5.2.5	ESPECIFICACION DE VENTANA SOBRE EL PLANO DE VISION	119
5.2.6	PARALELEPIPEDO INFINITO	119
5.2.7	VOLUMENES CON VISTA FINITO LIMITADO POR SEIS PLANOS	121
5.2.8	VOLUMEN CON VISTA DE UN PARALELEPI- PEDO REGULAR	123
5.2.9	CORTE DE UN VOLUMEN CON VISTA	125

CAPITULO VI

6.3.1	SISTEMA DE VISUALIZACION DEL LADO DERECHO CON LA DIRECCION DE VISION EN Z NEGATIVO	129
6.3.2	SUPRESION DE LA CARA ANTERIOR EN UN SISTEMA DE LADO DERECHO	129
6.4.1	TRES SUPERFICIES EN VARIAS PROFUNDIDA- DES EN UN SISTEMA DE LADO IZQUIERDO	132
6.4.2	POSICION EN UNA LINEA DE RASTREO	135

Pág.

CAPITULO VII

7.1	OPERACIONES LOGICAS EN UNA VISION 3D	141
7.2.1	ESTRUCTURA DE DATOS	143
7.2.2	TABLA DE POLIGONOS	145
7.3.1	SISTEMA DE COORDENADAS DE VISION	150
7.3.2	ORIENTACION DEL SISTEMA DE COOR- DENADAS DE VISION	151

CAPITULO VIII

8.2.1	SECUENCIA FARA LA TRANSFORMACION DE UNA DEFINICION DE COORDENADAS MAESTRAS	156
-------	--	-----

INDICE DE **TABLAS**

Pág.

CAPITULO VIII

8.4.1	OPCIONES DE MOVIMIENTOS 1	158
8.4.2	OPCIONES DE MOVIMIENTOS 2	159



INTRODUCCION

La **graficación** por computador puede definirse como la **creación** de imágenes gráficas por medio de un computador. Tal definición, sin embargo, no alcanza a describir la diversidad de aplicaciones y el impacto ejercido por esta rama de las ciencias de la computación cuyo desarrollo ha sido muy acelerado. Ciertamente, la graficación **por** computador empezó como una técnica destinada a enriquecer la presentación de información generada por computador, hoy en día prácticamente no existe ninguna área en la cual no puedan utilizarse los despliegues gráficos con alguna ventaja. Aunque **las** primeras aplicaciones en ciencia e ingeniería tenían **que** basarse en equipo costoso y complicado, los **adelantos** en tecnología de computación han hecho de las gráficas de computadoras interactivas una herramienta práctica.

En realidad, una estación de graficación suele componerse de varios microprocesadores, cada uno con su propia memoria y diseñado para realizar una función específica, ya sea gráfica o **alfanumérica**.

Podemos decir **que** las gráficas se utilizan rutinariamente en áreas **como** la ingeniería, administración, la industria, el gobierno, arte,

entretenimiento, publicidad,, educación, investigación, capacitación y medicina. Por las grandes ventajas que ofrece la **graficación** por computador se realizó esta tesis la cual consiste en la implementacidsn de librerías gráficas y una aplicación de este paquete la cual permite que se maneje desde un computador personal a un **tema** Grúa-Robot cuyos movimientos son observados simultáneamente en tres dimensiones en la pantalla del computador.



CAPITULO 1

PANORAMA GENERAL DE LOS **SISTEMAS** DE GRAFICOS

1.1 GENERALIDADES

Los sistemas de computación pueden adaptarse a aplicaciones de las gráficas en varias formas, de acuerdo a los recursos de hardware y software de que se dispone. Cualquier computadora de uso general puede utilizarse para hacer gráficas de caracteres utilizando elementos del conjunto de caracteres del sistema con objeto de formar modelos o patrones. En aplicaciones más complejas. se dispone de una variedad de paquetes de software y dispositivos de hardware.

1.2 DISPOSITIVOS DE DESPLIEGUE

La **operación** de muchos monitores de video se basa en el diseño **estándar** del tubo de rayos **catódicos** (CRT), pero existen otras tecnologías.

TUBOS DE RAYOS CATODICOS DE RENOVACION

La figura 1.2.1 ilustra la operación básica de un CRT. Un haz de electrones (rayos catódicos),

emitidos por un disparador de electrones, atraviesa los sistemas de enfoque y deflexión que dirigen el haz hacia puntos especificados en la pantalla cubierta de fósforo. El fósforo emite después una pequeña mancha de luz en cada punto contactado por el haz de electrones. Una manera de conservar el fósforo brillando consiste en trazar la imagen varias veces dirigiendo rápidamente el haz de electrones hacia atrás sobre los mismos puntos. Este tipo de despliegue se denomina **CRT de renovación**.

Se dispone diferentes tipos de fósforos, diferenciados principalmente por su persistencia. La persistencia se define como el tiempo que tarde la luz emitida en perder una décima parte de su intensidad original. Los fósforos de persistencia inferior requieren intensidades de renovación más altas para mantener una imagen en la pantalla sin que fluctúe.

Un fósforo con baja persistencia es útil en la animación, mientras que las sustancias fosfóricas de alta persistencia se adaptan mejor para desplegar imágenes estáticas muy complejas.

La deflexión del haz de electrones se realiza con campos eléctricos o bien con campos magnéticos. El método electrostático se ilustra en la figura 1.2.1.

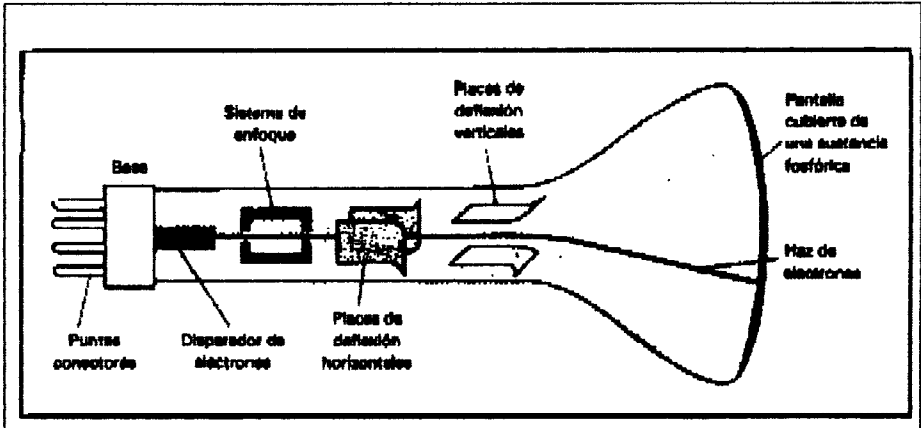


Figura 1.2.1 Diseño básico de un CRT, usando campos de deflexión electrostáticos.

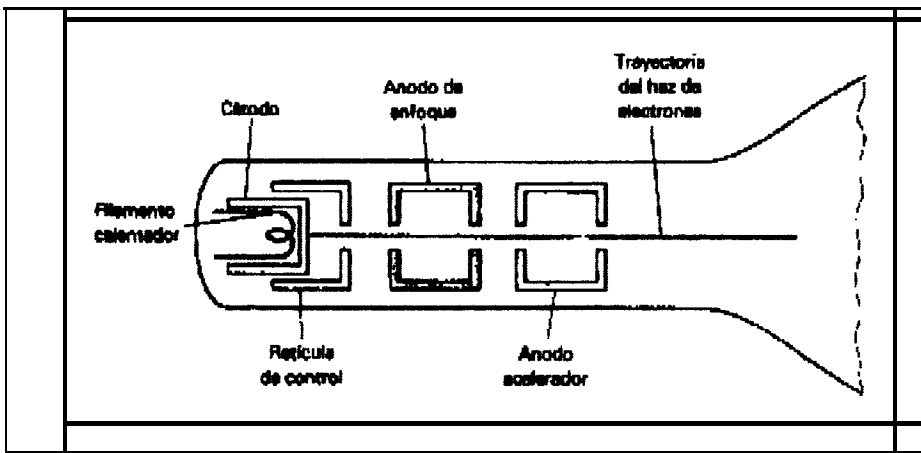


Figura 1.2.2 **Operación** de un disparador de electrones con un ánodo acelerador.

Aquí, el haz pasa entre **dos pares** de placas de metal: un par vertical **y** el otro horizontal. Se aplica una diferencia de tensión **a** cada par de acuerdo con la cantidad del haz que se deflexionará primero en cada dirección. A medida que el haz de electrones pasa entre cada par de placas, se dobla hacia la placa con la mayor tensión positiva. En la figura.1.2.1, el haz se deflexiona primero hacia un lado de la pantalla. Después, conforme el haz atraviesa las placas horizontales, éste se deflexiona hacia la parte superior o inferior de la pantalla. Las deflexiones adecuadas pueden lograrse ajustando la corriente que pasa por las bobinas colocadas alrededor de la parte exterior de la cubierta del CRT.

Los componentes básicos de un disparador de electrones en un CRT son el cátodo de metal calentado **y la** retícula de control (**fig.** 1.2.2). Se aplica calor al cátodo dirigiendo una corriente **a** través de una bobina de alambre, llamada filamento, hacia el interior de la estructura del cátodo cilíndrico. Esto ocasiona que los electrones "desaparezcan" de la superficie del cátodo caliente. En el interior al vacío de la cubierta CRT, los electrones libres con carga negativa se aceleran entonces hacia la cubierta fosfórica por medio de una tensión positiva alta. La tensión acelerada puede generarse mediante una

cubierta de metal con carga positiva en el interior de la cubierta del CRT cercana a la pantalla fosforescente o bien puede emplearse un ánodo acelerador, como en la figura 1.2.2.

Se controla la brillantez de un despliegue variando la tensión en la retícula de control. Se dispone de un botón de control en los monitores de video para fijar la brillantez de toda la pantalla.

El enfoque se logra con campos eléctricos o bien magnéticos. Para lograr un enfoque electrostático, el haz de electrones atraviesa un cilindro metálico con una tensión positiva, como se muestra en la figura 1.2.2. La tensión positiva obliga a los electrones a permanecer junto con el eje del haz.

Definiremos la resolución como el número de puntos por centímetro **que** pueden graficarse horizontal y verticalmente, aunque con frecuencia simplemente se anuncia como el número total de puntos en cada dirección. La resolución de una CRT depende del tipo de sustancia fosforescente que se utiliza y de los sistemas de enfoque y deflexión.

Una propiedad importante de los monitores de video es su razón de aspecto. Este número da la razón o proporción de puntos verticales a puntos horizontales se necesita para producir líneas de igual longitud en

ambas direcciones de la pantalla. Una razón de aspecto de $3/4$ significa **que** una línea vertical trazada con tres puntos tiene la misma longitud que una horizontal trazada con cuatro puntos.

MONITORES DE RASTREO AL AZAR Y CON RASTREADOR

Los CRT de renovación pueden ser **operados** como monitores de rastreo al azar o bien como monitores de rastreo con rastreador. Cuando se opera como una unidad de despliegue con rastreo al azar, un CRT tiene el haz de electrones dirigido solamente **a** las partes de la pantalla donde se trazará una figura. Los monitores de rastreo al azar dibujan una figura línea por línea y, por esta razón, **también** se conocen como despliegues verticales (o bien despliegues de escritura con golpe o bien caligráficas). Las líneas componentes de una figura pueden ser trazadas y renovadas por un sistema de rastreo al azar en cualquier orden que se especifique (fig. 1.2.3), una graficadora con pluma es un ejemplo de dispositivo copia dura.

Los monitores de video de rastreo con rastreador emiten el haz de electrones a todas las partes de la pantalla, subiendo **y** bajando la intensidad del haz para que coincida con la definición de la imagen. La figura se crea en la pantalla como un conjunto de

Figura
1.2.3

Sistema
de ras-
treo al
azar
traza
líneas
en cual-
quier
orden.

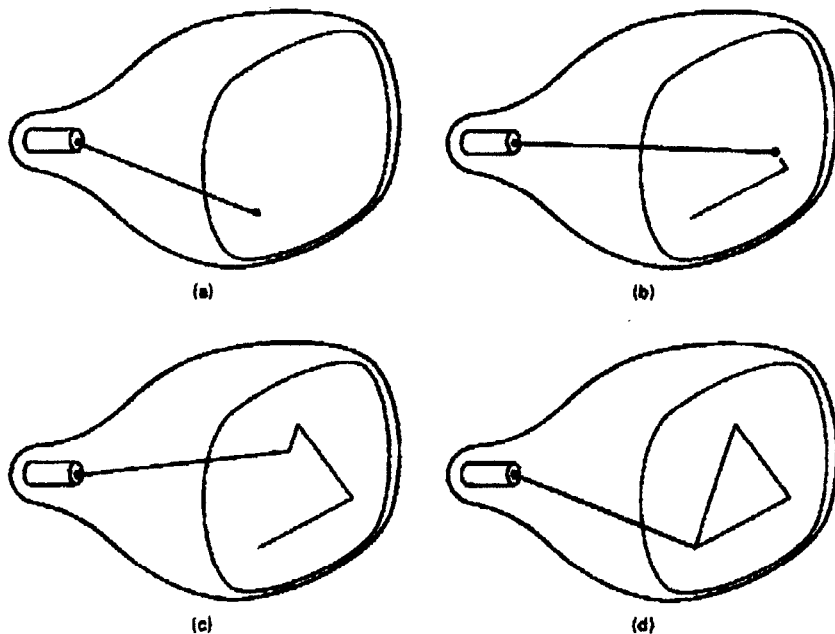
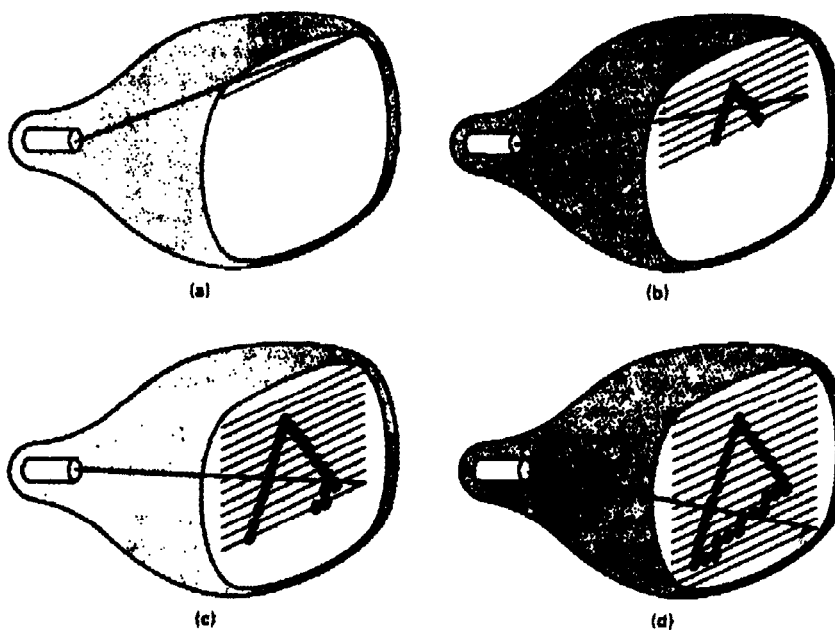


Figura
1.2.4

Sistema
de ras-
treo con
rastrea-
dor
desplie-
ga un
conjunto
de pun-
tos en
cada lí-
nea de
rastreo.



puntos (fig. 1.2.4), comenzando desde la parte superior de la pantalla. La definición de una imagen se almacena ahora como un conjunto de valores de intensidad de todos los puntos de la pantalla y estos valores almacenados se pintan en la pantalla, una hilera (línea de rastreo) a la vez. La capacidad de un sistema de rastreo con rastreador para almacenar información de intensidad de cada punto de la pantalla lo hace adecuado para desplegar áreas con sombra y color, mientras que los sistemas de vectores o vectoriales están restringidos a las aplicaciones de trazos de líneas. Los aparatos de televisión y las impresoras de tipo casero son ejemplos de otros sistemas que hacen uso de métodos de rastreo con rastreador.

A menudo, el ciclo de renovación en un monitor de rastreo con rastreador (y en aparatos de TV) se lleva a cabo llevando el haz a través de cada tercer línea en una vuelta de arriba hacia abajo, después regresando (repasso vertical) para recorrer las líneas restantes de la pantalla en la siguiente vuelta descendente por la pantalla (fig. 1.2.5), lo que ayuda a reducir el destello a menores intensidades de renovación. **Escencialmente** se observa todo el despliegue en la pantalla en la mitad del tiempo que tardaría en recorrer todas las líneas a la vez de

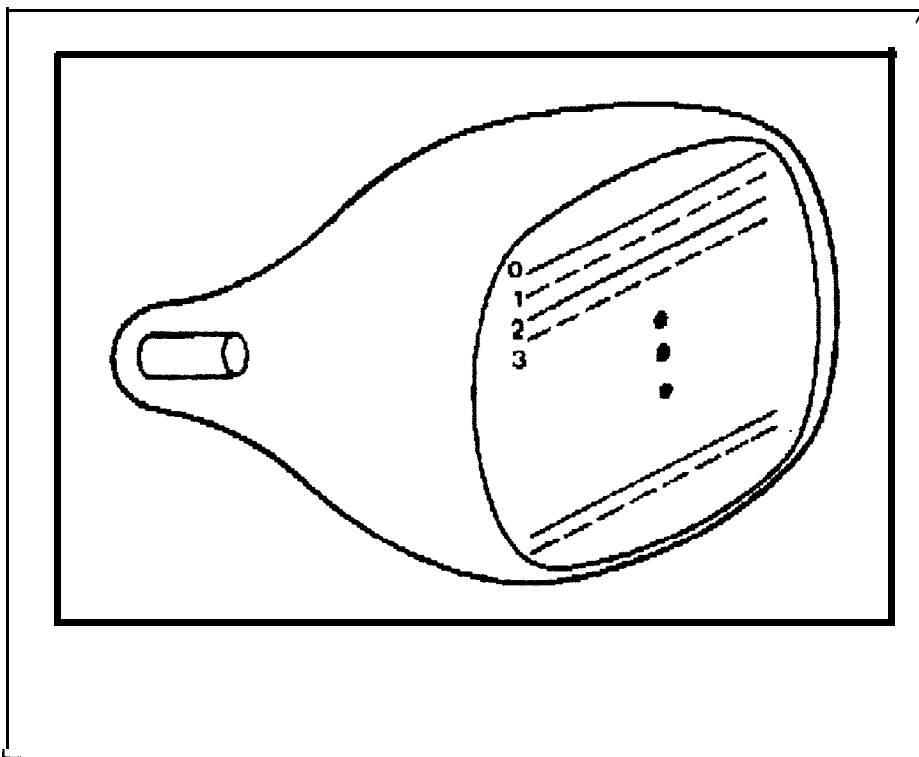


Figura 1.2.5 Un monitor de rastreo con rastreador puede programarse para entrelazar líneas de rastreo:

Primero, todos los puntos de las líneas(llenas) de número par son desplegadas; después se despliegan todos los puntos situados a lo largo de las líneas de número impar(punteadas).

arriba hacia abajo.

MONITORES CRT DE COLOR

Un monitor CRT despliega figuras de color utilizando una combinación de sustancias fosfóricas que emiten luz de diferentes colores. Combinando la luz emitida de las diferentes sustancias **fosforescentes** puede generarse una gama de colores. Las dos técnicas básicas para producción de despliegues a color con un CRT son el método de penetración del haz y el método de la máscara de sombra.

El método de penetración del haz para desplegar,.....: figuras a color se ha utilizado con monitores de rastreo al azar. Dos capas de sustancia fosforescente, por lo general rojo y verde, se colocan en la pantalla y el color exhibido depende de cuánto penetre el haz de electrones en las capas fosforescentes. Un haz de electrones lentos excita solamente la capa roja exterior. Un haz de electrones muy rápidos penetra a través de la capa roja y excita la capa verde interior. A velocidades intermedias del haz, se emiten combinaciones de luz roja y verde para mostrar dos colores adicionales, naranja y amarillo. La velocidad de los electrones y, por tanto, el color de la pantalla en cualquier punto se controla por la tensión de aceleración del

haz. La **penetración** del haz ha sido una forma poco costosa de producir color en monitores de rastreo al azar, pero sólo puede haber cuatro colores y la calidad de las imágenes no es tan buena como con otros métodos.

Los métodos de máscara con sombra se usan comúnmente en sistemas de rastreo con rastreador (inclusive TV cromática), ya que producen una gama de colores mucho más grande que el método de penetración del haz. Un CRT de máscara con sombra cubre la pantalla con modelos triangulares pequeños, cada uno de los cuales contiene tres puntos fosforescentes diferentes con muy poco espacio entre sí. Un punto fosforescente de cada triángulo emite una luz azul. Este tipo de CRT tiene tres disparadores de electrones, uno para cada punto de color y una retícula de máscara con sombra apenas detrás de la pantalla cubierta con sustancia luminiscente (fig. 1.2.6).

Los tres haces de electrones son desviados y enfocados como grupo sobre las máscaras con sombra, la cual contiene una serie de orificios alineados con los modelos fosforescentes. Cuando los tres haces atraviesan un orificio en la máscara con sombra, éstos activan un triángulo formado con puntos, el cual aparece como una pequeña mancha en la pantalla.

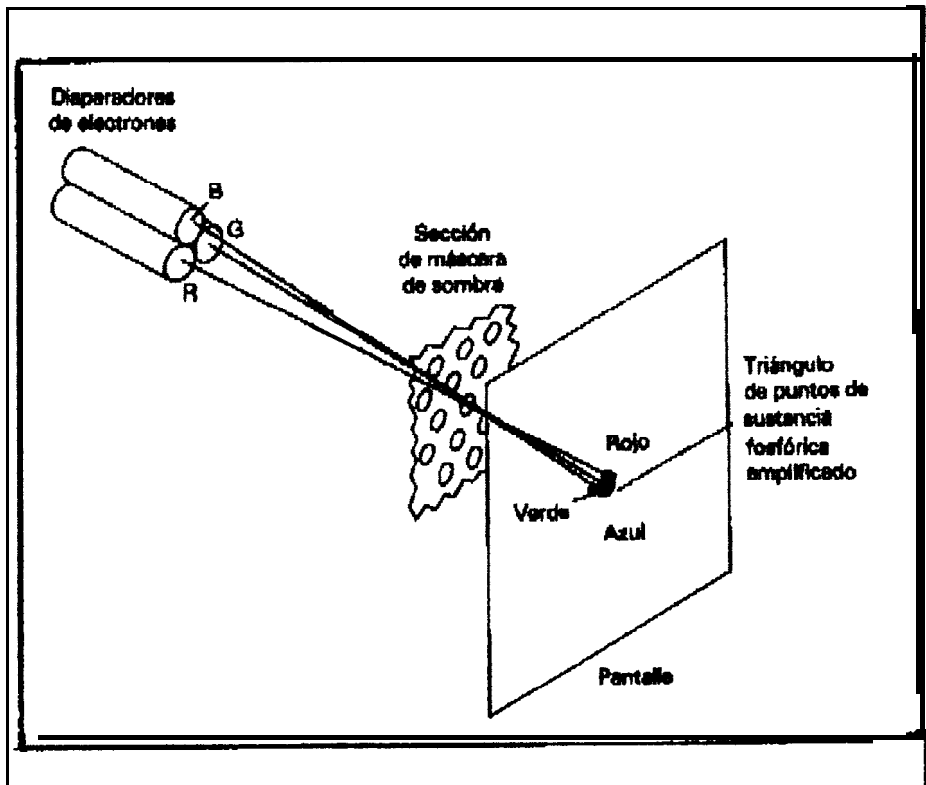


Figura 1.2.6 Operación de un CRT con máscara de sombra. Tres disparadores de electrones, dispuestos para coincidir con los modelos de puntos de color de la pantalla, se dirigen a cada triángulo punteado por medio de una máscara de sombra.

Los puntos fosforescentes de los triángulos se disponen de manera que cada haz de electrones pueda activar solamente su punto de color correspondiente cuando atraviese la máscara de sombra.

Las variaciones de color en un CRT con máscara de sombra se obtienen combinando varios niveles de intensidad de los tres haces de electrones. Al apagar los disparadores rojo y verde se obtiene sólo el color que proviene de la sustancia fosforescente azul. Otras combinaciones de intensidades de los haces producen una pequeña mancha de luz por cada triángulo, cuyo color depende de la cantidad de excitación de las sustancias fosfóricas roja, verde y azul del triángulo.

En sistemas de bajo costo, el haz de electrones sólo puede encenderse o apagarse, limitando los despliegues a ocho colores. Los sistemas más complejos pueden fijar niveles de intensidad intermedios de los haces de electrones, permitiendo con ello que se generen varios millones de colores.

Los sistemas de computación personal con recursos de gráficas de colores a menudo se diseñan para utilizarse con varios tipos de dispositivos de despliegue de CRT. Estos dispositivos incluyen aparatos de TV, monitores compuestos y monitores RGB

(rojo-verde-azul).

Los CRT de color de alta calidad son diseñados de la misma manera que los monitores RGB. Estos monitores toman el nivel de intensidad de cada disparador de electrones (rojo, verde y azul) directamente desde el sistema de computación sin existir ningún procesamiento intermedio. En esta **fórmula**, se generan menos distorsiones de señales.

TUBOS DE **ALMACENAMIENTO** CON VISTA **DIRECTA**

Otro método para conservar una imagen en la pantalla consiste en almacenar la información de la figura dentro del CRT en vez de renovar la pantalla. Un tubo de almacenamiento con vista directa (DVST) almacena la información de la figura como una distribución de carga detrás de la pantalla cubierta con sustancia fosforescente. En un DVST se utilizan dos disparadores de electrones. Uno, el disparador primario, se usa para almacenar el modelo de la imagen; el segundo, el disparador del flujo, conserva el despliegue de la imagen.

En la figura 1.2.7 se muestra una sección transversal simplificada de un DVST. Un monitor DVST tiene ventajas y desventajas en comparación con el CRT de renovación. Como no se necesita renovación, pueden desplegarse figuras muy complejas sin fluctuar. Las

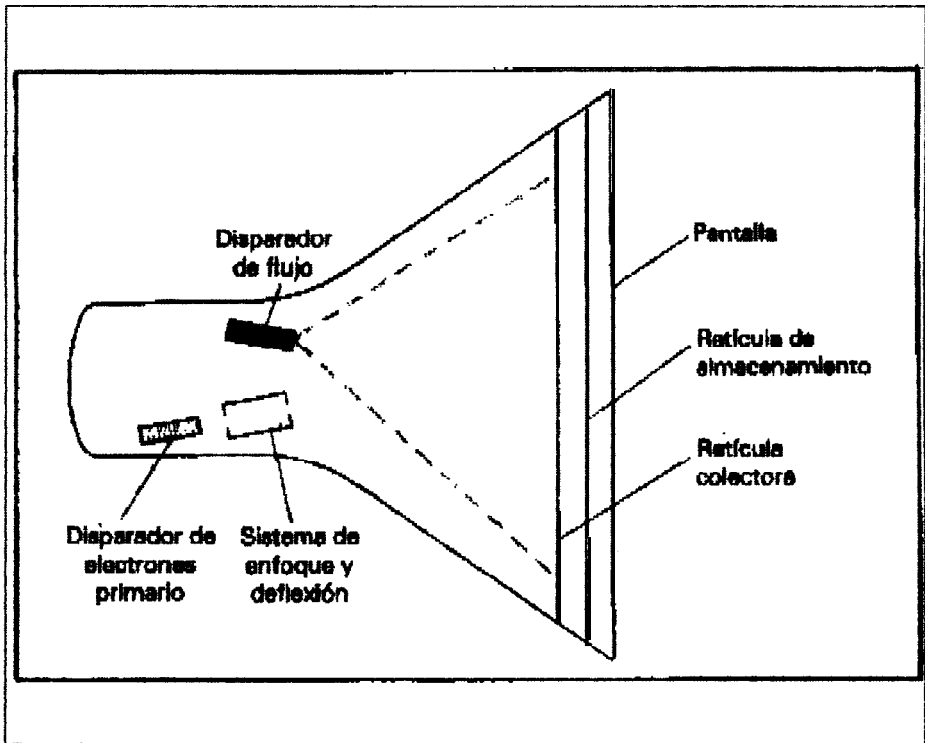


Figura 1.2.7 Operación de un DVST

desventajas de los sistemas DVST es que ordinariamente no exhiben color y que las partes seleccionadas de una figura no pueden borrarse. Para eliminar una **sección** de una imagen, la pantalla en su totalidad debe ser borrada almacenando una carga positiva en todas las partes de la retícula de almacenamiento. Después, los electrones del disparador de flujo chocan contra la cubierta fosforescente en todos los puntos de la pantalla, borrando la figura en un destello de luz. Toda la imagen se vuelve a trazar después, menos las partes que se vayan a omitir.

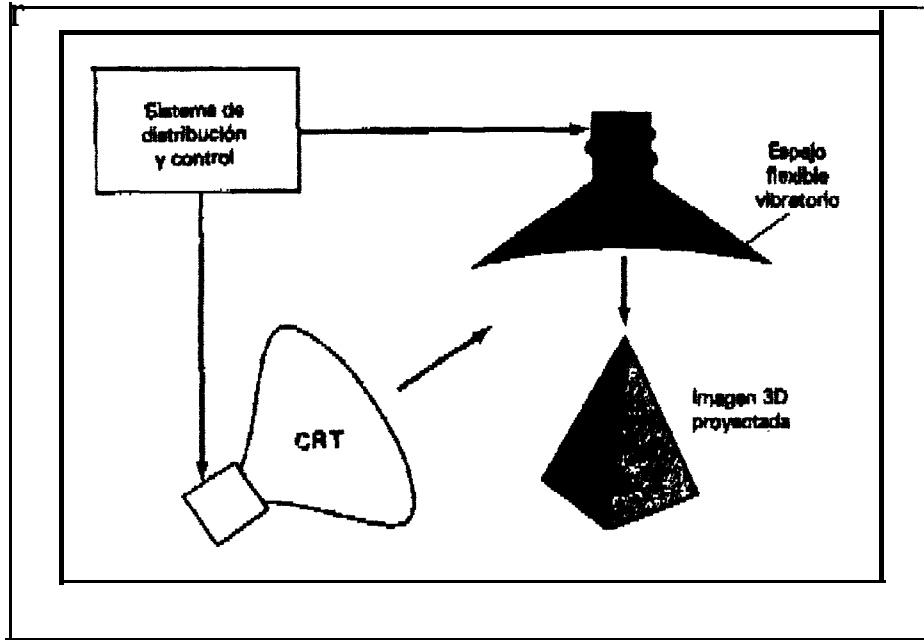
DISPOSITIVOS **LASER**

Una técnica adicional que no es de CRT para generar una salida de gráficas consiste en trazar modelos sobre película fotocromática, que se oscurece temporalmente por la exposición a la luz. Los modelos se forman con un haz de rayo láser, deflexionado por espejos controlados electromecánicamente. Después se utiliza otra fuente de luz para proyectar las imágenes en una pantalla. Un cambio de las imágenes de la pantalla se obtiene enrollando el carrete de película en el siguiente cuadro vacío y repitiendo el proceso. Pueden desplegarse modelos muy complejos en un tiempo muy corto con estos sistemas, pero no es posible un

borrado selectivo. Sólo pueden hacerse cambios a una figura volviendo a trazar por completo los modelos en el siguiente cuadro de la película.

Los monitores de gráficas para el despliegue de escenas tridimensionales se han ideado aplicando una técnica **que** refleja una imagen del CRT desde un espejo flexible vibratorio. La operación de tal sistema se demuestra en la figura 1.2.8. Conforme el espejo (llamado espejo varifocal) vibra, cambia la longitud focal. Estas vibraciones se sincronizan con el despliegue de un objeto en un CRT de manera que cada punto del objeto se refleje desde el espejo en una posición correspondiente a la profundidad de ese punto. Un observador puede ver debajo, alrededor o bien sobre la parte superior del objeto. Este sistema también puede exhibir "cortes" bidimensionales de secciones transversales de objetos seleccionados en diferentes profundidades. Dichos sistemas se utilizan en aplicaciones médicas para analizar datos de ultrasonografía y dispositivos de examen CAT, en aplicaciones geológicas para analizar datos topológicos y **sísmicos**, en aplicaciones de diseño en donde intervienen objetos tridimensionales y en simulaciones tridimensionales de sistemas, como moléculas y terreno.

1.3 DISPOSITIVOS DE COPIA DURA



BIBLIOTECA

Figura 1.2.8 Operación básica de un sistema de despliegue tridimensional que usa un espejo vibratorio que cambia la longitud focal para ajustar la profundidad de los puntos de una escena.



100 E 12

Muchos sistemas de gráficas están equipados para producir salida dura directamente de un monitor de video en al forma de acetatos o transparencias de 35 mm. Las figuras en copia dura también pueden obtenerse dirigiendo la salida de gráficas a una impresora o graficadora.

IHPRESORAS

Las impresoras producen salida por métodos de impacto o bien de no impacto. Las **impresoras de impacto** oprimen caras de caracteres formados contra una cinta entintada contra el papel. La conocida impresora de líneas es un ejemplo de un dispositivo de impacto, con los tipos montados sobre bandas, cadenas, tambores o margaritas (o discos). La impresora de margarita (o de disco) utiliza el método de impacto para imprimir caracteres de uno en uno. Una cabeza de impresión de matriz de puntos, que contiene un arreglo rectangular de puntas muy finas, se usa a menudo en las impresoras de caracteres de impacto para formar caracteres individuales activando modelos seleccionados de puntas de contacto. Las **impresoras de no impacto** son más rápidas y silenciosas y con frecuencia se valen de un método de matriz de puntos para imprimir caracteres o trazar líneas. Los atomizadores de chorro de tinta, técnica láser, procesos xerográficos (como los que se usan en las

máquinas fotocopadoras), métodos electrostáticos **y métodos** electrotérmicos se emplean en el diseño de impresoras de no impacto.

Los métodos de matriz de puntos ofrecen mayores posibilidades para obtener salidas de gráficas.

Los métodos del chorro de tinta producen la salida arrojando tinta en las hileras o líneas horizontales a **través** de un rollo de papel enrollado en un tambor. El flujo de la tinta, **que** está eléctricamente cargado, es desviado por un campo eléctricamente para producir modelos de matriz de puntos. Los métodos electrostáticos colocan una carga negativa en una hoja de papel plana, una hilera completa a la vez hasta que se termina el papel. Después el papel se expone **a** un "tóner" negro. El tóner está positivamente cargado y **por** eso es atraído a las áreas de carga negativa, donde se adhiere para generar la salida especificada. Los métodos electrotérmicos utilizan calor en la cabeza de impresión de la matriz de puntos para producir modelos en papel sensible al calor. Una impresora láser opera en forma análoga **a** una copidora **xerox**. El haz de láser crea una distribución de cargas en un tambor cubierto con un material fotoeléctrico, como el selenio. Se aplica tóner al tambor y después se transfiere al papel.

Se dispone de impresoras en blanco y negro y cromáticas. Se han utilizado cintas de diferentes colores en las impresoras de impacto para obtener variaciones de color, pero las impresoras de no impacto se adaptan mejor a la salida con color. Los dispositivos de no impacto utilizan varias técnicas para combinar tres pigmentos de color (azul-verde, magenta y amarillo) y producir una gama de modelos de color. Las impresoras láser y xerográficas depositan los tres pigmentos en pases separados; las impresoras de chorro de tinta emiten los tres colores en forma simultánea en un solo pase en cada línea de impresión del papel.

GRAFICADORAS

Estos dispositivos producen trazos de líneas en copia dura. Las graficadoras más comunes son las plumas de tinta para generar los trazos, Pero muchos dispositivos de **graficación** emplean ahora rayos láser, atomizadores de chorro de tinta y métodos electrostáticos.

1.4 DISPOSITIVOS DE ENTRADA INTERACTIVOS

Las estaciones de trabajo de gráficas, incluyen varios tipos de dispositivos de entrada y salida: Dos monitores de alta resolución y figuras de color

con rastreador. varios dispositivos de entrada para lograr interacción con los despliegues.

Los teclados se incluyen en muchos sistemas de gráficas para realizar la entrada de cadenas de caracteres y valores de datos.

Los valores coordenados que especifican posiciones en la pantalla se introducen con mayor **frecuencia** con una tableta de gráficas, pluma luminosa o pluma linterna, ratón o una palanca de mando. **Estos** dispositivos se usan comúnmente para trazar figuras o hacer selecciones de menú. Una pluma luminosa indicada en un monitor de video registra posiciones coordenadas, respondiendo a la luz emitida de sustancias fosforescentes en la pantalla. Y se utiliza un **ratón** o palanca de mando para posicionar el cursor en la pantalla en las localidades que se seleccionarán.

Otros tipos de dispositivos figuran los paneles de tacto, sistemas de voz y dispositivos para introducir información coordinada tridimensional.

1.5 PROCESADOR DE DESPLIEGUES

Los sistemas de gráficas interactivos emplean dos o mas unidades de procesamiento. Además de la unidad central de procesamiento o UCP, se utiliza un procesador de despliegues de uso general para

interactuar con la UCP y controlar la operación del dispositivo de despliegue (fig. 1.5.1). Básicamente, el procesador de despliegue se utiliza para convertir información digital de la UCP en valores de tensión correspondiente **que** necesita el dispositivo de despliegue. La forma en la cual se realiza esta conversión de digital a analógico depende del tipo de dispositivo de despliegue que se usa y de **las** funciones de gráficas particulares que se instrumentarán en hardware. En algunos sistemas, se usa más de un procesador para instrumentar las funciones de despliegue de gráficas.

Una tarea fundamental para el procesador de despliegue es la exhibición de segmentos de líneas. Los niveles de intensidad (o valores de color) que se usarán en posiciones coordinadas de **graficación** a lo largo de una línea son proporcionados por el programa de aplicaciones y se convierten en niveles de tensión, que después se aplican al dispositivo de despliegue. Para sistemas simples de blanco y negro, no necesita especificarse ninguna información de intensidad en el programa, **ya que** cualquier punto está encendido o bien apagado. Los sistemas de mayor calidad permiten que la intensidad de los puntos de la pantalla sea variada de manera **que** puedan desplegarse sombras de gris.

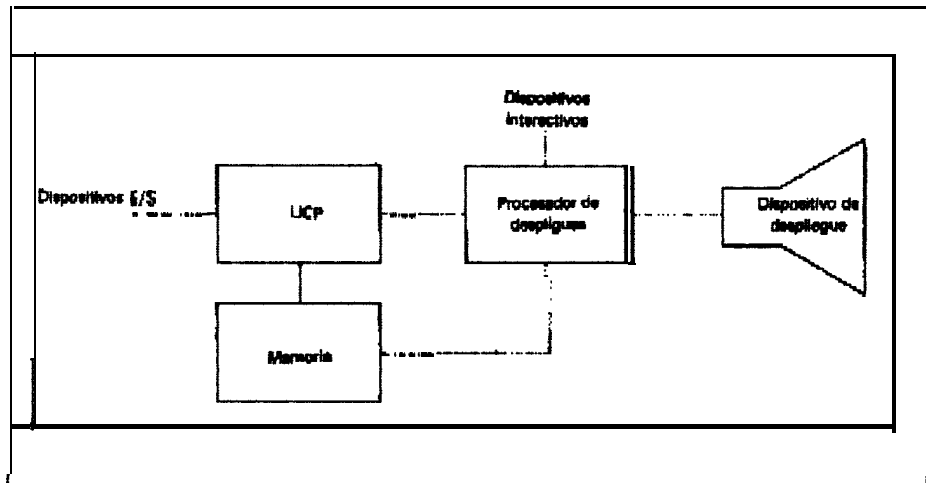


Figura 1.5.1 Diagrama de hardware simplificado de un sistema interactivo de gráficas.

Asimismo, los procesadores de despliegue están diseñados comúnmente para interactuar con dispositivos de entrada interactivos, **como** una pluma luminosa, en el caso de **sistemas** CRT de renovación, puede ser solicitado renovando la pantalla con la suficiente frecuencia para eliminar la fluctuación. En muchos sistemas, esta tarea de renovación de la pantalla puede asignarse a un procesador adicional.

SISTEMAS DE RASTREO AL AZAR

La figura 1.5.2 es un diagrama simplificado de las operaciones lógicas que realiza un sistema de rastreo al azar. Los comandos de gráficas de un programa de aplicaciones se traducen en un programa de archivo de despliegues, que es accesado por el procesador de despliegue para renovar la pantalla. El **procesador** de despliegue entra en un ciclo a través de cada comando del programa de archivo de despliegues una vez durante cada ciclo de renovación.

Cuando un controlador de despliegues se incluye en un sistema de rastreo al azar, se pueden usar dos archivos. Los comandos de gráficas traducidos se almacenan primero en un archivo de despliegues, como se muestra en la figura 1.5.3. **Después** el procesador de despliegues copia comandos en un archivo de desplegado de renovación para ser accesado por el

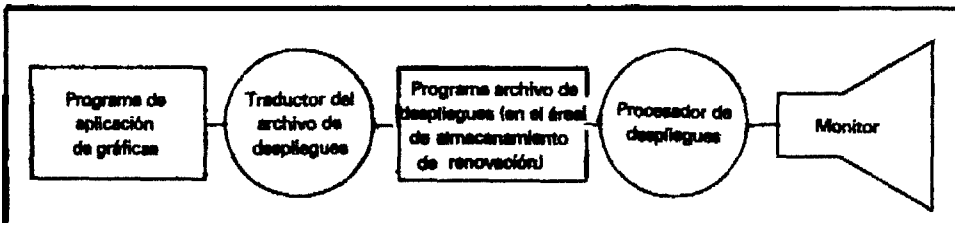


FIGURA 1.5.2
Diagrama de bloque simplificado de las funciones que realiza el sistema de rastreo al azar.

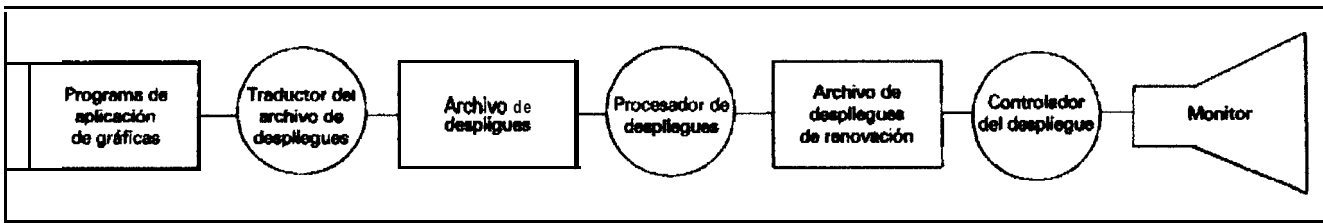


FIGURA 1.5.3
Diagrama de bloque de las
funciones que realiza un sistema
de rastreo al azar con un
controlador de despliegue.

controlador de despliegues. Este archivo de despliegue de renovación se crea aplicando las operaciones de visualización que seleccionan la vista particular que se desplegará en la pantalla. Durante el proceso de renovación, que ahora es afectado por el controlador de despliegues, el procesador de despliegue puede estar actualizando el archivo de renovación mientras se introducen comandos interactivos. Estas actualizaciones deben sincronizarse por el proceso de renovación, de manera que no se distorsione la imagen mientras está en proceso de renovación.

Los modelos de gráficas se trazan en un sistema de rastreo al azar dirigiendo el haz de electrones a lo largo de las líneas componentes de la figura. Las líneas se definen por los valores de sus puntos extremos coordinados y estos valores coordinados de entrada se convierten en tensiones de deflexión x y y . Después se traza una escena, una línea a la vez, posicionando el haz para llenar la línea entre los puntos extremos especificados.

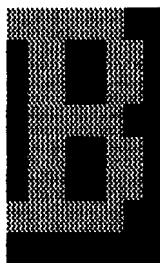
Las líneas rectas se trazan en un sistema de rastreo al azar con un generador de vectores, componente de hardware del procesador de despliegues (o controlador de despliegues), que produce tensiones de deflexión del haz de electrones. Los estilos de líneas, como

una línea punteada (con líneas), se manejan apagando y encendiendo alternativamente el haz de electrones conforme se traza la línea.

Dada la representación funcional de la curva, las instrumentaciones del hardware pueden idearse para desplegar la curva como una serie de segmentos de líneas recta cortos o bien como un conjunto de puntos.

Figura 1.5.4

Las retículas **rectan-
gulares** pueden definir
caracteres como un con-
junto de puntos



Un **caracter** se despliega superponiendo la retícula rectangular en la pantalla en una **posición** coordenada especificada Figura 1.5.4.

SISTEMAS DE RASTREO CON RASTREADOR

La **operación** de un sistema de rastreo con rastreador difiere de la de un sistema de rastreo al azar en que el área de almacenamiento de renovación se utiliza para almacenar información por cada posición de la pantalla, en vez de almacenar información de intensidad de la pantalla, en vez de comandos de

gráficas. En los sistemas de rastreo con rastreador, el almacenamiento de renovación se conoce generalmente como buffer de estructura o bien buffer de renovación (fig. 1.5.5). Cada posición del buffer de estructura se llama elemento de la figura o bien **pixel**. Las figuras se pintan en la pantalla desde el buffer de estructura, una hilera a la vez, de arriba hacia abajo. Cada línea horizontal de pixeles se conoce con **línea de rastreo**, y el proceso de generación de información de pixeles en el buffer de estructura del programa de aplicación se conoce como **conversión de rastreo**. Los valores de intensidad se meten en el rastreador durante el tiempo de **retrazo** vertical.

Las posiciones de los pixeles en el buffer de estructura se organizan como un arreglo bidimensional de valores de intensidad, correspondientes a posiciones coordenadas de la pantalla. El número de posiciones de pixeles en el rastreador se denomina **resolución** del procesador de despliegue (o bien resolución del buffer de estructura). Los sistemas rastreadores de buena calidad tienen una resolución de cerca de 1024 por 1024, aunque se dispone de sistemas de resolución más alta. Para generar imágenes de la mejor calidad, la resolución del monitor de video debe ser igual o mayor que la

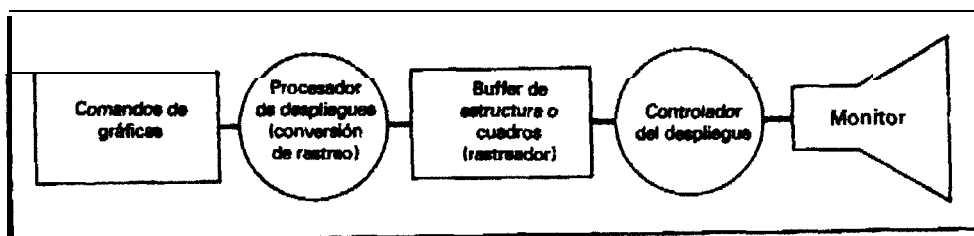


Figura 1.5.5 Diagrama de bloques simplificado de un sistema de rastreo con rastreador.

1.6 SOFTWARE DE GRAFICAS

REPRESENTACIONES DE COORDENADAS

Las coordenadas referidas por un usuario se denominan **coordenada mundiales** y las coordenadas que utiliza un dispositivo de salida particular reciben el nombre de **coordenadas de dispositivo** o bien **coordenadas de pantalla** en caso que se trate de un monitor de video. Las definiciones de las coordenadas mundiales permiten a un usuario fijar cualquier dimensión adecuada sin verse obstaculizado por las restricciones de un dispositivo de salida determinado.

El procedimiento consiste en convertir las definiciones de las coordenadas mundiales en **coordenadas de dispositivo normalizadas** antes de la conversión final a coordenadas de dispositivos específicas. Esto hace que el sistema sea lo suficientemente flexible para dar cabida a varios dispositivos de salida (fig. 1.6). Las coordenadas x y y normalizadas reciben los valores en el ámbito de 0 a 1. Estas coordenadas normalizadas se transforman después en coordenadas de dispositivo (enteras) dentro del intervalo $(0,0)$ a $(x_{máx}, y_{máx})$ para un dispositivo particular. Para tener en cuenta diferencias de escalas y razones de aspecto, las

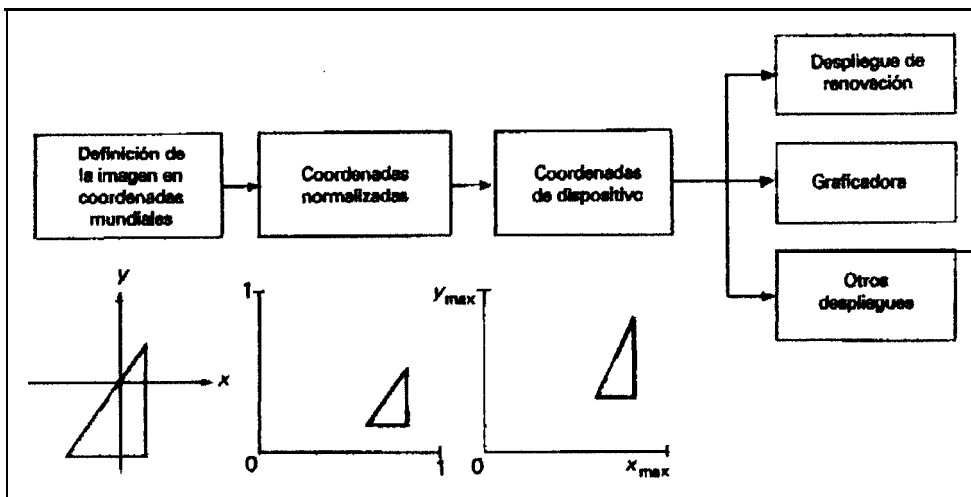


FIGURA 1.6

La transformación de la definición de imagen de un usuario (coordenadas mundiales) en coordenadas normalizadas a menudo es efectuada por un sistema de gráficas a fin de ofrecer una interfaz con diferentes tipos de dispositivos de salida.

coordenadas normalizadas pueden trazarse en un área cuadrada del dispositivo de salida de manera que se conserven las proporciones adecuadas. En un monitor de video, el área restante de la pantalla a menudo sirve para desplegar mensajes o listar opciones de programas interactivos.

CAPITULO II

TRANSFORMACIONES BIDIHENSIONALES

2.1 TRASLACIONES

Un punto en el sistema de coordenadas del mundo es transformado a otra posición modificando sus coordenadas x y y . En la figura 2.1.1 un punto en la posición (x, y) ha sido trasladado 10 unidades arriba y cinco a la izquierda.

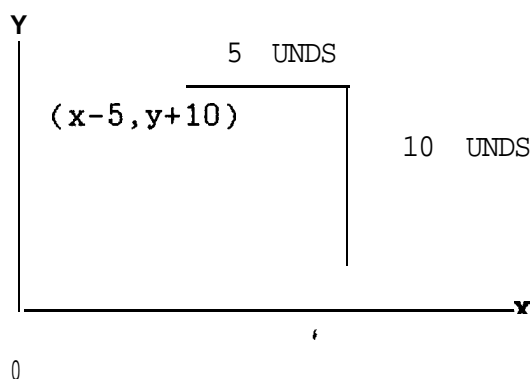
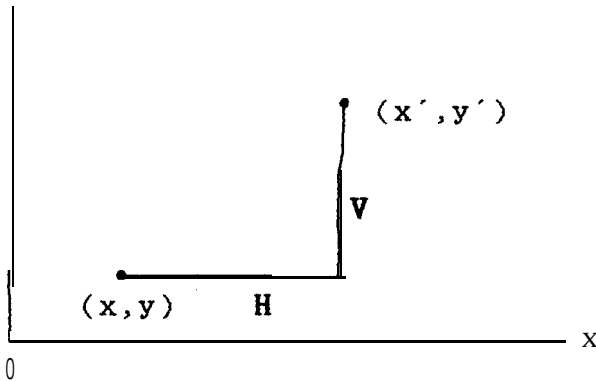


figura 2.1.1

Punto (x, y) se traslada 10 unidades a arriba y 5 izquierda

En general, un punto (x, y) es trasladado a una nueva posición (x', y') en H unidades en la dirección horizontal y V unidades en la dirección vertical (Fig. 2.1.2).

Fig. 2.1.2 Desplazamientos
horizontal y vertical



La **representación** matemática de esto es :

$$x' = x + H$$

$$y' = y + V$$

H y V representan el desplazamiento o distancia horizontal y vertical en que se movió el punto. Si H es positiva, el punto se mueve **a** la derecha; si H es negativa, el punto se mueve **a** la izquierda.

Similarmente, una V positiva desplaza el punto hacia arriba; una V negativa, hacia abajo.

En la mayoría de los casos, lo que se desea es trasladar no **sólo** un punto, sino uno o más objetos en una imagen. Para realizar esto, hemos de trasladar cada punto que define **el(los)** objeto(s). A continuación, todos los puntos se desplazan **a** la misma distancia, y el objeto es redibujado mediante estos puntos transformados (fig. 2.1.3).



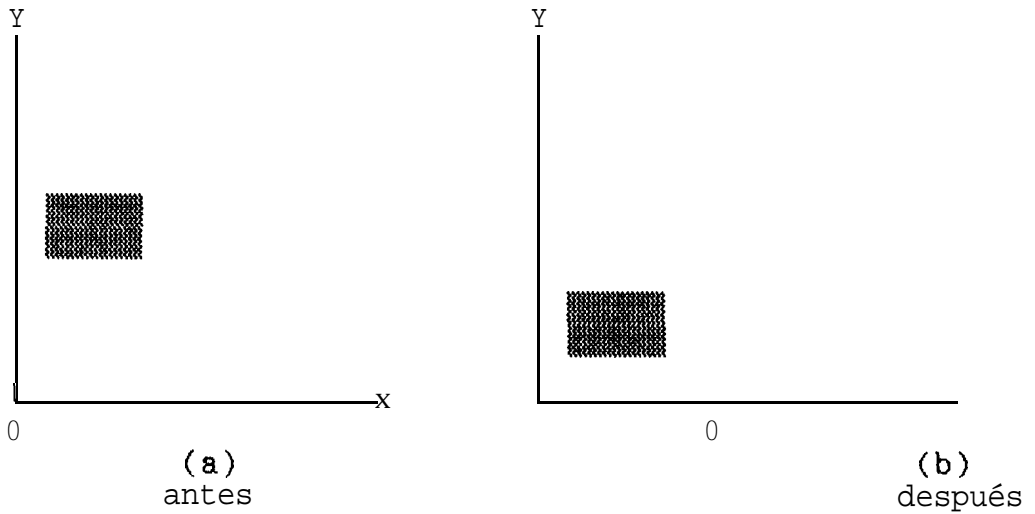


figura 2.1.3

Traslación de un
objeto.

2.2 ROTACIONES

Otra transformación útil es la rotación de un objeto en torno a un punto pivote especificado (fig. 2.2.1).

Después de que ha sido rotado, permanece a la misma distancia del pivote, aunque su orientación ha cambiado. Es posible rotar uno o más objetos, o la imagen completa, alrededor de cualquier punto en las coordenadas del mundo, ya sea en la dirección del movimiento de las manecillas del reloj o en la dirección contraria.

Se desea ahora **exminar** la matemática de una rotación, en dirección contraria al movimiento de las

manecillas del reloj, alrededor de un punto pivote arbitrario. A fin de hacer esta derivación un poco más accesible, primero se analizará una rotación en la que el pivote es el origen del espacio del mundo. Cualquier punto (x, y) puede ser representado por medio de su distancia radial, r , respecto del origen y su ángulo, Φ , al eje x . En la figura 2.2.2, el punto (x, y) se representa como:

$$x = r * \cos(\Phi)$$

$$y = r * \text{sen}(\Phi)$$

Si (x, y) es rotado un ángulo θ en dirección contraria al movimiento de las manecillas del reloj, el punto transformado (x', y') , como se ilustra en la figura 2.2.3, se representa como:

$$x' = r * \cos(\Phi + \theta)$$

$$y' = r * \text{sen}(\Phi + \theta)$$

Recurriendo a las leyes de la trigonometría para senos y cosenos, las ecuaciones anteriores se convierten en:

$$x' = r * \cos(\Phi) * \cos(\theta) - r * \text{sen}(\Phi) * \text{sen}(\theta)$$

$$y' = r * \text{sen}(\Phi) * \cos(\theta) + r * \cos(\Phi) * \text{sen}(\theta)$$

A partir de la definición de x y y , estas ecuaciones se reducen a:

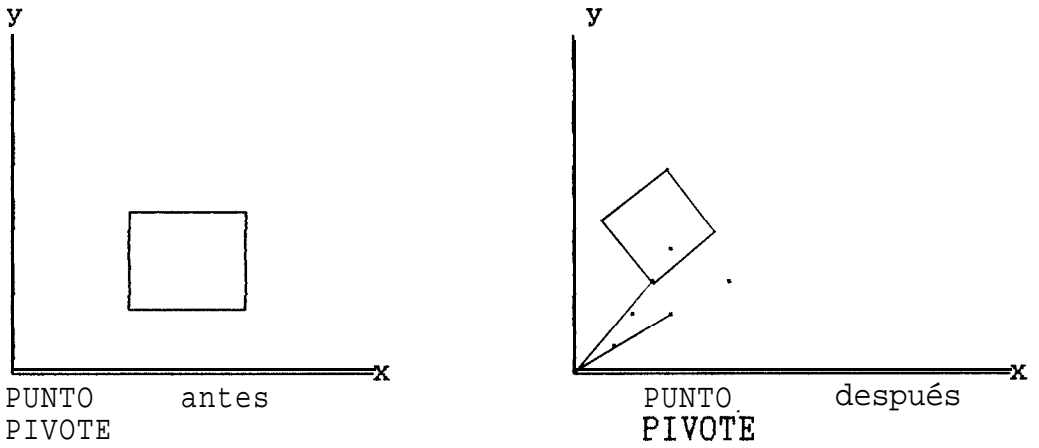


Figura 2.2.1 Rotación en torno a un pivote

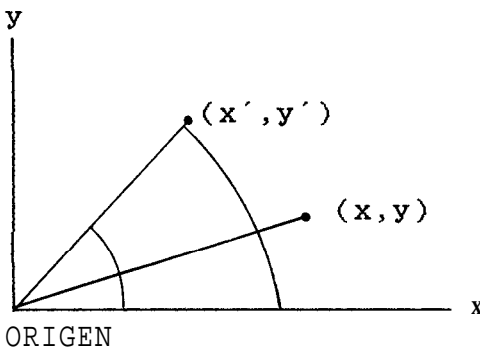


Figura 2.2.2 Rotación en torno al origen

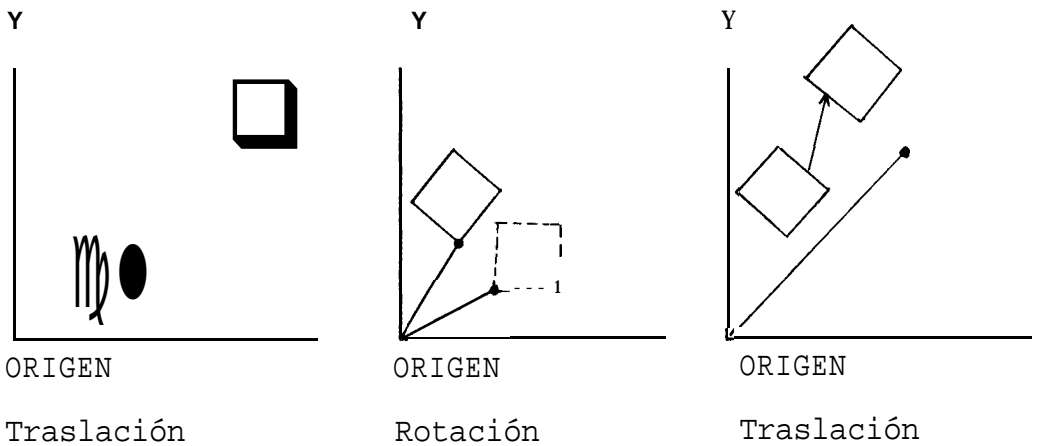


Figura 2.2.3 Rotación en torno al punto (p_x, p_y) .

$$x' = x * \cos(\theta) - y * \sin(\theta)$$

$$y' = y * \cos(\theta) + x * \sin(\theta)$$

Estas ecuaciones constituyen la transformación que rota un punto un ángulo θ en torno al origen en sentido contrario al movimiento de las manecillas del reloj. Para rotar un objeto, cada punto-que lo define debe ser transformado mediante estas ecuaciones, de modo que el objeto es redibujado usando la lista de puntos transformados.

Con frecuencia es necesario rotar un objeto un ángulo θ alrededor de un pivote que no sea el origen. Para lograrlo, se requieren tres pasos (Veáse la Fig. 2.2.1).

1.- Trasládese el pivote (p_x, p_y) al origen. Al hacer esto, cada punto (x, y) que define al objeto es trasladado a un punto nuevo (x', y') donde

$$x' = x - p_x$$

$$y' = y - p_y.$$

Obsérvese que, como se requirió líneas arriba, esta traslación envía el pivote (p_x, p_y) a $(0, 0)$.

2.- Rótese los puntos trasladados (x', y') θ grados alrededor del origen para obtener el punto nuevo (x'', y'') , donde

$$x'' = x' * \cos(\theta) - y' * \text{sen}(\theta)$$

$$y'' = y' * \cos(\theta) + x' * \text{sen}(\theta).$$

Al sustituir los valores anteriores de x' y y' en estas ecuaciones, se obtiene:

$$x'' = (x - px) * \cos(\theta) - (y - py) * \text{sen}(\theta)$$

$$y'' = (y - py) * \cos(\theta) + (x - px) * \text{sen}(\theta).$$

3.- Trasládese el centro de rotación de regreso al punto pivote (px, py) . Así, el punto (x'', y'') es trasladado a (x''', y''') donde

$$x''' = x'' + px$$

$$y''' = y'' + py$$

Si se substituyen los valores de x'' y y'' de las ecuaciones anteriores, se obtiene la ecuación deseada para una rotación en sentido contrario al que giran las manecillas del reloj, de un ángulo θ alrededor del punto (px, py) :

$$x''' = (x - px) * \cos(\theta) - (y - py) * \text{sen}(\theta) + px$$

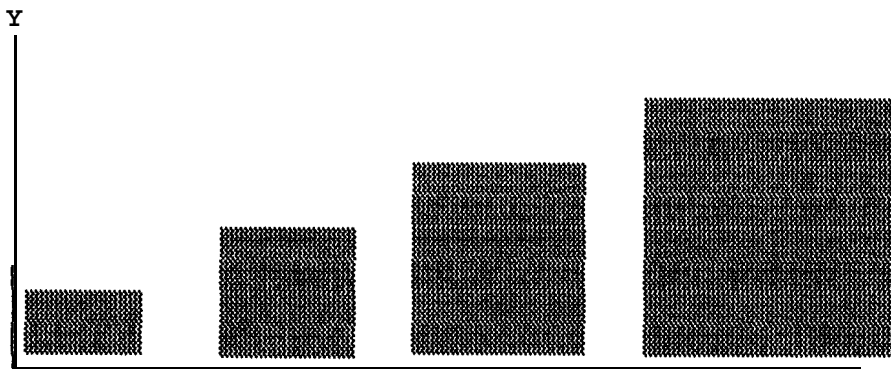
$$y''' = (y - py) * \cos(\theta) + (x - px) * \text{sen}(\theta) + py$$

Sería preguntar por qué se rota el objeto en la dirección antihorario, y no en la dirección horario. La razón es que, en las matemáticas, la dirección antihorario suele considerarse como la dirección de rotación positiva. Si se desean direcciones en

sentido horario, sólo se cambia el ángulo θ a $-\theta$ en las ecuaciones. Recuérdese que $\cos(-\theta) = \cos(\theta)$ y $\sin(-\theta) = -\sin(\theta)$.

2.3 ESCALAMIENTO

Un objeto puede hacerse más grande incrementando la distancia entre los puntos que lo describen (fig. 2.3.1). Por lo general, puede cambiarse el tamaño de un objeto, o de la imagen completa multiplicando las distancias entre los puntos por un factor de amplificación o reducción. Este factor se conoce como **factor de escalamiento**, y la operación que incide en el tamaño se denomina **escalamiento**



PUNTO FIJO

Fig. 2.3.1

Escalamiento

Si el factor de escalamiento es mayor que 1, el objeto se amplifica; si el factor es menor que 1, se reduce; un factor de 1 no tiene efecto alguno sobre el objeto.

Siempre que se realiza un escalamiento, existe un punto que permanece en la misma posición. Se llama a éste punto fijo de transformación de escalamiento (fig. 2.3.1). Si el punto fijo se halla en el origen (0,0), como en la figura 2.3.1, un punto (x,y) puede ser escalado mediante un factor E_x en la dirección x, y E_y en la dirección y, hacia el nuevo punto (x',y) por medio de las multiplicaciones:

$$x' = x * E_x$$

$$y' = y * E_y.$$

E_x y E_y se conocen como factores de escala horizontal y vertical, respectivamente. Si E_x es diferente de E_y , el objeto es una distorsión del original.

Si ambos factores de escalamiento son mayores que 1, el objeto escalado es amplificado y desplazado más allá del punto fijo. Por el contrario, un factor de escalamiento menor que 1 acerca el objeto al punto fijo, además de reducirlo de tamaño. Las figuras 2.3.2 y 2.3.3 ilustran estas características, con el punto fijo en el origen.

Es posible escoger cualquier punto (px,py) como el punto fijo de escalamiento. Para hacer esto, se siguen tres pasos similares a los descritos en la rotación alrededor de un punto arbitrario.

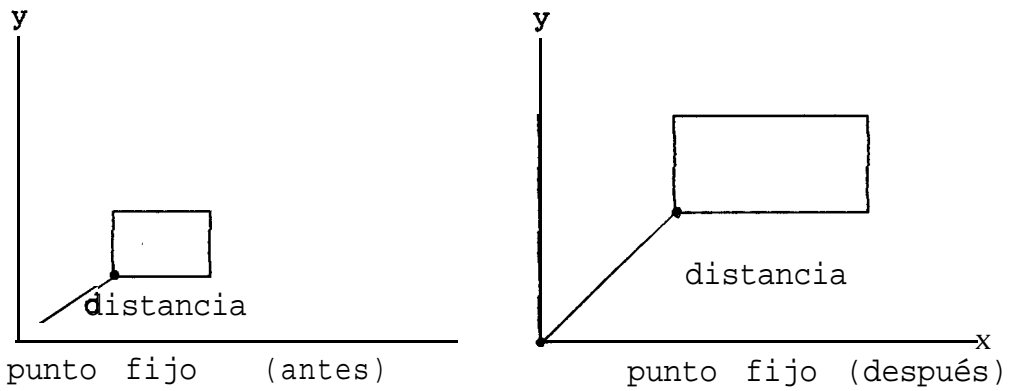


Figura 2.3.2 Escalamiento cambia la distancia a partir de un punto fijo

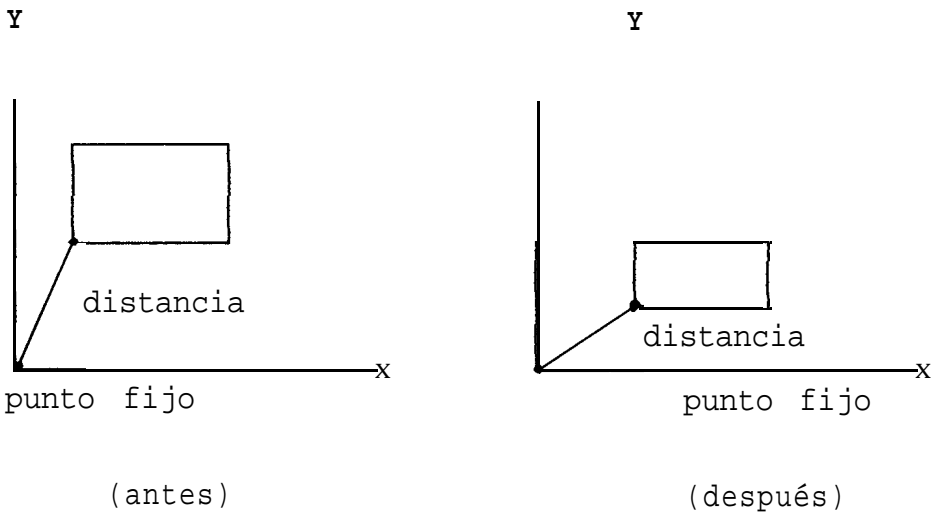


Figura 2.3.3 Escalamiento

- 1.- Trasládese el punto (p_x, p_y) al origen. Todo punto (x, y) se mueve al punto nuevo (x', y') :

$$x' = x - p_x$$

$$y' = y - p_y.$$

- 2.- Escálese los puntos trasladados con el origen

como punto fijo:

$$x'' = x' * E_x$$

$$y'' = y' * E_y.$$

3.- Trasládese el origen de rastreo al punto (p_x, p_y) :

$$x'' = x'' + p_x$$

$$y'' = y'' + p_y.$$

Al substituir los valores de x' , y' , x'' y y'' en las últimas ecuaciones, se obtiene la transformación de escalamiento deseada, con (p_x, p_y) como punto fijo:

$$x'' = (x - p_x) * E_x + p_x$$

$$y'' = (y - p_y) * E_y + p_y.$$

Se puede comprobar que (p_x, p_y) es el punto fijo de esta transformación resolviendo la ecuación de x'' en x y la de y'' en y . Esto es,

$$x = x'' = (x - p_x) * E_x + p_x \text{ ó}$$

$$x = x * E_x - p_x * E_x + p_x \quad \text{ó}$$

$$x * (1 - E_x) = p_x * (1 - E_x) \quad \text{ó}$$

$$x = p_x, \text{ suponiendo que } E_x \text{ sea diferente de } 1.$$

Es posible efectuar una secuencia simétrica de los pasos para demostrar que p_y es la coordenada y del punto fijo.

2.4 AFILAMIENTO

Una transformación de afilamiento produce la distorsión de un objeto, o bien, una imagen completa. Se examinan dos tipos de afilamiento: afilamiento y y afilamiento x . Un afilamiento y transforma el punto (x,y) al punto (x',y') , donde:

$$x' = x$$

$$y' = A_y y \quad x + y, \quad A_y \text{ es diferente de } 0.$$

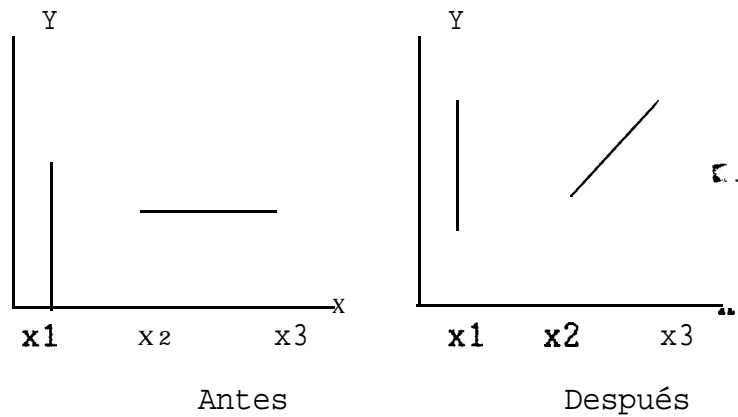


Figura 2.4.1 Afilamiento de una línea vertical y una horizontal

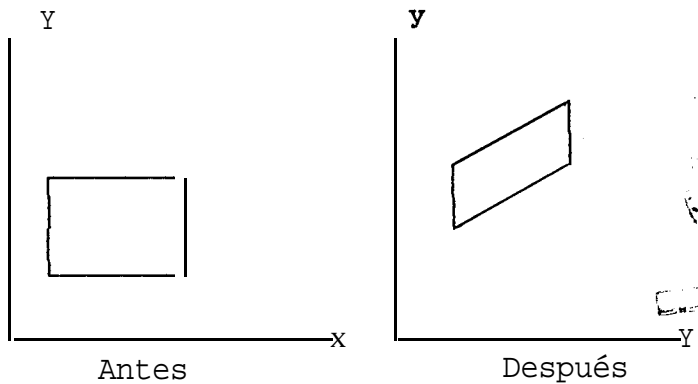


Figura 2.4.2 Afilamiento Y

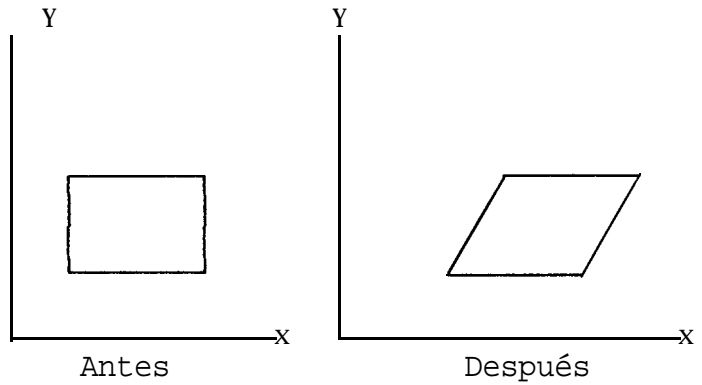


Figura 2.4.3 Afilamiento X

Un afilamiento y desplaza una línea vertical hacia arriba o hacia abajo, dependiendo del signo del factor de afilamiento Afy (Fig. 2.4.1). Una línea horizontal se distorsiona hasta formar una línea inclinada con pendiente Afy .

Combinando esas dos observaciones, se aprecia que el afilamiento y distorsiona un rectángulo hasta darle la forma de un paralelogramo (fig. 2.4.2).

Un afilamiento x tiene el efecto opuesto, esto es, el punto (x, y) se transforma en el punto (x', y') , donde

$$x' = x + Afx * y, \quad Afx \neq 0$$

$$y' = y.$$

Las líneas se vuelven inclinadas con pendiente Afx , en tanto que las horizontales se corren a la derecha o a la izquierda, dependiendo del signo de Afx .

(Figura 2.4.3).

2.5 TRANSFORMACIONES INVERSAS

Para deshacer una transformación, se debe realizar lo contrario de la transformación. La inversa de un transformación es un proceso del mismo tipo, aunque con parámetros inversos. Tales parámetros son:

Trasladar: $-H, -V$
 Rotar: $-\theta$
 Escalar: $1/E_x, 1/E_y$
 Afilar: $-A_{fx}, -A_{fy}$.

2.6 MATRICES

Una matriz es un arreglo rectangular de números como los que se muestran a continuación:

$$[1 \ 4] \quad \begin{bmatrix} 2 & 0 \\ 3 & 8 \end{bmatrix} \quad \begin{bmatrix} 6 & -1 & 2 \\ 0 & 3 & 7 \\ 5 & 8 & 1 \end{bmatrix}$$

Los subarreglos horizontales y verticales dentro de la matriz se conocen como renglones y columnas, respectivamente. Se dice que una matriz con m renglones y n columnas es una matriz $m \times n$. Si m es igual a n , la matriz es cuadrada. Las matrices anteriores son, de izquierda a derecha, 1×2 , 2×2 , y 3×3 .

Un vector es una matriz especial que presenta un

renglón o una columna, esto es m , o n , pero no ambos, igual a 1. Los vectores son importantes, ya que un par de coordenadas (x,y) puede expresarse como un vector renglón $[xy]$, o un vector columna:

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

En este capítulo la segunda matriz B siempre será una matriz cuadrada de 3×3 . Si A es un vector renglón (matriz) del 1×3 , entonces $A * B$ es asimismo una matriz cuadrada de 3×3 .

La **multiplicación** de matrices no es conmutativa; esto es, $A * B$ puede no ser igual a $B * A$. Una matriz cuadrada con varios 1 en la diagonal que va de arriba a la izquierda hasta abajo a la derecha, conocida como la diagonal principal, y ceros en cualquier otro lugar, es la matriz identidad, denotada por I :

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

La inversa de una matriz cuadrada A es la matriz A^{-1} con la propiedad

$$A^{-1} * A = A * A^{-1} = I$$

La inversa de una matriz cuadrada puede no existir, pero cuando existe, el método general para

encontrarla puede exigir muchas operaciones de punto flotante. La inversa del producto de dos matrices cuadradas es el producto de las inversas en orden contrario:

$$(A * B)^{-1} = B^{-1} * A^{-1}.$$

La **traspuesta** de una matriz A es la matriz A^T , la cual se forma a partir de A intercambiando sus renglones y columnas.

La traspuesta del producto del producto de dos matrices es el producto de la traspuesta de las matrices en orden inverso:

$$(A * B)^T = B^T * A^T.$$

2.7 REPRESENTACION MATRICIAL DE LAS TRANSFORMACIONES

Cada una de las transformaciones que se han presentado, excepto la traslación, puede representarse como un producto del vector renglón $[x \ y]$ y una matriz de $2 * 2$. No obstante, se pueden representar las cuatro transformaciones como el producto del vector renglón $1 * 3$ y una matriz apropiada de $3 * 3$. La ventaja de usar esta representación uniforme (homogénea) se hará evidente en la siguiente sección, cuando se combinen las transformaciones.

Ya que se van a usar matrices de 3×3 , es necesario convertir el par de coordenadas bidimensionales (x,y) en un **vector homogéneo** tridimensional. Esto se logra asociando (x,y) con el vector renglón homogéneo $[x \ y \ 1]$. Después de multiplicar este vector por una matriz de 3×3 , se obtiene otro vector renglón homogéneo de tres componentes, con el último igual a 1: $[x' \ y' \ 1]$. Los primeros dos términos de este vector son el par de coordenadas (x',y') , que es la transformación de (x,y) . Esta representación tridimensional (no única) del plano bidimensional se conoce como la representación de **coordenadas homogéneas**. Puede darse ahora la representación matricial de las cuatro transformaciones: traslación, rotación, escalamiento y afilamiento.

Traslación:

$$[x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} 1 & 0 & H \\ 0 & 1 & V \\ 0 & 0 & 1 \end{bmatrix}$$

Rotación:

$$[x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Escalamiento:

$$[x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} E_x & 0 & 0 \\ 0 & E_y & 0 \\ 0 & & 01 \end{bmatrix}$$

Afilamiento y:

$$[x' \ y \ 1] = [x \ y \ 1] \begin{bmatrix} 1 & Af_y & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Afilamiento x:

$$[x' \ y \ 1] = [x \ y \ 1] \begin{bmatrix} 1 & 0 & 0 \\ Af_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Las matrices inversas pueden obtenerse de manera directa sustituyendo los parámetros inversos adecuados en las matrices de **transformación** de 3×3 .

2.8 COMBINACION DE TRANSFORMACIONES

Cualesquiera de las transformaciones anteriores pueden combinarse para producir nuevas transformaciones. Por ejemplo, una rotación alrededor de un punto pivote arbitrario requiere una traslación, seguida de una rotación, seguida de otra traslación (la inversa de la primera). Contando con la representación matricial de una transformación, las transformaciones pueden ser combinadas multiplicando sus matrices correspondientes. (Esta es la razón por la que todas las transformaciones bidimensionales se representan como matrices de 3×3). Tal relación entre combinación de

transformaciones y multiplicación de matrices hace particularmente atractiva la representación matricial. Es mucho más fácil hacer multiplicación de matrices que realizar las sustituciones necesarias, para combinar transformaciones.

Existen dos formas de combinar transformaciones. Cada una produce el mismo resultado, aunque una de éstas es más eficiente. A manera de ejemplo, supóngase que una rotación Rot es seguida por una traslación Tr. Por lo tanto,

$$[x' \ y' \ 1] = [x \ y \ 1] * \text{Rot} * \text{Tr}.$$

El primer método multiplica el vector renglón por la matriz de rotación, obteniendo el vector intermedio:

$$[x' \ y' \ 1] = [x'' \ y'' \ 1] * \text{Tr}.$$

Este paso requiere **nuve** multiplicaciones y seis sumas más, resultando en un total de 18 multiplicaciones y 12 sumas. Estos dos pasos deben aplicarse **a** cada par (x,y) en la lista que define **el(los)** objeto(s) transformado(s). En el caso de listas que contienen cientos o **miles** de puntos, debe realizarse una cantidad considerable de aritmética de punto flotante.

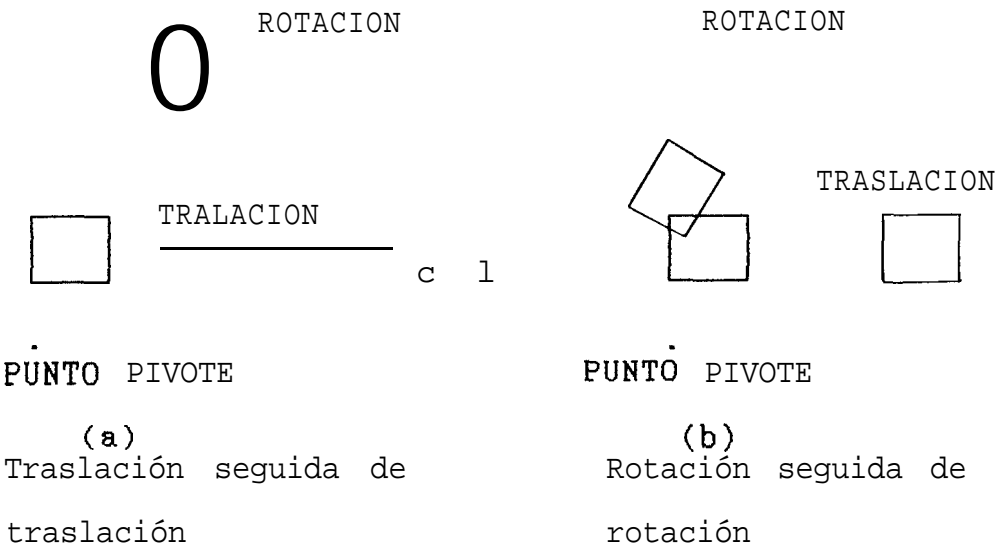
El segundo método reduce el número de operaciones aritméticas combinando, primero que nada, las dos

matrices de 3 * 3. La multiplicación Rot * Tr requiere de 27 multiplicaciones y 18 sumas. La observación importante por hacer aquí es que esta multiplicación es realizada sólo una vez y no para cada par (x,y) en la lista del objeto. Ahora la matriz combinada se multiplica a continuación por el vector renglón:

$$[x' \ y' \ 1] = [x \ y \ 1] * (Rot * Tr).$$

Esta multiplicación de matrices requiere nueve multiplicaciones y seis sumas. Por lo tanto, excepto para la multiplicación Rot * Tr, que se realiza sólo una vez, el segundo método de combinación de transformaciones puede reducir a la mitad el número de operaciones aritméticas.

Figura 2.8 Cambio de orden de una traslación y una rotación



Dado que la multiplicación de matrices no es conmutativa, siempre que se combinan diferentes tipos de transformaciones debe tenerse cuidado con el orden en **que** se realizan. La figura 2.8 ilustra una traslación, seguida de una rotación, y una rotación seguida de una traslación.

2.9 PROCEDIMIENTO DE TRANSFORMACIONES

Quienquiera que haya intentado escribir procedimiento para multiplicación de matrices sabrá que se trata de una operación muy lenta. Por esta y otras razones, la mayoría de los sistemas de **graficación** cuentan con **tales** transformaciones y realizan la multiplicación en código de bajo nivel o en hardware especializado. Estas transformaciones se designan con órdenes del sistema de coordenadas del mundo:

trasladar(H,V)

con H unidades horizontales y V unidades verticales;

rotar(θ)

en una dirección antihorario alrededor del origen; θ está dada en radianes;

escalar(E_x, E_y)

con E_x unidades en el eje x, E_y unidades en el eje y,

con el origen como punto fijo.

Antes de usar estas transformaciones, es necesario comentar la forma en que afectan a las órdenes de dibujo. Todas las transformaciones son acumulativas y actúan, no en las coordenadas iniciales, sino en las coordenadas transformadas. Por ejemplo, si se había realizado previamente una traslación y ahora se efectúa una rotación, la matriz de rotación, y la matriz nueva se usa para futuras transformaciones.

CAPITULO III

REPRESENTACIONES TRIDIMENSIONALES



3.1 INTRODUCCION

El mundo se compone de **imagenes** tridimensionales. Los objetos no sólo tienen altura y anchura, sino también profundidad. La presentación de objetos tridimensionales en pantallas bidimensionales paracería una tarea imposible: si la altura y la anchura están representados **por** coordenadas x , y , ¿Cómo podría representarse la tercera dimensión, que es la profundidad?.

Las técnicas empleadas en la **graficación** por computador para representar este universo tridimensional se basan en los mismos **principios** a los que un artista o un fotógrafo recurre al crear una imagen realista sobre papel o película. La diferencia es que el computador se vale de un **modelo** matemático en vez del pincel o los lentes.

A medida que incrementa al realismo de una imagen generada por computador, también se incrementa la complejidad de las matemáticas usadas en su



realización.

3.2 SISTEMAS DE COORDENADAS

Una imagen tridimensional suele definirse **por** los puntos, líneas **y** planos que la componen. Para especificar estos primitivos elementos de construcción, se necesita un sistema de coordenadas tridimensional, el cual puede concebirse como una extensión del sistema bidimensional. Así pues, la tercera dimensión, que es la profundidad, se representará mediante el eje **z**, que se halla en un ángulo recto respecto del plano de coordenadas **x,y**. Cualquier punto en este sistema de coordenadas rectangular puede describirse por medio de una tríada ordenada **(x,y,z)** de valores.

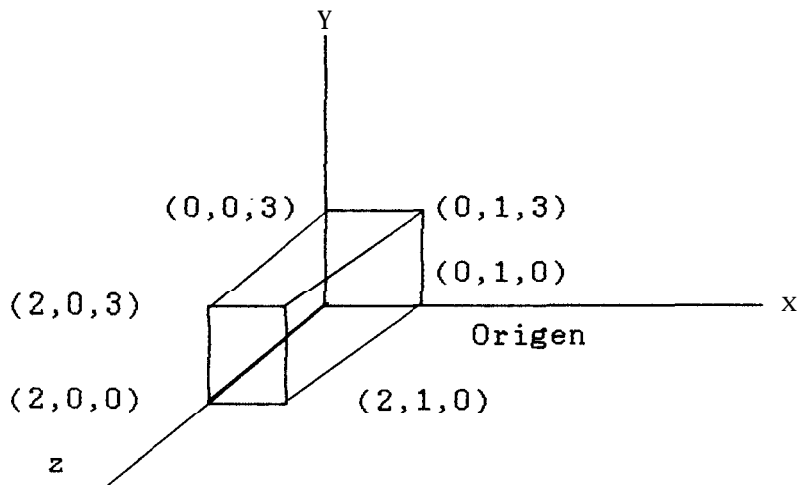
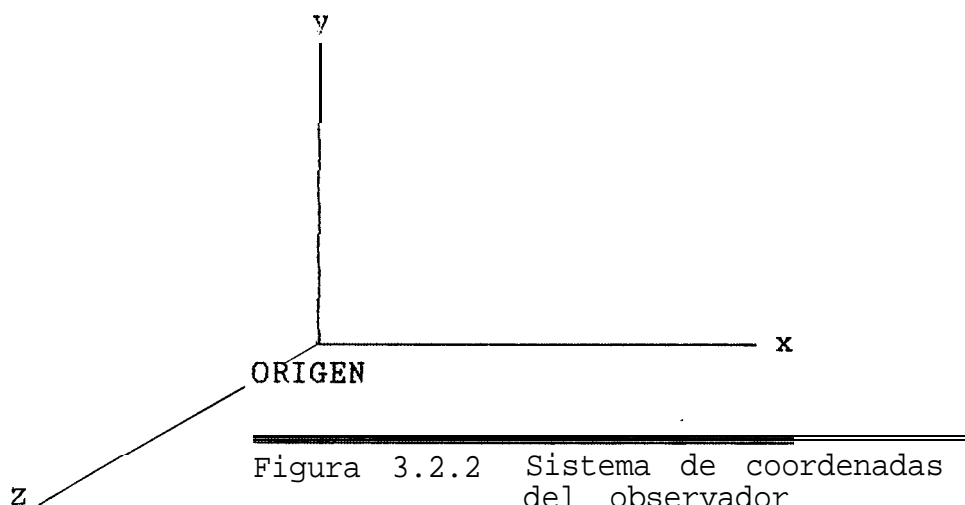


Figura 3.2.1 Sistema de coordenadas tridimensional

El sistema de coordenada ilustrado es de **mano**

derecha(Figura 3.1.2). Si se cierran los dedos de la mano derecha en el sentido que va del eje positivo x al eje positivo y , el pulgar apunta en la dirección del eje positivo z . La selección de un sistema de mano derecha es arbitraria y se usa principalmente porque los **matemáticos** se sienten familiarizados con él. Este se conoce también como **sistema de coodenadas del mundo**.

Existe otro sistema de coordenadas tridimensional que se usa para presentar imágenes generadas por el computador. La pantalla es un plano bidimensional con el origen en la esquina inferior izquierda, el eje y positivo hacia arriba y el eje x positivo hacia la derecha. El eje z empieza en el origen y apunta al interior de la pantalla. Este sistema de coordenadas tiene la propiedad de que, cuando la pantalla es visualizada, los objetos que están en el fondo adoptan valores positivos z más grandes. Se conoce a este sistema como de **mano izquierda**, ya que si se cierran los dedos de la mano **izquierda** en la trayectoria de x a y , el pulgar apunta en la dirección z . Tal sistema de coordenadas se puede definir de manera independiente de la pantalla de **presentación**, y se la conoce también como **sistema de coordenadas del observador**, ya que el ojo del observador está en el origen (Figura 3.2.2).



3.3 SUPERFICIES DE POLIGONOS

Cualquier **objeto** tridimensional puede representarse como un conjunto de superficies poligonales planas. Para algunos objetos, como un polihedro, esto define precisamente las características de la superficie. En otros casos, una representación de un polígono ofrece una descripción aproximada del objeto.

TABLAS DE POLIGONOS

Una vez que el usuario haya definido cada superficie del polígono, el paquete de gráficas organiza los datos de entrada en las tablas que se utilizarán en el procesamiento y despliegue de las superficies. Las tablas de datos contienen las propiedades geométricas y **de** atributos del objeto, organizadas para facilitar el procesamiento. Las tablas de datos geométricos contienen coordenadas y **parámetros** de fronteras para identificarla orientación en el

espacio de las superficies poligonales. La información de atributo del objeto incluye designaciones de cualquier modelo de color y sombreado que se aplicará a las superficies.

Los datos geométricos están organizados en tres listas: una tabla de vértices, una de aristas y una de polígonos. Los valores coordenados de cada vértice del objeto se almacenan en la tabla de vértices. La tabla de aristas **enlista** los vértices extremos que definen a cada arista. Cada polígono se define en la tabla de polígonos como una lista de aristas componentes. Este esquema se detalla en el capítulo VII .

ECUACIONES DE PLANOS

Los parámetros **que** especifican la orientación espacial de cada polígono se obtienen de los valores coordenados de los vértices y las ecuaciones que definen los planos poligonales. Estos parámetros de planos se utilizan en transformaciones de visión, modelos de sombreado y algoritmos de superficies ocultas que determinan qué líneas y planos se traslapan a lo largo de la línea de visión.

La ecuación de una superficie plana puede expresarse así:

$$AX + BY + CZ + D = 0$$

donde (x,y,z) es cualquier punto del plano. Los coeficientes A,B,C,D son constantes que pueden calcularse utilizando los valores coordenados de tres puntos no colineales en el plano. Comúnmente, se usan las coordenadas de tres vértices sucesivos en una frontera de un polígono para hallar valores de estos coeficientes. Al denotar las coordenadas de tres vértices de un polígono como (x_1,y_1,z_1) , (x_2,y_2,z_2) y (x_3,y_3,z_3) , se puede resolver el siguiente conjunto de ecuaciones planas simultáneas para las razones A/D , B/D y C/D :

$$(A/D)X_i + (B/D)Y_i + (C/D)Z_i = -1; \quad i=1,2,3$$

Utilizando un método de solución como la regla de Cramer, se puede escribir la solución de los parámetros del plano en forma de determinantes:

$$A = \begin{vmatrix} 1 & Y_1 & Z_1 \\ 1 & Y_2 & Z_2 \\ 1 & Y_3 & Z_3 \end{vmatrix}$$

$$B = \begin{vmatrix} X_1 & 1 & Z_1 \\ Y_1 & 1 & Z_2 \\ Z_1 & i & Z_3 \end{vmatrix}$$

$$C = \begin{vmatrix} X_1 & Y_1 & i \\ Y_1 & Y_2 & . \\ Z_1 & Y_3 & i \end{vmatrix}$$

$$D = \begin{vmatrix} X_1 & Y_1 & Z_1 \\ Y_1 & Y_2 & Z_2 \\ Z_1 & Y_3 & Z_3 \end{vmatrix}$$

Podemos ampliar los determinantes y escribir los cálculos de los coeficientes del plano en la forma explícita:

$$A = y_1(z_2 - z_3) + y_2(z_3 - z_1) + y_3(z_1 - z_2)$$

$$B = z_1(x_2 - x_3) + z_2(x_3 - x_1) + z_3(x_1 - x_2)$$

$$C = -x_1(y_2z_3 - y_3z_2) - x_2(y_3z_1 - y_1z_3) - x_3(y_1z_2 - y_2z_1)$$

Los valores de **A, B, C** y **D** se almacenan en la estructura de datos que contiene la información de coordenadas y atributos referente al polígono definido en este plano.

La orientación de una superficie plana se especifica por medio del vector normal al plano, como se indica en la figura 3.3.1. Este vector normal tridimensional tiene las coordenadas cartesianas **(A, B, C)**.

Puesto que con frecuencia trabajamos con superficies poligonales **que** encierran un objeto interior se necesita distinguir entre los dos lados de la superficie. El lado del plano que da la cara al objeto interior se denomina interior y el lado visible o externo se llama exterior. Si se especifican vértices en un sentido igual al del reloj cuando se observa el lado externo del plano en un sistema coordinado por la derecha, la dirección del vector normal irá de adentro hacia afuera. Esto se

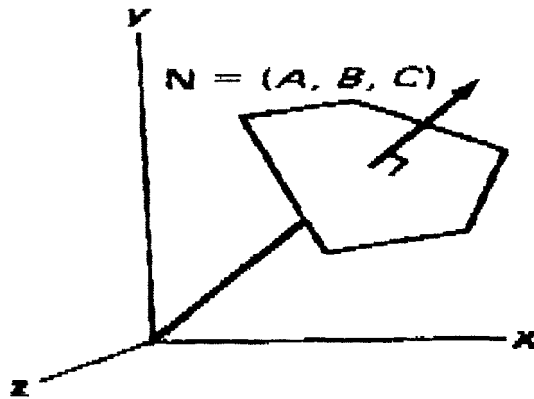
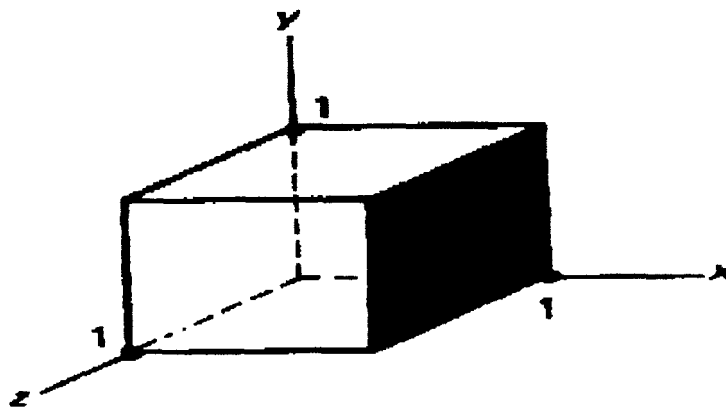


FIGURA 3.3.1

El vector N , normal a la superficie de un plano descrito por la ecuación $Ax + By + Cz + D = 0$ tiene las coordenadas (A, B, C) .

FIGURA 3.3.2

La superficie poligonal sombreada del cubo unitario tiene la ecuación del plano $x - 1 = 0$ y el vector normal $(1, 0, 0)$.



demuestra para un plano de un cubo unitario en la figura 3.3.2.

Para determinar las componentes del vector normal de la superficie sombreada que se muestra en la figura 3.3.2, se seleccionan tres de los cuatro vértices situados a lo largo de la frontera del polígono. Los vértices deben de ser no colineales.

Estos puntos se seleccionan en un sentido igual al del reloj cuando observamos del exterior del cubo hacia el origen. Las coordenadas de estos vértices, en el orden seleccionado, se utilizan en las ecuaciones anteriores a fin de obtener los coeficientes del plano:

$$A = 1$$

$$B = 0$$

$$C = 0$$

$$D = -1$$

El vector normal de este plano está en el sentido del eje x positivo. Las ecuaciones del plano se utilizan también para identificar puntos interiores y exteriores.

Cualquier punto (x, y, z) exterior a un

plano satisface la desigualdad

$$AX + BY + CZ + D > 0$$

Análogamente, cualquier punto situado en el interior del plano produce un valor negativo de la expresión $AX + BY + CZ + D$. para la superficie sombreada de la figura 3.3.2, cualquier punto exterior al **plano** cumple la desigualdad $x - 1 < 0$, mientras que cualquier punto interior al plano tiene un valor de coordenada $x < 1$.

3.4 SUPERFICIES CURVAS

Los despliegues tridimensionales de las superficies curvas pueden generarse a partir de un conjunto de entrada de las funciones matemáticas que **define las** superficies o bien a partir de un conjunto de **puntos** de datos especificados por el usuario. Cuando se especifican funciones de curvas, un paquete puede emplear las ecuaciones definidoras para localizar y **graficar** posiciones de píxeles a lo largo de las trayectorias de la curva.

Podemos representar una línea curva tridimensional en forma analítica con la pareja de funciones

$$y = f(x) \quad z = g(x)$$

Con la **coordemada** x seleccionada como variable independiente. Los valores de las variables

dependientes y z se determina **después** a partir de las ecuaciones anteriores a medida que se avanza **a** través de valores de x de un extremo de la línea al otro. Esta representación tiene algunas desventajas. Si se desea una gráfica aislada, se debe cambiar la variable independiente siempre que la primera **derivada(pendiente)** de $f(x)$ o bien $g(x)$ se vuelva mayor que 1. Esto significa que se deben verificar continuamente los valores de las derivadas, **que** pueden volverse infinitas en algunos puntos. Asimismo, las ecuaciones anteriores ofrecen **un** formato desproporcionado para representar funciones con valores múltiples. Una representación más propicia de las curvas para aplicaciones de las gráficas es en términos de ecuaciones paramétricas.

ECUACIONES **PARAMETRICAS**

Mediante la introducción de un cuarto parámetro, u , en la descripción coordenada de una curva, se puede expresar cada una de las tres coordenadas cartesianas en forma paramétrica. Cualquier punto de la curva puede representarse entonces por medio de la función vectorial

$$P(u) = (x(u), y(u), z(u))$$

Por lo general, las ecuaciones paramétricas se

constituyen de manera que el parámetro u se defina en el intervalo de 0 a 1 . Por ejemplo, una circunferencia en el plano xy con centro en el origen coordenado podría definirse en forma paramétrica como

$$x(u) = r\cos(2\pi u), \quad y(u) = r\sen(2\pi u), \quad z(u) = 0$$

También son posibles otras formas paramétricas para describir circunferencias y arcos circulares. En el caso de una curva arbitraria, puede ser difícil idear un conjunto de ecuaciones paramétricas que define completamente la forma de la curva. Pero cualquier curva puede aproximarse utilizando diferentes conjuntos de funciones paramétricas sobre partes diferentes de la curva. Por lo general estas aproximaciones se forman con funciones polinomiales. Dicha construcción **por** partes de una curva debe implantarse cuidadosamente para asegurar que haya una transición sencilla de una sección de la curva a la siguiente. La uniformidad de una curva puede describirse **a** partir de la continuidad de la curva entre las secciones. La continuidad de orden cero se refiere simplemente **a** que las curvas se interceptan. Continuidad de primer orden significa que las líneas tangentes (primeras derivadas) de dos secciones adyacentes de la curva son las mismas en el punto de adyacencia. Continuidad de segundo orden quiere

decir que las curvaturas (segundas derivadas) de las dos secciones de la curva son las mismas en la intersección. La figura 3.4.1 muestra ejemplos de los tres órdenes de continuidad.

Las ecuaciones paramétricas de las superficies se formulan con dos parámetros, u y v . Una posición coordenada de una superficie se representa entonces por medio de la función vectorial paramétrica.

$$P(u, v) = (x(u, v), y(u, v), z(u, v))$$

Las ecuaciones de las coordenadas x , y y z a menudo se acomodan de modo que los parámetros u y v estén definidos dentro del intervalo de 0 a 1. Por ejemplo, una superficie esférica puede describirse con las ecuaciones

$$x(u, v) = r \operatorname{sen}(\pi u) \cos(2\pi v)$$

$$y(u, v) = r \operatorname{sen}(\pi u) \operatorname{sen}(2\pi v)$$

$$z(u, v) = r \cos(\pi u)$$

Donde r es el radio de la esfera. El parámetro u describe líneas de latitud constante sobre la superficie, mientras que el parámetro v describe líneas de longitud constante. Al mantener uno de estos parámetros fijo mientras se varía el otro sobre cualquier valor del intervalo de 0 a 1, se podría trazar líneas de latitud y longitud de cualquier

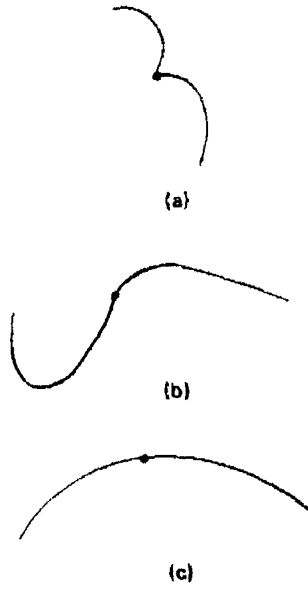
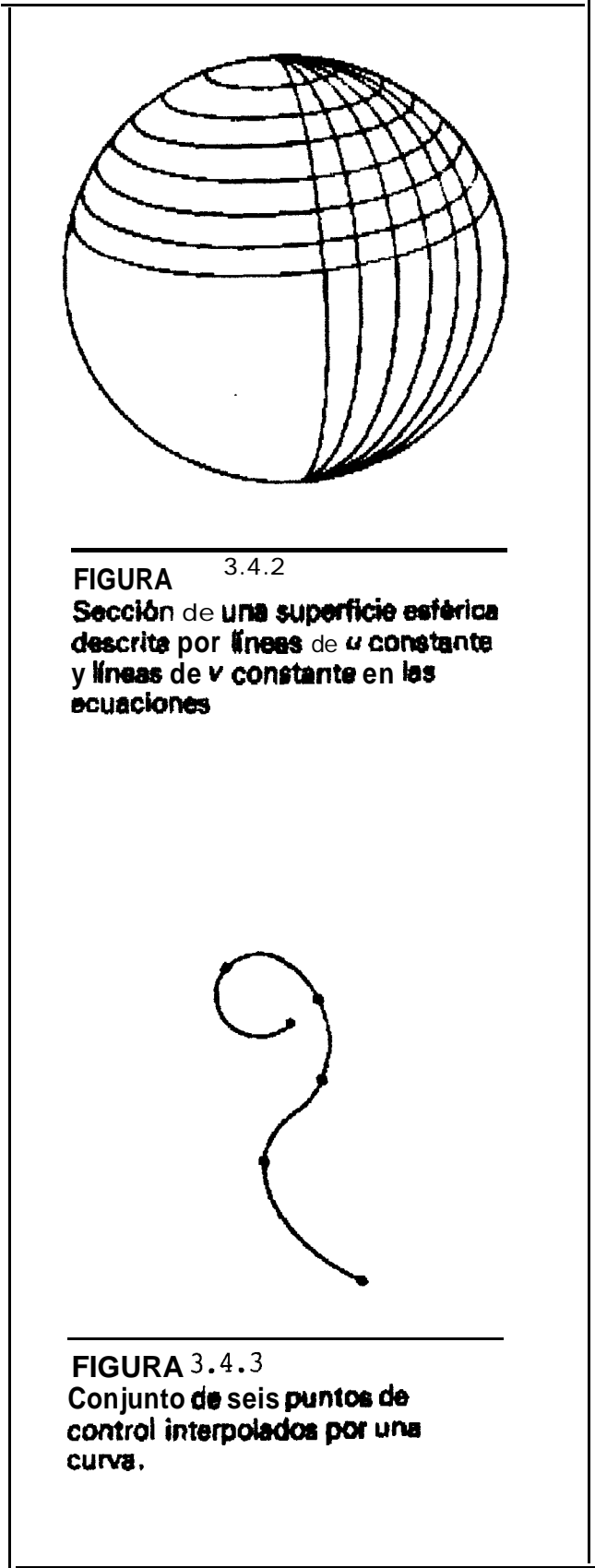
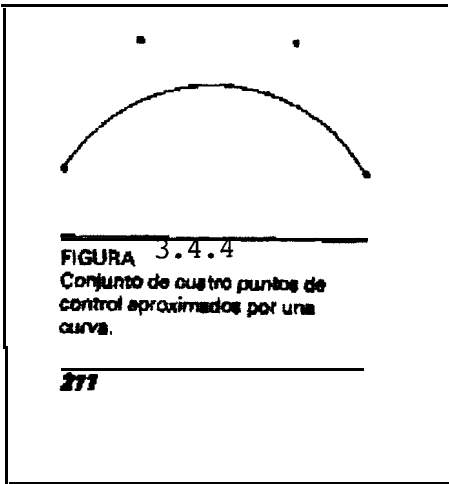


FIGURA 3.4.1

Especificación por partes de una curva mediante la conexión de dos segmentos de curva con órdenes de continuidad variantes: (a) sólo continuidad de orden cero; (b) continuidad de primer orden y (c) continuidad de segundo orden.



sección esférica (Figura 3.4.2).

En aplicaciones de **diseño**, una curva o superficie a menudo se define especificando interactivamente un conjunto de puntos de control, los cuales indican la forma de las curvas. Estos puntos de control son usados por el paquete para formar ecuaciones paramétricas polinomiales para desplegar la curva definida. Cuando la curva desplegada pasa a través de los puntos de control, como se indica en la figura 3.4.3, se dice que ésta interpola los puntos de control. Por otro lado, se dice que los puntos de control se aproximan si la curva desplegada pasa cerca de ellos (Figura 3.4.4).

CAPITULO IV

TRANSFORMACIONES TRIDIMENSIONALES

4.1 TRASLACION

La transformación que traslada un punto (x,y,z) T_x unidades en la dirección x , T_y unidades en la dirección y y T_z Unidades en la dirección z hacia el punto nuevo (x',y',z') se representan por medio de la matriz.

$$\text{Tr}(T_x, T_y, T_z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

El punto homogéneo trasladado es

$$[x', y', z', 1] = [x, y, z, 1] \times \text{Tr}(T_x, T_y, T_z)$$

donde

$$\begin{aligned} x' &= x + T_x \\ y' &= y + T_y \\ z' &= z + T_z \end{aligned}$$

La matriz de traslación inversa se obtiene invirtiendo el signo de los valores de T_x, T_y, T_z . Así

para trasladar el punto (x',y',z') de regreso a (x,y,z) se multiplica por el homogéneo $[x',y',z',1]$ Por la matriz $Tr(-Tx,-Ty,-Tz)$.

4.2 ROTACION

Una **rotación** tridimensional se efectúa alrededor de un eje. Por ahora, se **considerarán** sólo en los que el eje de rotación es uno de los tres **ejes de** coordenadas: x,y o z . El ángulo de rotación será positivo si se realiza en un eje de rotación desde el origen, y se lleva a cabo una rotación en sentido contrario a las manecillas del reloj (Figura 4.2.1).

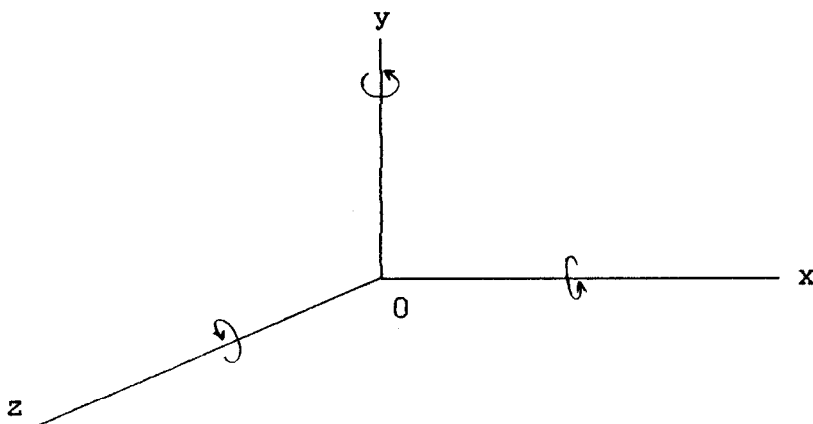


Figura 4.2.1 Trayectorias de las rotaciones tridimensionales positivas.

Una rotación bidimensional en torno al origen equivale a una rotación tridimensional en el plano X,y en torno al eje z . En general, la rotación del punto alrededor del eje se realiza en un plano

perpendicular a éste; esto significa que el punto permanece en este plano.

La rotación de un ángulo θ sobre el eje z se representa por medio de la matriz.

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & \text{sen}(\theta) & 0 & 0 & 0 \\ -\text{sen}(\theta) & \cos(\theta) & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Aplicando $R_z(\theta)$ a un punto homogéneo $[x, y, z, 1]$ se obtiene el punto transformado

$$[x', y', z', 1] = [x, y, z, 1] \times R_z(\theta)$$

donde

$$x' = x \cos(\theta) - y \text{sen}(\theta)$$

$$y' = x \text{sen}(\theta) + y \cos(\theta)$$

$$z' = z.$$

Nótese que el valor de la coordenada z no cambia. Esto se debe a que la rotación alrededor del eje z se realiza en un plano paralelo al plano (x, y) .

La rotación de un punto alrededor del eje se realiza en un plano paralelo y, z . El valor de la coordenada Y no cambia. La matriz de rotación es

$$Ry(\theta) = \begin{bmatrix} \cos(\theta) & 0 & -\text{sen}(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ \text{sen}(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

El punto homogéneo $[x, y, z, 1]$ se convierte en un punto homogéneo $[x', y', z', 1]$ donde

$$x' = x\cos(\theta) + z\text{sen}(\theta)$$

$$y' = y$$

$$z' = x\text{sen}(\theta) + z\cos(\theta).$$

La rotación de un punto alrededor del eje x sucede en un plano paralelo al plano y, z . El valor de la coordenada x no cambia. Su matriz de rotación es

$$Rx(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \text{sen}(\theta) & 0 \\ 0 & -\text{sen}(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

El punto homogéneo $[x, y, z, 1]$ se convierte en el punto homogéneo $[x', y', z', 1]$ donde

$$x' = x$$

$$y' = y\cos(\theta) + z\text{sen}(\theta)$$

$$z' = y\text{sen}(\theta) + z\cos(\theta)$$

Basta con usar planos de coordenadas correspondientes para realizar cálculos. A fin de poder visualizar la

geometría tridimensional, el eje de rotación deberá ser perpendicular al papel (Figura 4.2.2).

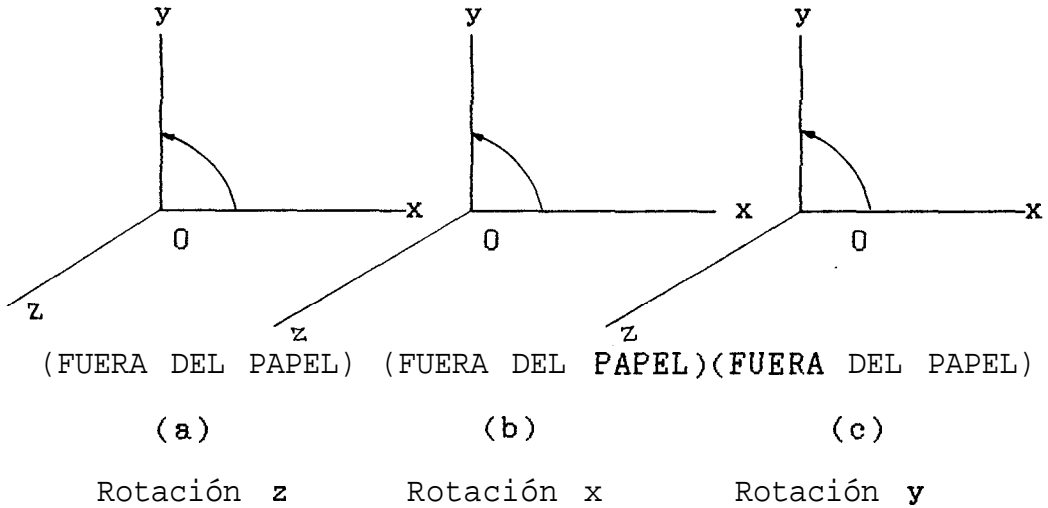


Figura 4.2.2 Eje de rotación perpendicular al papel

La inversa de un rotación de θ grados es simplemente una rotación de $-\theta$ grados (o grados en la dirección de las manecillas del reloj). Las matrices inversas se obtienen reemplazando θ por $-\theta$ en la matriz correspondiente. Como $\cos(-\theta)$ y $\sin(-\theta) = -\sin(\theta)$ es necesario cambiar los signos sólo en los términos seno.

4.3 ESCALAMIENTO

El escalamiento tridimensional permite la contracción o la expansión en cualesquiera de las direcciones x, y o z . Como en el caso bidimensional, se describe

el escalamiento cuando el punto fijo es el origen. Para obtener un escalamiento con un punto fijo arbitrario, éste se debe trasladar al origen, escalar el objeto y después realizar el inverso de la traslación original. La matriz de **escalamiento** con factores de escala E_x, E_y, E_z en las direcciones x, y, z respectivamente, está dada por la matriz.

$$Es(E_x, E_y, E_z) = \begin{bmatrix} E_x & 0 & 0 & 0 \\ 0 & E_y & 0 & 0 \\ 0 & 0 & E_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

El punto homogéneo $[x, y, z, 1]$ se transforma en el punto homogéneo $[x', y', z', 1]$ donde

$$x' = x \cdot E_x$$

$$y' = y \cdot E_y$$

$$z' = z \cdot E_z$$

La inversa de un escalamiento se obtiene usando los recíprocos de los factores de escala:

$$1/E_x, 1/E_y, 1/E_z.$$

4.4 ROTACION EN TORNO A UN EJE ARBITRARIO

Se han descrito las transformaciones alrededor de cualquiera de los tres ejes de coordenadas. Sin embargo, existen situaciones en las que el eje de

rotación no es ninguno de estos. Asumamos que los puntos $P1:(x1,y1,z1)$ y $P2:(x2,y2,z2)$ definen una recta (Figura 4.4.1). Las ecuaciones paramétricas de la recta que pasa por estos puntos son

$$x = (x2 - x1)t + x1$$

$$y = (y2 - y1)t + y1$$

$$z = (z2 - z1)t + z1$$

donde t supone valores reales. Si $t=0$, se alcanza el punto $P1$, mientras que $t=1$ proporciona el punto $P2$. El segmento de recta que va de $P1$ a $P2$ tiene valores de t restringidos entre 0 y 1.

Las tres ecuaciones anteriores pueden convertirse en una forma algo diferente, lo que conviene para nuestros objetivos. Sea

$$a = (x2 - x1)$$

$$b = (y2 - y1)$$

$$c = (z2 - z1)$$

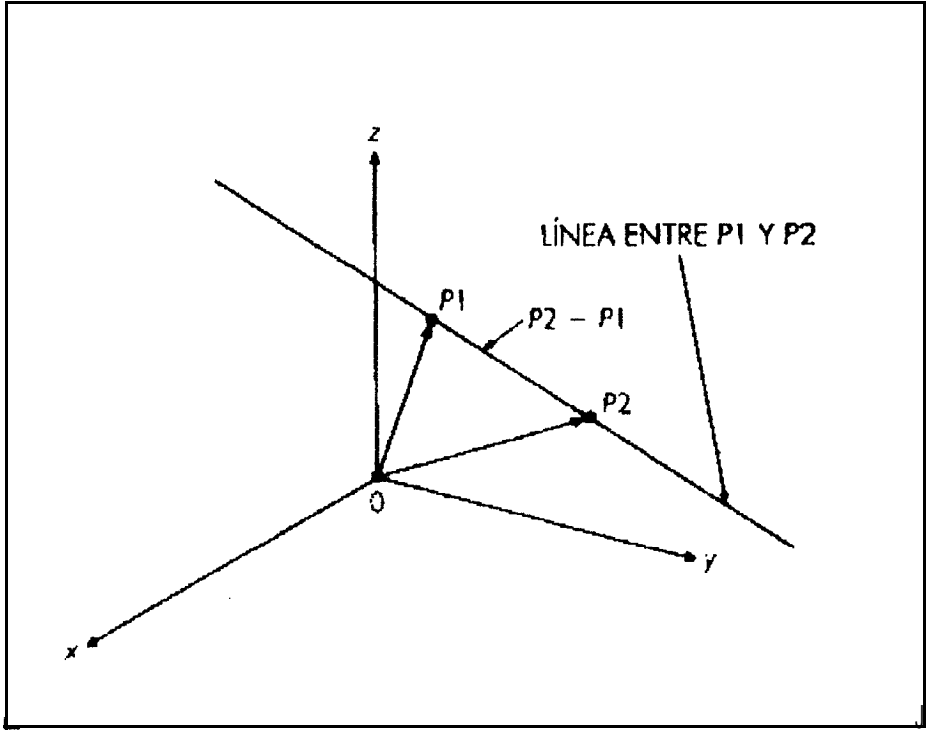
Ahora las ecuaciones de la recta tiene forma

$$x = at + x1$$

$$y = bt + y1$$

$$z = ct + z1$$

Obsérvese que $P1$ y $P2$ pueden considerarse como vectores, y que la diferencia:



BIBLIOTECA

Figura 4.4.1 Representación vectorial de la recta



$$P2 - P1 = (x2 - x1, y2 - y1, z2 - z1) = (a, b, c)$$

es el vector (dirección) de $P1$ a $P2$ sobre la recta que describe la misma trayectoria (Figura 4.4.1). Por lo tanto, son los tres valores a, b, c los que describen la **dirección** de la recta que transcurre del punto $(x1, y1, z1)$ al punto $(x2, y2, z2)$. De aquí que una recta puede definirse por medio de un punto superpuesto $(x1, y1, z1)$ y por una dirección (a, b, c) .

Volviendo con el eje de rotación, sea $(x1, y1, z1)$ un punto sobre el cual pasa el eje de rotación u cuyo vector de dirección es (a, b, c) . La rotación de un ángulo θ alrededor de este eje puede describirse por medio de los siete pasos siguientes:

1. Trasladar el punto $(x1, y1, z1)$ al origen
2. Rotar en torno al eje x hasta que el eje de rotación alcance el plano x, z .
3. Rotar sobre el eje y hasta que el eje de rotación corresponda al eje z .
4. Rotar sobre el eje z un ángulo θ .
5. Realizar la rotación inversa del paso 3.
6. Realizar la rotación inversa del paso 2.
7. Realizar la rotación inversa del paso 1.

Estos pasos transforman el eje de rotación en eje z y realizan la rotación del ángulo θ , para después

transformar el eje de rotación en su posición original. Pongamos ahora en practica los siete pasos. La traslación inicial se representa por medio de la matriz.

$$\text{Tr}(-x_1, -y_1, -z_1) = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10 \\ -x_1 & -y_1 & -z_1 & 1 \end{bmatrix}$$

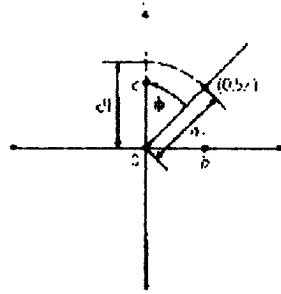
Déspues de esta traslación, el vector de dirección (a, b, c) que define el eje de rotación está en posición ilustrada en la figura 4.4.2 a. Como el paso 2 rota esta recta alrededor del eje x , se puede usar su proyección en el plano y , y considerar esto como una rotación en torno al origen, con el eje x saliendo del papel. Cuando el eje de rotación se proyecta sobre el plano y y z . Cualquier punto sobre éste tiene la coordenada x igual a cero. En particular, $a=0$.

Esta recta proyectada, o lo que es lo mismo, el punto $(0, b, c)$ se rota ϕ grados hasta que la recta corresponda al eje z . En realidad no es necesario encontrar el ángulo ϕ sino el seno y el coseno del ángulo ya que estas funciones trigonométricas delinean la matriz de rotación. La figura 4.4.2 muestra que la distancia entre el origen y $(0, b, c)$ es $\sqrt{b^2 + c^2}$ dl. De aquí

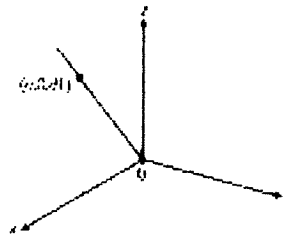
FIGURA 4.4.2.
Rotación en torno a
un eje arbitrario



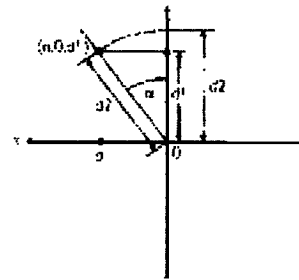
(a)



(b)



(c)



(d)

$$\text{sen}\Phi = b/d1.$$

$$\text{cos}\Phi = c/d1.$$

Al substituir estos valores en la matriz de rotación sobre el eje x, se tiene

$$R_x(\Phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d1 & b/d1 & 0 \\ 0 & -b/d1 & c/d1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Después de este paso, el eje de rotación se halla en el plano xz y el punto $(0, b, c)$ ha sido transformado en el punto $(0, 0, d1)$ (Figura 4.4.2 c). Dado que la rotación en torno al eje x no cambia los valores de las coordenadas x, el punto (a, b, c) está ahora en posición $(a, 0, d1)$ (Figura 4.4.2c).

El siguiente paso es una rotación alrededor del eje y hasta que punto $(a, 0, d1)$ está en el eje z. Como $(a, 0, d1)$ está en el plano xz , se puede visualizar como una rotación alrededor del origen, con el eje y rebasando el papel (Fig.4.4.2d). La rotación de un ángulo α en el sentido de las manecillas del reloj se encarga de esta tarea. Dado que la dirección antihorario es positiva, se debe hacer una rotación de $-\alpha$. Al igual que antes, sólo se necesita calcular el seno y coseno de α . A partir de la figura 4.4.d,

se sabe que la hipotenusa es

$$d2 = \sqrt{a^2 + d1^2} = \sqrt{a^2 + b^2 + c^2}.$$

Así,

$$\text{sen } \alpha = a/d2$$

$$\text{cos } \alpha = d1/d2.$$

y

$$\text{sen } (-\alpha) = -a/d2$$

$$\text{cos } (-\alpha) = d1/d2$$

Al substituir estos valores en la matriz de rotación y , se obtiene

$$Ry(-\alpha) = \begin{bmatrix} d1/d2 & 0 & a/d2 & 0 \\ 0 & 1 & 0 & 0 \\ -a/d2 & 0 & d1/d2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

El paso 4 requiere sólo la matriz de rotación $Rz(\theta)$. Los últimos tres pasos las matrices inversas de las primeras tres transformaciones. Las inversas de las matrices de rotación $Rx(\phi)$ y $Ry(-\alpha)$ son $Rx(-\phi)$ y $Ry(\alpha)$ respectivamente. La inversa de la matriz de traslación $Tr(-x1, -y1, -z1)$ es $Tr(x1, y1, z1)$. La transformación compuesta que representa los siete pasos está expresado en el producto:

$$Tr(-x1, -y1, -z1) * Rx(\phi) Ry(-\alpha) * Rz(\theta) Ry(\alpha) Rx(-\phi) Tr(x1, y1, z1)$$

4.5 REFLEXIONES

Esta clase de transformaciones produce reflexiones de coordenadas en torno a un plano de reflexión especificado. Las matrices de reflexión tridimensionales se forman análogamente a aquellas de dos dimensiones.

Una reflexión **que** convierte las especificaciones coordenadas de un sistema de coordenadas del lado derecho en un sistema del lado izquierdo (o **vice versa**). Esta transformación cambia el signo de las coordenadas **z**, dejando inalterados los valores de las coordenadas **x** y **y**. La representación matricial de esta reflexión de puntos relativa al plano **xy** es:

$$RF_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Las matrices de transformación para invertir valores de las coordenadas **x** y **y** se definen en forma similar, como reflexiones relativas al plano **yz** y al plano **xz** respectivamente.

4.6 TRANSFORMACION DE SISTEMAS DE COORDENADAS

Examinemos ahora la situación de la figura (4.6.1a), donde el segundo sistema de coordenadas, denotado por **x', y'**, se halla en un ángulo de 45 grados horario del

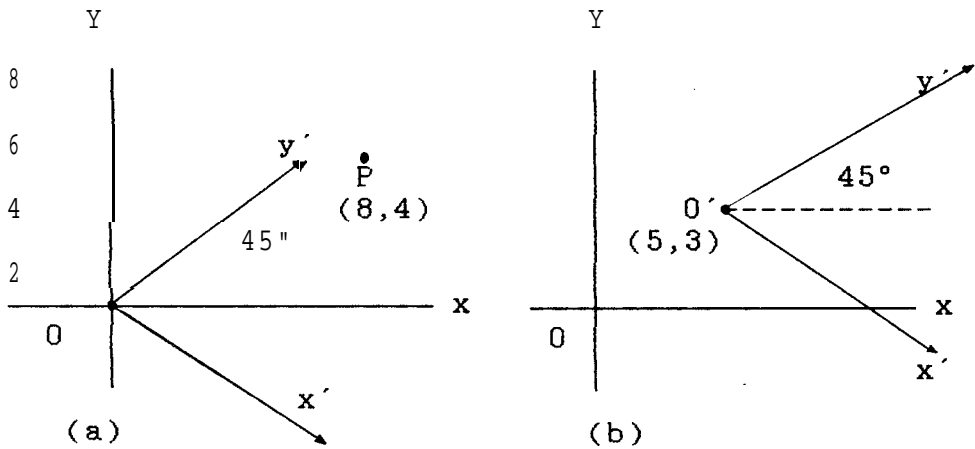


Figura 4.6.1 Segundo sistema de coordenadas (a) rotado y (b) trasladado y rotado con respecto al sistema de coordenadas inicial.

sistema de coordenadas x, y . De acuerdo con esta regla, la transformación bidimensional requerida que permite el segundo sistema de coordenadas, x', y' , al primer sistema de coordenadas, x, y , es una **rotación** de 45 grados en sentido antihorario.

$$\text{Rotar}(\pi/4) = \begin{bmatrix} \cos \pi/4 & \text{sen } \pi/4 & 0 \\ -\text{sen } \pi/4 & \cos \pi/4 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

0

$$\text{Rotar}(\pi/4) = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Por lo tanto el punto $P=(8,4)$ del sistema de coordenadas x,y se representa como $P'=(2\sqrt{2}, 6\sqrt{2})$ en el sistema x',y' , donde

$$[2\sqrt{2}, 6\sqrt{2}, 1] = [8, 4, 1] \times \text{Rotar}(\pi/4)$$

Si el sistema de coordenadas rotado x',y' tiene su origen en $(5,3)$ con respecto al sistema de coordenadas x,y (Fig. 4.6.1b) la transformación de coordenadas se obtiene premultiplicando la matriz de rotación anterior por la matriz de traslación bidimensional.

$$\text{Trasladar } (-5, -3) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5 & -3 & 1 \end{bmatrix}$$

Esta técnica funciona correctamente siempre que el ángulo de rotación esté dado con facilidad. Sin embargo, en la mayoría de los casos los ángulos son desconocidos y su cálculo requiere de un esfuerzo considerable. En particular, se examinarán situaciones en las cuales el nuevo sistema de coordenadas está definido por su origen y las direcciones de sus tres ejes perpendiculares con respecto al sistema de coordenadas estándar. Como se dijo en la sección anterior, estas direcciones son vectores que recorren los tres ejes.

CAPITULO V

VISTA TRIDIMENSIONAL

5.1 PROYECCIONES

En la **graficación** por computador, un objeto tridimensional es presentado en una pantalla bidimensional. La proyección es una transformación que convierte una representación tridimensional en bidimensional.

PROYECCION PARALELA

El método más sencillo de proyección de **imagenes** tridimensionales en dos dimensiones consiste en descartar una de las coordenadas. Se conoce a este método como **proyección** paralela u ortogonal. Aquí se considera sólo el caso de la **eliminación** de la coordenada z , lo cual proyecta el sistema de coordenadas x, y, z sobre el plano x, y . (Fig. 5.1.1).

La proyección de un punto $Q: (x, y, z)$ dentro del cubo es el punto $Q': (px, py)$ en el plano x, y donde una línea pasa por Q y que es paralela al eje z **intersecta** el plano x, z . Estas líneas paralelas se

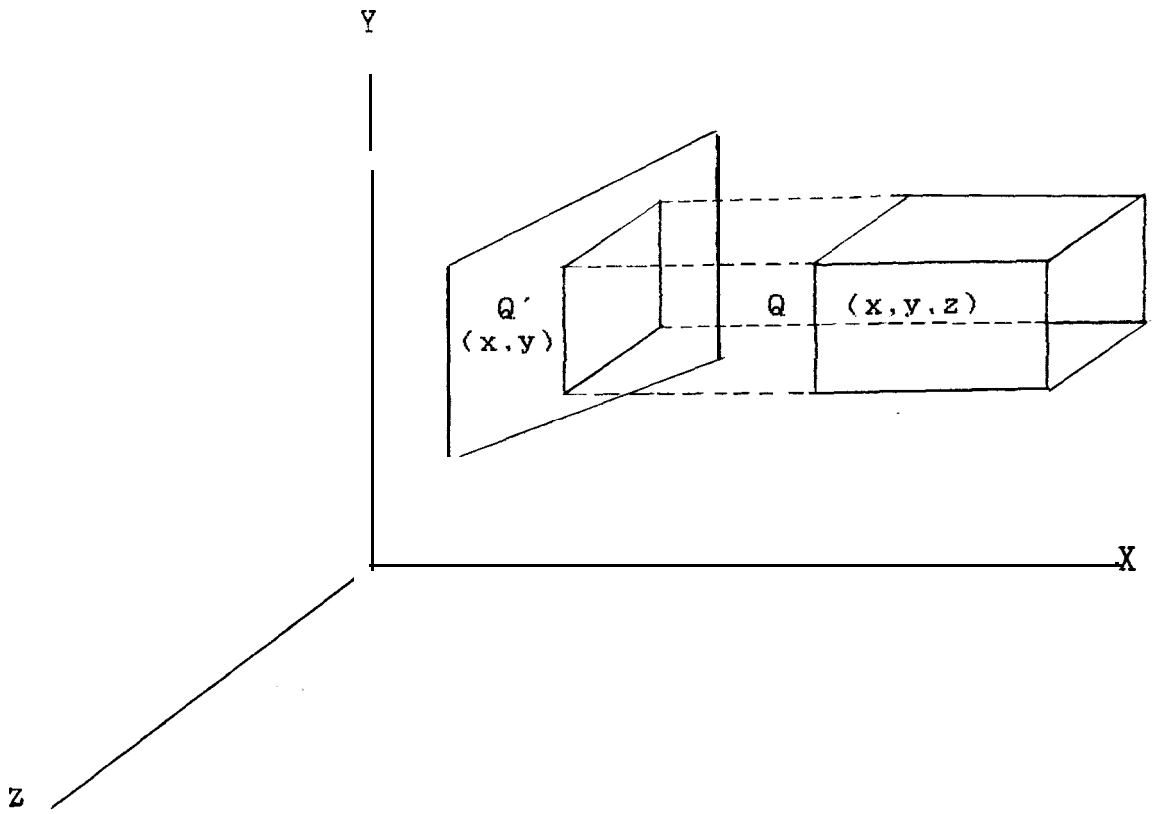


Figura 5.1.1 Proyección en paralelo

conocen como **proyectores**.

La matemática que describe lo anterior es muy simple:

$$px = x$$

$$py = Y$$

Las proyecciones paralelas tienen la propiedad de que las líneas rectas permanecen como **tales**. Por ello, sólo se necesita proyectar los extremos de una recta en tres dimensiones, para luego dibujar una línea bidimensional entre estos puntos. Esto mejora considerablemente la velocidad del proceso de transformación.

La principal desventaja de la proyección paralela es la carencia de información en torno a la profundidad. El observador de la proyección de un cubo especialmente un cubo cuadrado no tiene idea de que el objeto original en tres dimensiones es el cubo. Las imágenes bidimensionales realistas deben proporcionar al observador una representación más exacta de la imagen tridimensional.

Una **proyección oblicua** se obtiene proyectando puntos a lo largo de líneas paralelas que no son perpendiculares al plano de proyección. La figura 5.1.2 muestra una proyección oblicua de un punto (x, y, z) por una línea de proyección a la posición

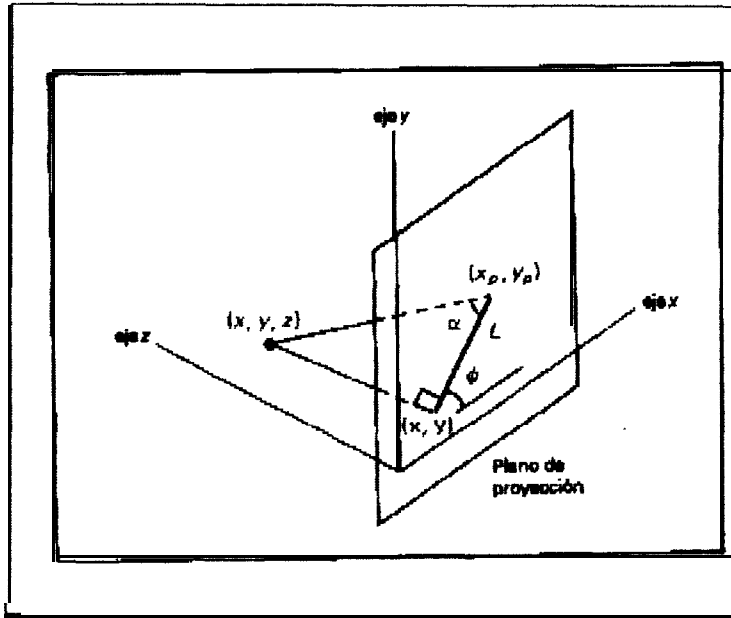


Figura 5.1.2 Proyección oblicua del punto (x, y, z) a la posición (x_p, y_p) sobre el plano de proyección.

(x_p, y_p) . Las coordenadas de **proyección** ortogonal en el plano son (x, y) . La línea de proyección oblicua forma un ángulo α con la línea sobre el plano de proyección que une (x_p, y_p) y (x, y) . Esta línea, de longitud L , está en un ángulo ϕ con la dirección horizontal en el plano de proyección. Podemos expresar las coordenadas de proyección en **términos** de x, y, L y ϕ :

$$x_p = x + L \cos \phi \quad (5.2)$$

$$y_p = y + L \sin \phi$$

Una dirección de proyección puede definirse seleccionando valores para los ángulos ϕ y α . Alternativas comunes del ángulo ϕ son 30° y 45° , los cuales despliegan una vista combinada del frente, lado y parte superior (o bien del frente, lado y parte inferior) de un objeto. La longitud L es función de la coordenada z y podemos evaluar este parámetro a partir de las relaciones

$$\tan \alpha = \frac{z}{L} = \frac{1}{Ll} \quad (5.3)$$

donde Ll es la longitud de la línea de proyección de (x, y) a (x_p, y_p) cuando $z=1$. De la ecuación 5.3, se tiene

$$L = z Ll \quad (5.4)$$

y las ecuaciones de **proyección oblicua** (5.2) pueden escribirse como

$$P \text{ paralelo} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ L1 \cos\phi & L1 \sin\phi & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.6)$$



Una **proyección ortogonal** se obtiene cuando $L1 = 0$ (que ocurre en un ángulo de **proyección** α de 90°). Las **proyecciones oblicuas** se generan con valores distintos de cero para $L1$. La **matriz de proyección** 12.6 tiene una estructura similar a la de una **matriz de corte del eje z**. De hecho, el efecto de esta **matriz de proyección** es el de **cortar planos de z constante** y **proyectarlos sobre el plano de visión**. Los valores de las **coordenadas x y y** dentro de cada **plano de z** del **plano de** manera que los **ángulos, distancias y líneas paralelas** del **plano** se **proyecten con exactitud**.

Dos **ángulos** que se usan comúnmente en las **proyecciones oblicuas** son aquellos para los cuales $\tan \alpha = 1$ y $\tan \alpha = 2$. En el primer caso, $\alpha = 45^\circ$ y las **vistas que se obtienen** se denominan **proyecciones caballera**. Todas las **líneas perpendiculares al plano de proyección** se **proyectan sin cambio de longitud**. Cuando el **ángulo de proyección** se escoge tal que $\tan \alpha = 2$, la **vista resultante** se llama **proyección de**

gabinete, este ángulo (63.4°) ocasiona que las líneas perpendiculares a la superficie de visión se proyecten en una mitad de su longitud.

5.2 TRANSFORMACION DE LA VISION

ESPECIFICACION DEL PLANO DE VISION

El plano de visión es la superficie sobre la cual se proyecta la vista de un objeto. El plano de visión se establece definiendo un sistema de coordenadas de visión, como se muestra en la figura 5.2.1. Las posiciones coordenadas mundiales se volverán a definir y se expresarán relativas a este sistema de coordenadas.

Para establecer las coordenadas de visión, el usuario elige una posición en coordenadas mundiales para que sirva como punto de referencia de la visión. Este será el origen del **sistema** de coordenadas de visión. La orientación del plano de visión se define especificando el vector *normal del plano de visión*, N . Este vector establece la dirección del eje z positivo del **sistema** coordenado de visión. Un vector vertical V , denominado vector de *vista superior*, se utiliza para definir la dirección del eje y positivo. La figura 5.2.2 ilustra la orientación del **sistema de** coordenadas de visión donde el plano de

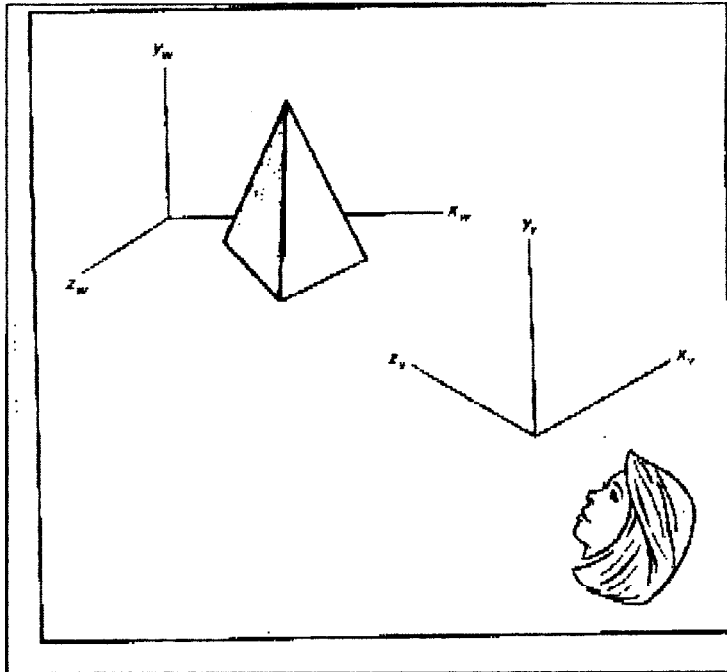


Figura 5.2.1 Sistema de coordenadas de visión. Las descripciones del objeto en coordenadas mundiales se transforman en coordenadas de visión.

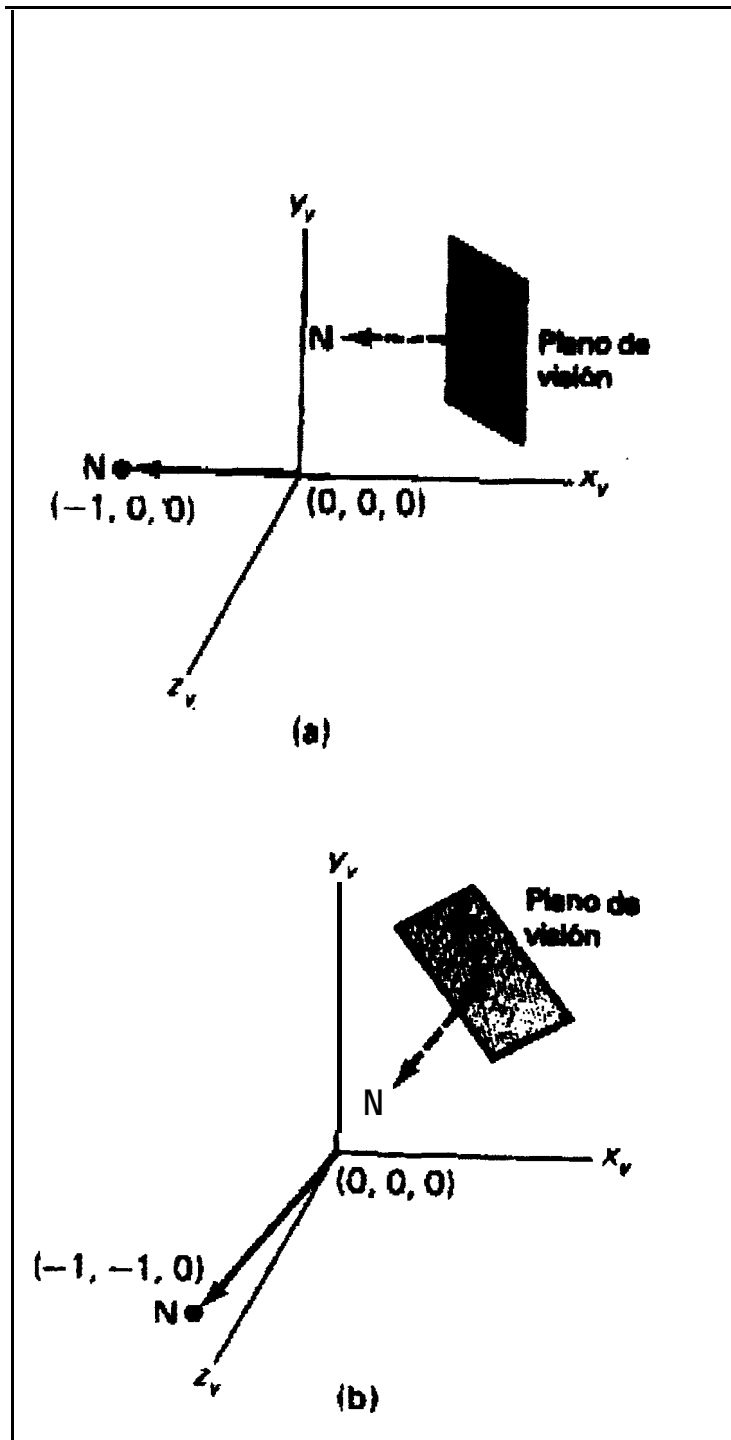


Figura 5.2.2 El punto de referencia de visión y los vectores N y V posicionan y orientan el sistema de coordenadas de visión.

visión es el plano xy .

La normal del plano de visión N puede establecerse especificando una posición coordinada relativa al origen coordinado mundial. Esto define la dirección del vector normal **como** la línea que va del origen a la posición coordinada especificada.

Algunas veces se utiliza un tercer vector U para indicar la dirección x del sistema de coordenadas de visión. El **sistema** de visión puede describirse entonces como un sistema uvm y al plano de visión se **le llama** plano UV . Supondremos que la dirección x positiva es la dirección que se muestra en la figura **5.2.3**. La dirección de U y V en esta imagen es consistente con la orientación de los ejes x y y y en el dispositivo de despliegue. Podemos suponer que el plano de visión en este sistema de visión es un dispositivo lógico en el cual se desplegará la imagen.

La matriz **que** representa esta secuencia de transformación puede obtenerse eslabonando las siguientes matrices de transformación:

1. Refléjese en relación con el plano xy , invirtiendo el signo de cada coordenada z . Esto cambia el sistema de coordenadas de visión del

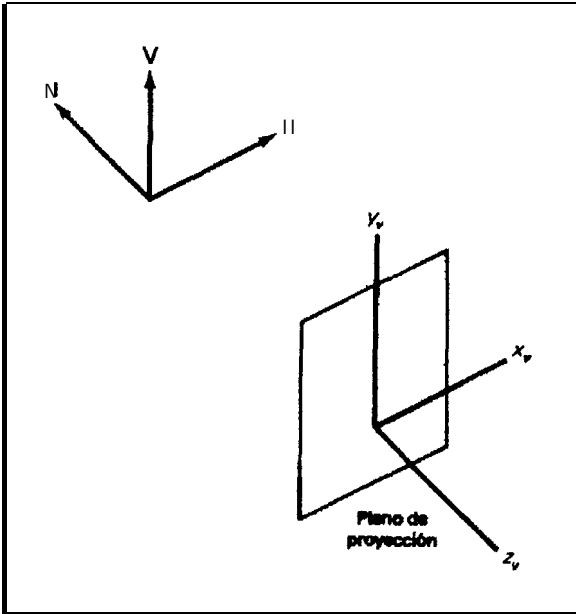


FIGURA 5.2.3
Sistema de visión del lado
derecho definido con los vectores
 U , V y N .



lado izquierdo por un sistema del lado derecho.

2. Trasládese el punto de referencia de **visión** al origen del sistema de coordenadas mundiales
3. Gírese en torno al eje x de las coordenadas mundiales para traer el eje z de las coordenadas de **visión** al plano xz del sistema de coordenadas mundiales.
4. Gírese en torno al eje y de las coordenadas mundiales hasta que se alineen los ejes de ambos sistemas
5. Gírese en torno al eje z de **coordenads** mundiales para alinear los ejes y de coordenadas mundiales de **visión**.

El efecto de cada una de estas transformaciones se muestra en la figura 5.2.4.

VOLUMENES CON VISTA

La ventana de proyección se define por valores mínimos y máximos de x y de y en el plano de visión, como se muestra en la figura 5.2.5. Las coordenadas de visión se utilizan para dar los límites de la ventana, los cuales pueen aparecer en cualquier parte del plano de visión.

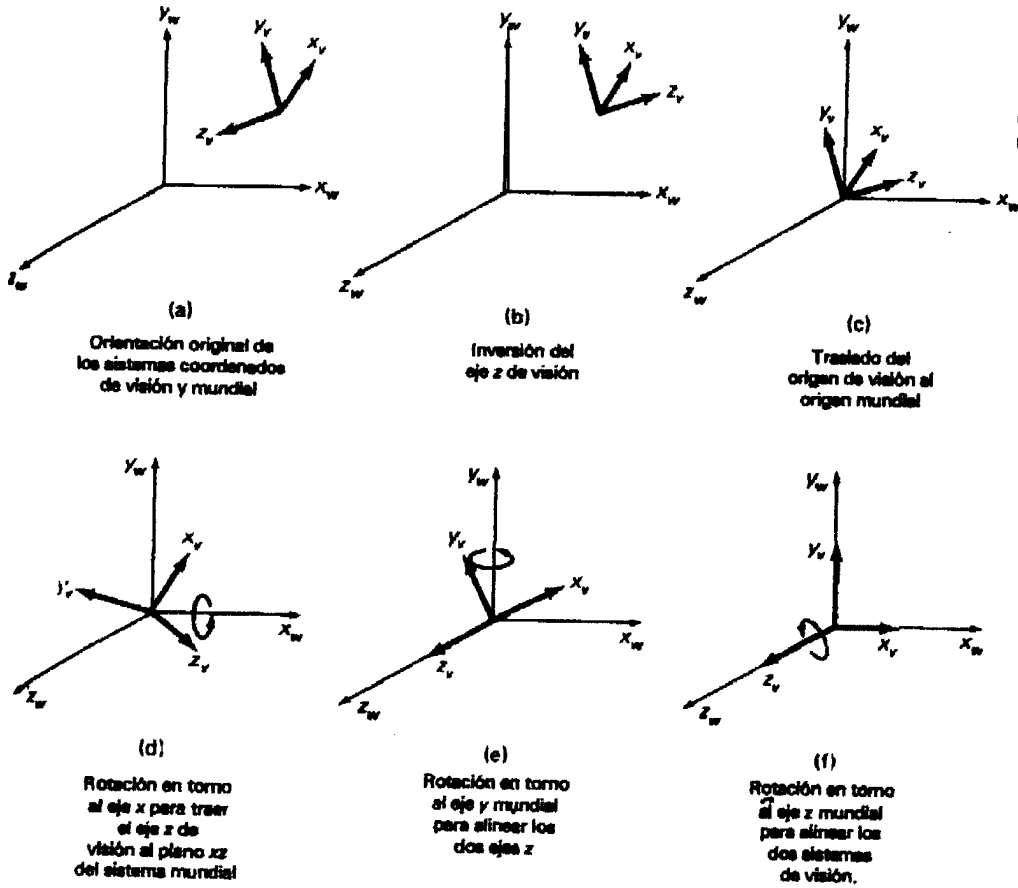


FIGURA 5.2.4
 Secuencia de transformaciones para alinear un sistema de visión con los ejes de coordenadas mundiales.

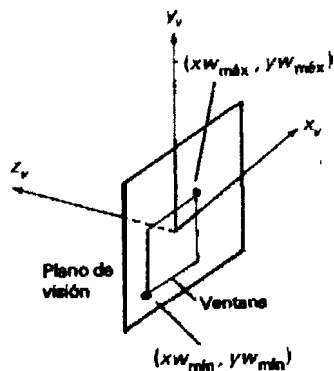
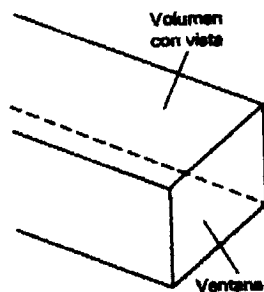


FIGURA 5.2.5
Especificación de ventana sobre el plano de visión, con coordenadas mínimas y máximas dadas en el sistema de referencia de visión.

FIGURA 12-20 .2.6
Volumen cal vista de una proyección en paralelo.

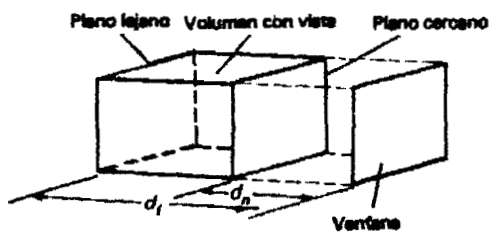


La ventana de proyección se utiliza para definir un **volumen con vista**. Sólo aquellos objetos que estén dentro del volumen con vista se proyectan y despliegan en el tipo de proyección requerido por un usuario. En cualquier caso, cuatro lados del volumen atraviesan las aristas de la ventana. En una proyección en paralelo, estos cuatro lados del volumen con vista forman un paralelepípedo infinito, figura 5.2.6.

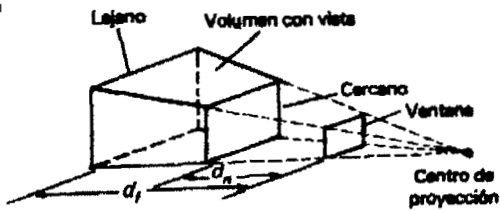
En las proyecciones en paralelo, la dirección de proyección define la orientación del volumen con vista. Dando una posición relativa al origen coordenado de visión, un usuario define un vector que fija la orientación del volumen con vista relativo al plano de visión.

Con frecuencia, se utilizan uno o dos planos adicionales para definir aun más el volumen con vista. Incluso un **plano cercano** y un **plano lejano** produce un volumen con vista finito limitado por seis planos, como se muestra en la figura 5.2.7. Los planos cercano y lejano siempre son paralelos al plano de visión y se especifican por medio de distancias desde el plano de visión en coordenadas de visualización.

estos planos, el usuario puede eliminar partes de la



(a)
Proyección en paralelo



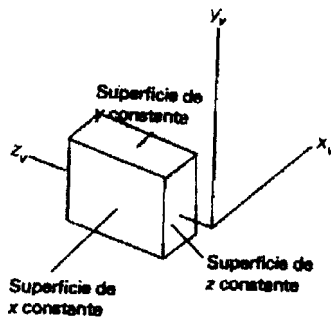
(b)
Proyección en perspectiva

FIGURA 5.2.7
Volúmenes con vista limitados por planos cercanos y lejanos y por planos superior, inferior y lateral. Las distancias a los planos cercanos y lejanos se especifican por d_n y d_l .

de las líneas **con** estos planos es simplemente la coordenada z del plano correspondiente. **Pero** los otros cuatro lados del volumen con vista pueden tener orientaciones espaciales arbitrarias. Para determinar la intersección de una línea con uno de estos lados se requiere obtener la ecuación del plano que contiene el lado del volumen con vista. Sin embargo, esto se vuelve innecesario si convertimos el volumen con vista antes de recortar un paralelepípedo regular.

El recorte contra un paralelepípedo regular es más simple porque cada superficie es ahora perpendicular a uno de los ejes coordenadas. Como se observa en la figura 5.2.8, las partes superior e inferior de este volumen con vista **son** planos de y constante, los lados son planos de x constante y los planos cercano y lejano tienen un valor fijo de z . Todas las líneas que cortan el plano superior del paralelepípedo, por ejemplo, tienen ahora el valor de la coordenada y de ese plano. Además de simplificar la operación de recorte, la conversión **a** un paralelepípedo regular reduce el proceso de proyección **a** una proyección. Primero consideramos la manera de convertir un volumen con vista en un paralelepípedo regular y después analizamos la operación de proyección.

FIGURA 5.2.8
Volumen con vista de un
paralelepípedo regular.



En el caso de una proyección ortogonal en paralelo, el volumen con vista ya es un paralelepípedo rectangular. Para una proyección oblicua en paralelo se corta el volumen con vista para alinear la dirección de proyección con el vector normal al plano de visión, \mathbf{N} . Esta transformación de corte trae los lados del volumen con vista perpendiculares a la superficie de visión, como se observa en la figura 5.2.9.

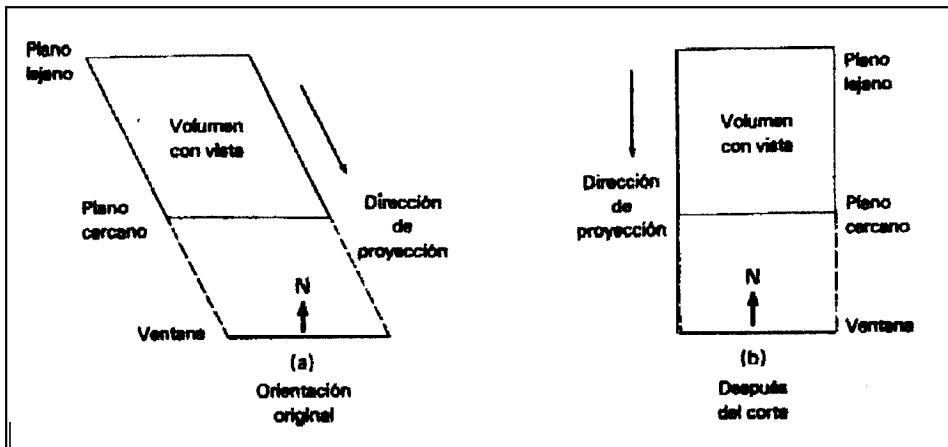


FIGURA 5.2.9
Corte de un volumen con vista
con proyección en paralelo
oblicua en un paralelepípedo
regular (vista superior).

CAE'ITULO VI

SUPRESION DE SUPERFICIES Y LINEAS OCULTAS

6.1 GENERALIDADES

Una consideración importante en la **generación** de escenas realistas es la identificación y supresión de las partes de la imagen definida que no son visibles desde una posición de observación seleccionada. Algunos **métodos** requieren más memoria, en algunos interviene más tiempo de procesamiento y algunos solo se aplican **a** tipos especiales de objetos. El método elegido para una aplicación determinada depende de factores **tales** como la complejidad de la escena, los tipos de objetos que se desplegarán, el equipo de disposición y si se van a generar despliegues animados o bien estáticos.

6.2 CLASIFICACION DE **ALGORITMOS**

Los algoritmos de líneas y superficies ocultas **a** menudo se clasifican según se refieren **a** definiciones de objetos en forma directa o bien con sus imágenes proyectadas. Estos dos métodos se denominan métodos de objeto - espacio y métodos de imagen - espacio,

respectivamente. El primer método compara objetos y partes de objetos unos con otros para determinar qué superficies y líneas, como un todo, deben rotularse como invisibles. En un algoritmo de imagen-espacio. la visibilidad se decide punto por punto en cada **posición** de pixel sobre el plano de proyección. Los algoritmo de línea oculta por lo general emplean métodos de objeto-espacio aunque muchos algoritmos de superficie oculta de imagen-espacio pueden adaptarse **fácilmente a la supresión** de líneas ocultas.

El rendimiento se utiliza para facilitar las comparaciones de profundidad mediante el ordenamiento de las líneas, superficies y objetos individuales de una escena de acuerdo con su distancia desde el plano de **visión**. Los métodos de coherencia se usan para aprovechar las regularidades de una escena. Puede esperarse **que** una línea de rastreo individual contenga intervalos (corridas) de intensidades de pixel constantes y los modelos de **líneas** de rastreo a menudo cambian poco de una línea **a** la siguiente. Los marcos o cuadros de **animación** contienen variaciones solamente en la vecindad de objetos en movimiento. Y **a** menudo puede establecerse relaciones constantes entre objetos y superficies de una escena.

6.3 SUPRESION DE LA CARA ANTERIOR

Un método simple de objeto-espacio para identificar

las caras anteriores de objetos se basa en la ecuación de un plano:

$$Ax + By + Cz + D = 0 \quad (6.1)$$

Como se observó anteriormente, cualquier punto (x', y', z') especificado en un sistema de coordenadas del lado derecho está en el "interior" de este plano si cumple la desigualdad

$$Ax + By + Cz + D < 0 \quad (6.2)$$

Si el punto (x', y', z') es la posición de visión, cualquier plano para el cual la desigualdad 6.2 se cumple debe ser una cara anterior. Esto es, es aquella que no podemos observar desde la posición de visión.

Podemos realizar una prueba más simple de cara anterior observando al vector normal \mathbf{a} a un plano que describe la ecuación 6.1. Este vector normal tiene las componentes cartesianas (A, B, C) . En un sistema de visualización del lado derecho con la dirección de visión \mathbf{a} lo largo del eje z negativo (fig. 6.3.1), el vector normal tiene componente C paralela a la dirección de visión si $C < 0$. el vector normal apunta lejos de la posición de visión y el plano debe ser una cara anterior.

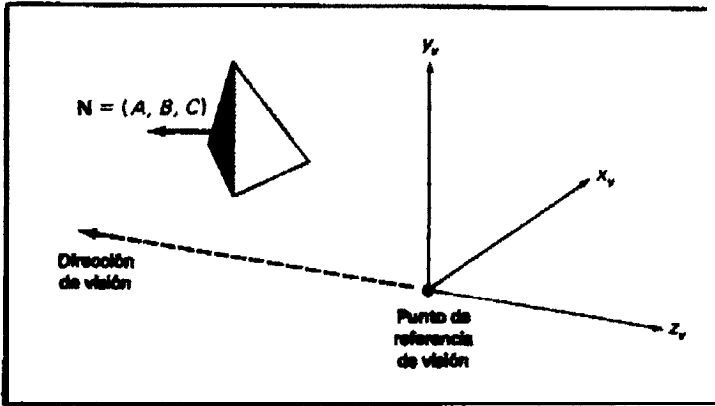


FIGURA 6.3.1
 Plano con parámetro $C < 0$ en un sistema coordenado de visión del lado derecho que se identifica como UN cara anterior cuando la dirección de visión es a lo largo del eje z_v negativo.

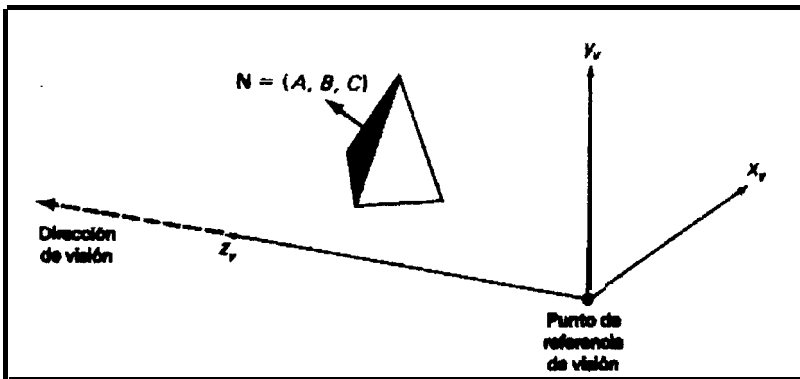


FIGURA 6.3.2
 En un sistema de visión del lado izquierdo con la dirección de visualización a lo largo del eje z_v positivo, una cara anterior es aquella con el parámetro plano $C > 0$.

Pueden aplicarse **métodos** similares en paquetes que utilizan un sistema de visión del lado izquierdo. En estos paquetes, los parámetros de planos A, B, C y D pueden calcularse a partir de coordenadas de **vértice** especificada en un sentido igual al del reloj (en vez de en el sentido contrario al del reloj que se usa en un sistema del lado derecho). La desigualdad 6.2 permanece entonces como una prueba válida de puntos interiores. Asimismo las caras anteriores tienen vectores normales que apuntan lejos de la **posición** de visión y se identifica por $C > 0$ cuando la **dirección** de visión se extiende a lo **alargo** del eje z positivo (figura 6.3.2). En nuestra aplicación utilizamos el sistema de visión de lado izquierdo.

Examinando el **parámetro** C en los diferentes planos que definen un objeto, podemos identificar de inmediato todas las caras anteriores. Para un polihedro convexo, como la pirámide de la figura 6.3.1, esta prueba identifica todas las superficies ocultas del objeto, ya **que** cada superficie es completamente visible o bien completamente oculta. Asimismo, **quizá** haya que determinar si algunos objetos son parcial o completamente oscurecidos **por** otros objetos. En general, puede esperarse que la supresión de la cara anterior elimine cerca de la mitad de las superficies de una escena sin mas

pruebas de visibilidad.

6.4 **METODO** DEL BUFFBR CON PROFUNDIDAD

Un **mètodo** de imagen-espacio que se utiliza **comùnmente** para eliminar superficies ocultas en el **mètodo** buffer con profundidad, tambièm conocido como **mètodo** del buffer **z**. **Bàsicamente** este algoritmo verifica la visibilidad de superficies un punto **a** la vez. En cada **posiciòn** del pixel (x,y) sobre el plano de visiòn, la superficie con menor coordenada **z** en la **posiciòn** es visible. La figura 6.4.1 muestra tres superficies en varias profundidades con respecto **a** la **posiciòn** (x,y) en un sistema de **visiòn** del lado izquierdo. La superficie **S1** tiene el menor valor de **z** en esta **posiciòn**, de manera que se salva su valor de intensidad en (x,y) .

Se requieren dos **àreas** diferentes para implantar este **mètodo**. Un buffer con profundidad se utiliza para almacenar valores de **z** para cada **posiciòn** (x,y) **a** medida que se comparan las superficies y el buffer de **renovaciòn** almacena los valores de intensidad de cada **posiciòn**.

Este **mètodo** puede implantarse adecuadamente en coordenadas normalizadas, con valores de profundidad que varían de 0 **a** 1. Suponiendo que es trazado un

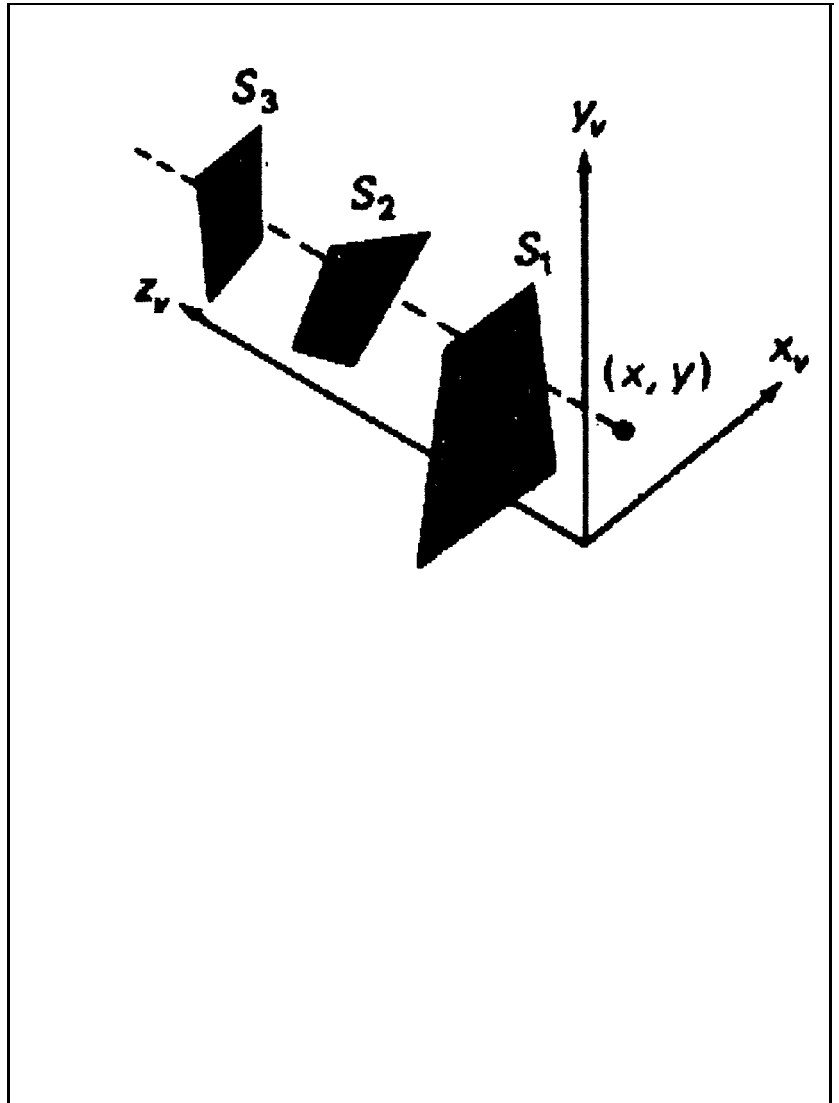


Figura 6.4.1 En la posición (\mathbf{x}, \mathbf{y}) , la superficie S_1 tiene el menor valor de profundidad y de esta manera es visible en esa posición.

volumen con **proyección** en un volumen con vista de un paralelepípedo normalizado, la planimetría de cada superficie situada sobre el plano de visión es una **proyección** ortogonal. La profundidad en los puntos situados sobre superficie de un polígono se calcula a partir de la **ecuación** del plano. Inicialmente, todas las posiciones del buffer con profundidad se hacen igual a 1 (profundidad **máxima**) y el buffer de renovación se inicializa en la intensidad del fondo. Cada superficie enlistada en las tablas de polígonos se procesa **después**, una línea de rastreo a la vez, calculando la profundidad o el valor de **z**, en cada **posición** (x,y) . El valor de **z** calculado se compara con el valor que se almaceno previamente en el buffer con profundidad en esa **posición**. Si el valor de **z** calculado es menor que el valor almacenado en el buffer con profundidad, el nuevo valor de **z** se almacena y la intensidad de la superficie en esa **posición** se coloca a la misma localidad del buffer de renovación.

Podemos resumir las etapas de un algoritmo de buffer con profundidad como sigue:

- 1.- Inicialícese el buffer con profundidad y el de renovación de manera que para todas las posiciones de coordenadas (x,y) , $depth(x,y)=1$ y

refresh(x,y) = background.

2.- Para cada posición de cada superficie, compárese los valores de la profundidad con los valores previamente almacenados en el buffer con profundidad para determinar la visibilidad.

a.- Calcúlese valores de **z** para cada posición **(x,y)** de la superficie.

b.- Si $z < depth(x,y)$ entonces hágase $depth(x,y)=i$, donde *i* es el valor de la intensidad en la superficie en la posición **(x,y)**.

En el último paso, si **z** no es menor que el valor del buffer con profundidad de esa posición, el punto no es visible. Cuando se haya realizado este proceso en todas las superficies, el buffer con profundidad contiene valores **z** para las superficies visibles y el buffer de renovación contiene sólo los valores de intensidad visible.

Los valores de profundidad de una posición **(x,y)** se calculan a partir de la ecuación del plano de la superficie:

$$z = \frac{-Ax - By - D}{c}$$

Para cualquier línea de rastreo (fig. 6.4.2) las coordenadas **x** que atraviesan la línea difieren en 1 y



BIBLIOTECA

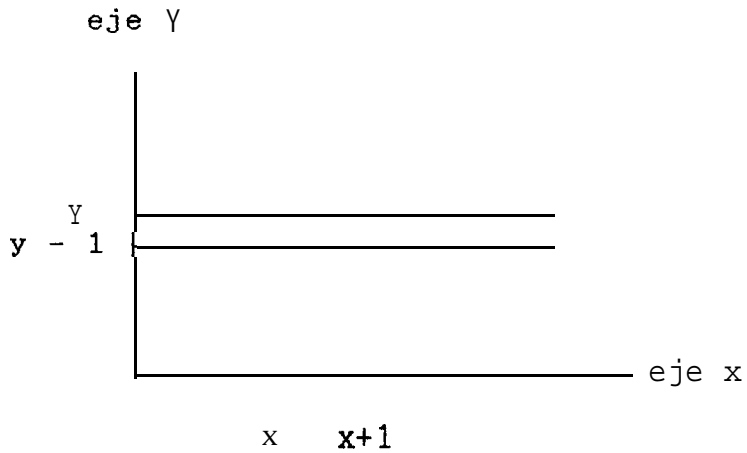


Figura 6.4.2. De la posición (x, y) en una línea de rastreo, la siguiente posición que cruza la línea tiene las coordenadas $(x + 1, y)$ y la posición que está inmediatamente debajo de la siguiente línea tiene las coordenadas $(x, y - 1)$.

los valores de y entre las líneas difieren en 1. Si la profundidad de la posición (x, y) se ha determinado como z , la profundidad z' de la siguiente posición (x, y) a lo largo de la línea de rastreo se obtiene a partir de la ecuación 6.3 como:

$$z' = \frac{-A(x + 1) - By - D}{D}$$

o bien

$$z' = z - A/C \quad (6.4)$$

La razón de A/C es constante en cada superficie, de manera que los valores de profundidad sucesivos a

travès de una línea de rastreo se obtienen **a** partir de los valores anteriores con **sòlo** una resta.

El mètodo del buffer con profundidad es **fàcil** de implantar y no requiere el ordenamiento de las superficies de una escena. Pero si requiere disponer de un segundo buffer ademas del buffer de **renovaciòn**. Por ejemplo, un sistema con una **resoluciòn** de 1024 por 1024 requerirìa mas de un **millòn** de posiciones en el buffer con profundidad, con cada **posiciòn** que contiene los bits suficientes para representar el numero de incrementos de la coordenadas **z** que se necesitan. Una manera de reducir los requisitos de almacenamiento consiste en procesar un **secciòn** de la escena **a** la vez, utilizando un buffer con profundidad de menor tamaño. Despuès de que se procesa cada **secciòn** de la vista, el buffer se vuelve **a** utilizar en la siguiente **secciòn**.

6.5 **METODO** DE LA LINEA DE RASTREO

Este **mètodo** de imagen-espacio para eliminar superficies ocultas es una extensiòn del algoritmo de la línea de rastreo para llenar interiores de polìgonos. En vez llenar solo una superficie, ahora se trabaja con multiples superficies. Conforme se procesa cada línea de rastreo, todas las superficies poligonales que cortan esa línea se examinan **a** fin de

determinar cuales son visibles. En cada posicion situada a lo largo de una linea de rastreo, se hacen **cálculos** de profundidad en cada superficie para determinar cual es la **màs** prbxima al plano de **visiòn**. Cuando se ha determinado la superficie visible, el valor de intensidad de esa **posiciòn** se mete en el buffer de **renovaciòn**. La **presentaciòn** de las superficies poligonales de una escena tridimensional puede formarse para incluir una tabla de aristas y una de polìgonos. La tabla de aristas contiene extremos coordenados de cada linea de escena, la pendiente inversa de cada **lìnea** y apuntadores en la tabla de polìgonos para identificar las superficies limitadas **por** cada linea. La tabla de polìgonos contiene coeficientes de la **ecuaciòn** del plano para cada superficie, **informaciòn** de intensidad de las superficies y posibles apuntadores en la tabla de aristas. Para facilitar la b̀squeda de superficies **que** atraviesan una linea de rastreo dada, podemos preparar una lista activa de aristas **a** partir de la **informaciòn** contenida en la tabla de aristas. Esta lista activa contendra solamente aristas **que** atraviesen la linea de rastreo regular, ordenada en el sentido de x creciente. Ademàs, se define una se˜al para cada superficie que se hace **on** u **off** para indicar si una posicion **a** lo largo de una linea de rastreo **està** dentro o fuera de la superficie. Las

lineas de rastreo se procesan de izquierda **a** derecha. En la frontera de **màs a** la izquierda de la superficie, la señal de la superficie se activa; **y** en la frontera de **màs a** la derecha, la señal se desactiva.

6.6 **ELIMINACION** DE LINEAS OCULTAS

Cuando **sòlo** se va **a** desplegar el perfil de un objeto, los **mètodos** de líneas ocultas se utilizan para eliminar las aristas de objetos que son oscurecidos por superficies **màs** cercanas del plano de **visiòn**. Los **mètodos** para suprimir líneas ocultas pueden desarrollarse considerando en forma directa las aristas del objeto o bien adaptando **mètodos** de **superfices** ocultas. Un **mètodo** directo para eliminar líneas ocultas consiste en comparar cada **lìnea** con cada superficie de una escena. El proceso implicado aquí es similar **a** recortar líneas contra formas de ventanas arbitrarias, excepto que ahora se desean cortar las partes ocultas por superficies. Para cada línea, los valores de profundidad se comparan con las superficies para determinar que secciones de **lìneas** no son visibles. Podemos utilizar **mètodos** de coherencia para identificar segmentos de línea oculto sin probar en realidad cada **posiciòn** coordenada. Si ambas intersecciones de la línea con la **proyecciòn** de una frontera de superficie tienen mayor profundidad

que la superficie en esos puntos, el segmento de línea situado entre las intersecciones está completamente oculto. Cuando la línea tiene mayor profundidad en una **intersección** de frontera y menor profundidad que la superficie en la otra **intersección** de la frontera, la línea debe penetrar el interior de la superficie. En este caso, calculamos el punto de **intersección** de la línea con la superficie utilizando la **ecuación** del plano y desplegando sólo las secciones visibles. Algunos **métodos** de superficie oculta se adaptan **fácilmente** a la **supresión** de líneas ocultas. Mediante un **método** de cara anterior, podríamos identificar todas las superficies anteriores de un objeto y desplegar solamente las fronteras de las superficies visibles. Mediante el procesamiento de las superficies del frente a la parte anterior, las líneas ocultas son borradas por las superficies más cercana.

CAPITULO VII

CREACION DE LIBRERIAS

7.1 OBJETIVOS DE LA PLANEACION

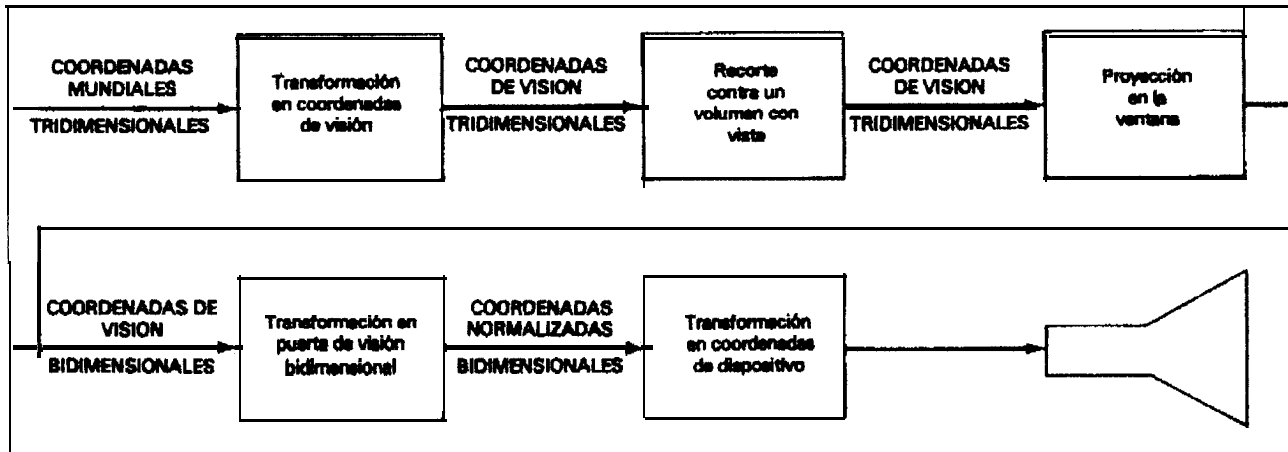
El objetivo es contar con un sistema gráfico accesible que propicie la creatividad del usuario y la facilidad de uso, considerando también la eficiencia del sistema de **presentación** del computador la facilidad en la **programación**.

El estilo de **programación** sigue los **métodos** de la programación estructurada, lo cual exige un enfoque descendente que divide al programa en módulos de unidades independientes unos y otros como unidades combinadas.

La creación de estas librerías proporciona una herramienta útil para futuras aplicaciones, **ya que** pueden ser modificadas, corregidas, y actualizadas de manera rápida y fácil. El paquete de gráficas podría construirse para que produzca diseños de ingeniería, proyectos de dibujo mecánico.

Para una visión tridimensional se necesitan las

FIGURA 7.1
Operaciones lógicas en una visión tridimensional.



siguientes operaciones lógicas, como se presenta en la figura 7.1.

La entrada del sistema es a través de un archivo tipo texto el cual contiene las coordenadas de los vértices, aristas y polígonos de las figuras tridimensionales.

7.2 METODOLOGIA PARA LA CLASIFICACION DE APLICACIONES

DEFINICION DE LA ESTRUCTURA DE DATOS

Para cada figura, se crean tres listas:

- Tabla de **vertices** v
- Tabla de aristas e
- Tabla de poligonos s

Los valores coordenados de cada vértice del objeto se almacenan en la tabla de vértices. La tabla de aristas **enlista** los vértices extremos que definen a cada arista. Cada polígono se define en la tabla de polígonos como una lista de aristas componentes, como se ilustra en la figura 7.2.1.

Cada celda es un registro de tres campos:

dato :**tipo** real

ptr1,ptr2 :**puntero**

TABLA DE
VERTICES (V)

TABLA DE
ARISTAS (E)

TABLA DE
POLIGONOS (S)

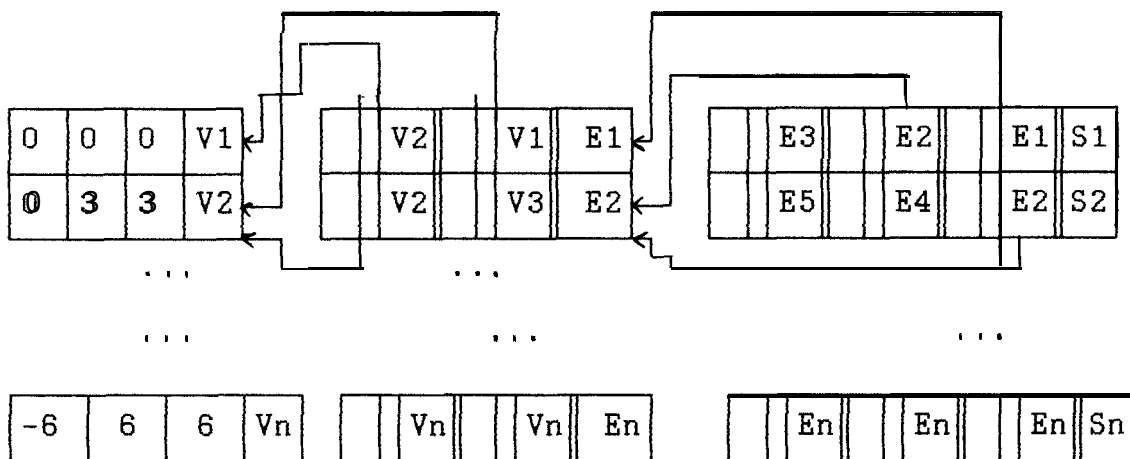


FIGURA 7.2.1 Estructura de los datos

Este listado de los datos **geométricos** en tres tablas, como se muestra, ofrece una referencia adecuada de las componentes (vértices, aristas y polígonos) de un objeto.

El objeto puede desplegarse eficazmente mediante el uso de datos de la tabla de aristas para trazar las líneas componentes.

Los elementos de las listas se enlazan a través del puntero **ptr2**. La tabla de polígonos incluye apuntadores en la tabla de aristas (a través de **ptr1**) de modo que pudieran identificarse aristas comunes entre polígonos y no ser trazadas nuevamente, de esta misma forma se simplifican también los métodos de sombreados de superficies. La tabla de aristas igualmente incluye apuntadores (a través de **ptr1**) hacia la tabla de vértices.

Como **vamos a** tener en un gráfico varias figuras, entonces **incluimos** un vector de punteros que contiene la dirección de las tablas de polígonos correspondientes a cada figura, como se muestra en la figura 7.2.2. Así cada figura se configura en forma independiente.

Es muy importante que el archivo de datos contenga las coordenadas correctas, en caso de que se grafique

TABLA DE POLIGONOS (S)

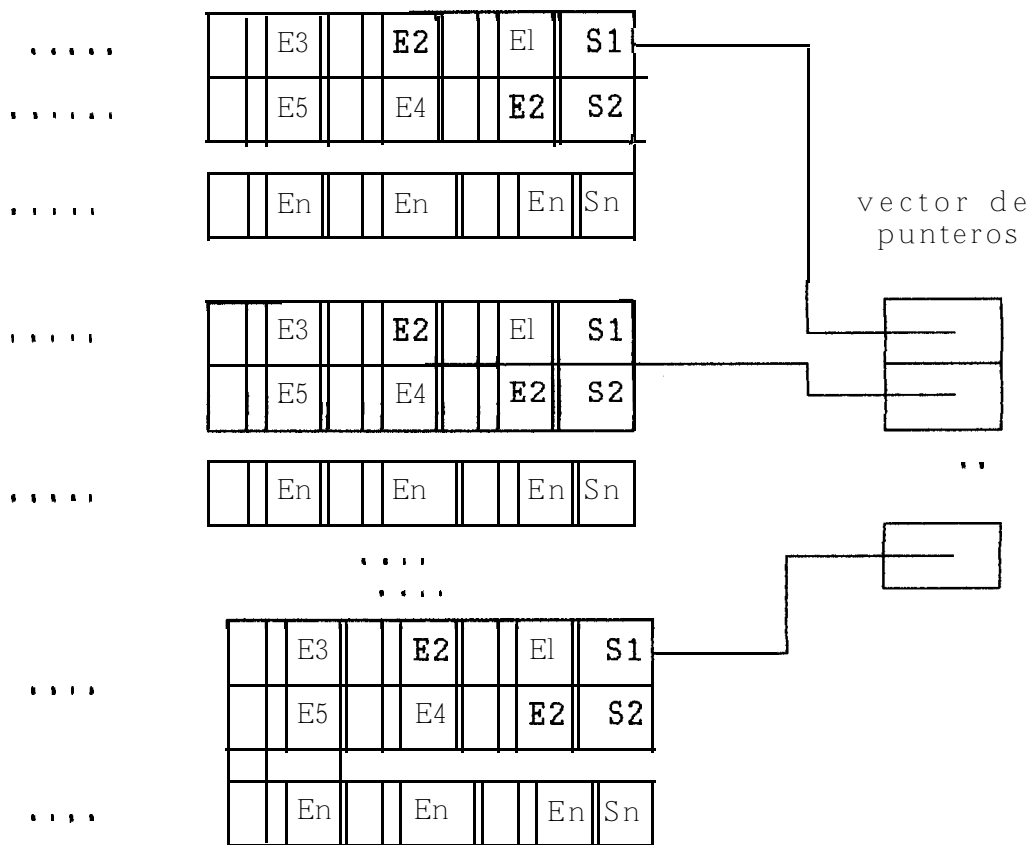


FIGURA 7.2.2

un diseño que no está incluido en el paquete de gráficas tridimensionales.

DESCRIPCION DE LAS LIBRERIAS GRAFICAS

El sistema de **graficación** contiene cinco librerías gráficas:

- Tablas
- Visible
- Dos-d
- **Motion**
- Imagen

UNIDAD TABLAS

Este módulo junto con lista se encargan de la creación de listas que contienen los datos de la figura a representar.

La unidad Tablas se encarga de enlazar las diferentes listas, es decir **a** la tabla polígonos con aristas **y la** tabla de aristas con la de vértices.

Otra operación importante que desempeña esta unidad es la de obtener todos los vértices de las tablas para que sean renovados según la opción de movimiento que se presente.

De la misma forma. luego de terminar el movimiento, los vértices son actualizados, es decir los nuevos valores son devueltos a sus posiciones iniciales en las tablas correspondientes.

UNIDAD VISIBLE

La principal operación que realiza `cara_ocu`, es la de determinar para cada figura cuales son las caras visibles y cuales están ocultas. Esta operación es determinada por medio de la ecuación del plano $AX + BY + CZ + D = 0$. Si el valor de la ecuación evaluando en el punto de visión es mayor que cero es una cara visible, si es menor que cero la cara es oculta. Los valores constantes de A, B, C Y D son calculados aplicando la regla de **Cramer** utilizando tres vértices consecutivos en sentido contrario a las manecillas del reloj para un sistema de coordenadas de lado derecho.

UNIDAD MOTION

Esta unidad contiene tres procedimientos principales:

- Traslación
- Rotación
- Escalación

Los tres procedimientos están implementados para crear

una matriz de **transformación** de movimiento que va a modificar los **vértices** de la figura. Los únicos parámetros de entrada son el *factor* de **escalación** para el procedimiento **Scale**, el *ángulo* de *rotación* para el procedimiento **rotate_x**, **rotate_y**, **rotate_z**, finalmente las distancias para el procedimiento **traslate**.

UNIDAD **DOS_D**

Transforma vértices tridimensionales en vértices bidimensionales aplicando la proyección oblicua en paralelo.

UNIDAD **IMAGEN**

Realiza el trazo de líneas de las figuras, así también el sombreado de las superficies.

7.3 PROGRAMACION DE VISTAS **TRIDIMENSIONALES**

Los parámetros **Xo**, **Yo**, **Zo** especifican el origen (punto de refernecia de visión) del sistema de visualización. Dichos parámetros tienen valores de:

$$Xo = 40$$

$$Yo = 25$$

$$zo = 40$$

El plano de visión es la superficie sobre la cual se

proyecta la vista de un objeto. Podemos considerarlo como la película de una cámara que se ha posicionado y orientado para una toma determinada. El plano de visión se establece definiendo un sistema de coordenadas de visión, como se muestra en la figura 7.3.1.

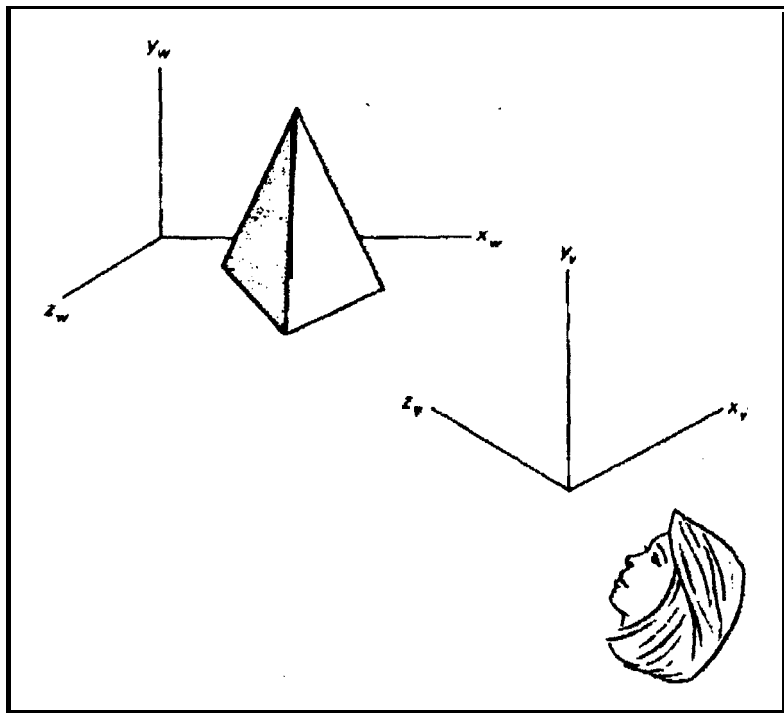
La orientación del plano de visión se define especificando el vector normal del plano de visión, N . Este vector establece la dirección del eje z positivo del sistema coordenado de visión. Un vector vertical V , denominado vector de vista superior, se utiliza para definir la dirección del eje y positivo. La figura 7.3.2 ilustra la orientación del sistema de coordenadas de visión utilizado donde el plano de visión es el plano xy en el origen de las coordenadas de visión. Esto nos permite proyectar hacia el plano $z=0$. Cualquier cambio en las coordenadas de visión no podrá ser accesado por el usuario, los cambios se deberán hacer en el programa.

7.4 EXTENSIONES DEL **MODELO TRIDIMENSIONAL**

Como hemos dicho anteriormente, el sistema de referencia de visión es uno solo, para modificarlo podemos sugerir el uso del siguiente comando:

```
crea_mat_vision(xo,yo,zo,xn,yx,zn,xv,yv,view_mat)
```

FIGURA 7.3.1
Sistema de coordenadas de visión con los ejes y_w , y_v y z_v . Las descripciones del objeto en coordenadas mundiales se transforman en coordenadas de visión.



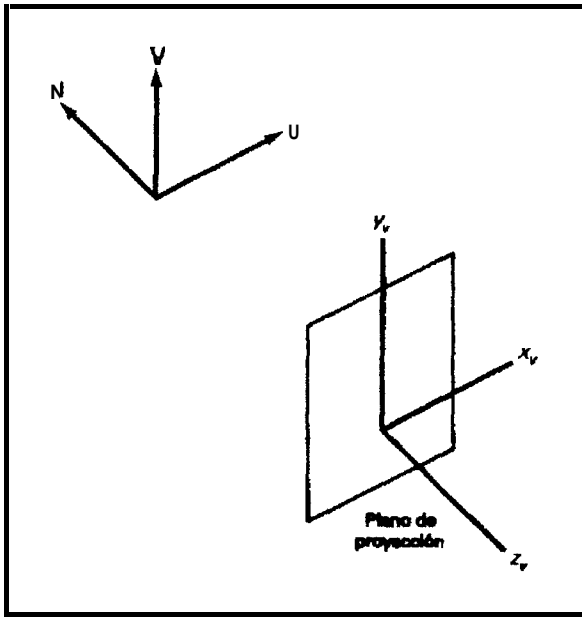


FIGURA 7.3.2
Sistema de visión del lado
derecho definido con los vectores
 U , V y N .

Los parámetros **xo, yo** y **zo** especifican el origen (punto de referencia de visión del sistema de visualización. El eje **z** positivo del sistema de visión se establece en la dirección del vector del origen de las coordenadas mundiales al punto **(xn, yn, zn)**. Y la posición **coordenada(xv, yv, zv)** especifica el vector de vista superior. La proyección de este vector sobre el plano de visión define el eje **y** positivo del sistema de coordenadas de visualización. Estos parámetros se utilizan para construir una **view_mat** para transformar posiciones en coordenadas mundiales en coordenadas de visión.

Para especificar un segundo sistema de coordenadas de visión, el usuario puede redefinir algunos o todos los parámetros coordenados e invocar **crea_mat_vision** con una nueva designación de matriz. En esta forma, puede definir cualquier número de las coordenadas mundiales para visualizar planimetrías coordenadas.

Una vez que se ha definido una matriz para transformar coordenadas mundiales en coordenadas de visión, los parámetros de proyección pueden especificarse con la función :

```
set_mat_vision (indice_de_vision, view_mat,
                tipoqproyeccion, xp, yp, zp, xw_min,
                xw_max, yw_min, yw_max, cerca, lejos,
```



```
xv_min, xv_max, yv_min, yv-max,
zv_min, zv_max)
```

El parámetro `indice_de_vision` sirve como número de identificación del `a` transformación de visión. La matriz de transformación para trazar coordenadas mundiales en coordenadas de visión se especifica en `view_mat` y `a` tipogroyeccion se le asigna un valor de paralelo. La posición `coordenada(xp,yp,zp)` establece la dirección de proyección o bien el centro de proyección, según el valor de entrada del parámetro `tipo-degroyeccion`. Los límites de la ventana de proyección se definen con valores coordenados `xw_min`, `xw_max`, `yw_min`, y `yw_max`, que se especifican relativos al origen coordenado de visión. Los parámetros `cercano` y `lejano` especifican la localidad de los planos `correspoindientes`. Por último, se dan las fronteras de la puerta de visión tridimensional con los parámetros `xv_min`, `xv-max`, `yv_min`, `yv-max`, `zv_min`, `zv_max`, que se especifican en coordenadas normalizadas. Podría incluirse otro parámetro en esta función para permitir a un usuario posicionar el plano de visión a cualquier distancia del origen de visión. Puede definirse; cualquier número de `transfomaciones` de visión con esta función, mediante el uso de diferentes valores de `view_index`.

Un usuario selecciona una transformación de visión

determinada con `set_indice_de_vision` (`vi`). El número del **índice** de visión `vi` identifica el conjunto de parámetros de **transformación** de la visión que se aplicarán a las primitivas de salida que se especifiquen después.

CAPITULO VIII

DISEÑO DE LA INTERFASE DEL USUARIO

8.1 **COMPONENTES** DE LA INTERFAZ DEL USUARIO

Al diseñar el menu para la aplicación de este paquete de librerías gráficas, consideramos solamente las operaciones de **graficación** concernientes **a** los movimientos que realiza la grúa mecánica. Además, se proporciona un **medio** adecuado para que el usuario **accese a** las funciones básicas, como el despliegue de objetos, atributos, tipo de sombreado, y color.

8.2 **MODELO** DEL USUARIO

Las transformaciones de modelado convierten las definiciones en coordenadas maestras en coordenadas mundiales, que después se transmiten al paquete de gráficas para la transformación final de coordenadas de dispositivo. La figura 8.2.1 contiene una secuencia para transformar una definición en coordenadas maestras en coordenadas de dispositivo cuando las rutinas de modelado se sincronizan con el paquete de gráficas. La figura 8.2.2 muestra un

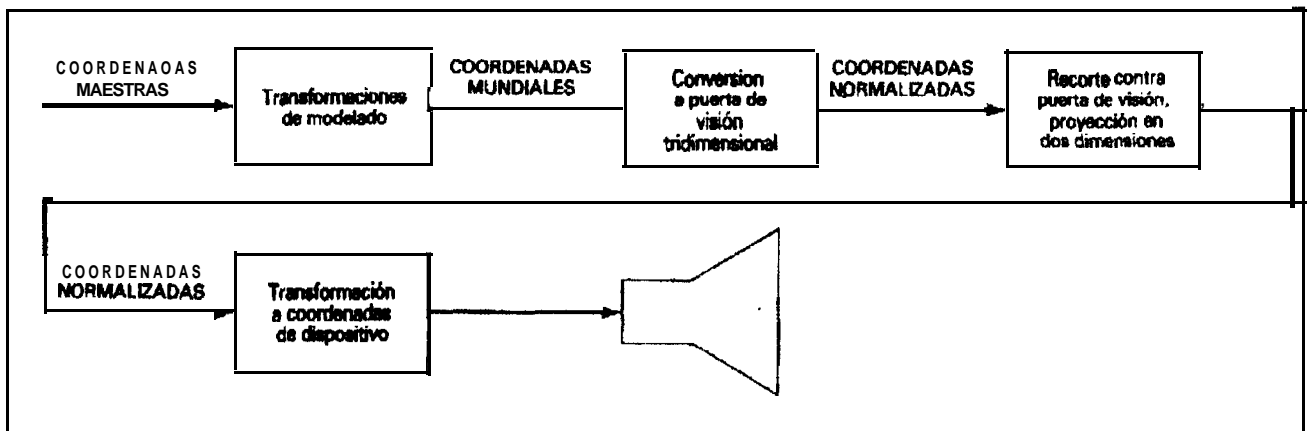


FIGURA 8.2.1
Transformación de coordenadas maestras en coordenadas de dispositivo.

esquema acerca del desarrollo de las operaciones principales de apoyo.

8.3 LENGUAJE DE COMANDO

Debido a la aplicación práctica con el dispositivo externo móvil, el lenguaje de comandos se diseñó en función de un menú simplificado. Sin embargo, cambios en el menú de opciones se pueden planificar eficazmente con el paquete de gráficas.

Los comandos deben diseñarse de modo que el usuario no tenga que aprender nuevos conceptos, así como también el nuevo lenguaje.

FACILIDADES DE AYUDA PARA EL USUARIO

Una de las facilidades que presenta este trabajo, es que el contenido del texto tiene una proyección **tutorial** que ofrece instrucción acerca de la **graficación** por computadoras, así como también se detallan los algoritmos del paquete de gráficas.

TIEMPO DE RESPUESTA

El tiempo que tarda el sistema en responder a la entrada de un usuario depende de la complejidad de la tarea solicitada. Cuando un usuario ingresa una solicitud de procesamiento complicada, puede esperarse alguna demora y este retraso podría

utilizarse para planificar la siguiente fase de la aplicación. Los cálculos matriciales así como **también** los aritméticos son relativamente complejos.

8.4 DISEÑO DEL MENU

El menú inicial contiene los formatos y atributos permitidos para el trazo de líneas, sombreado de superficies y color de los gráficos. El menú de movimientos incluye las siguientes opciones:

1. Traslación
2. Rotacion
- 3 . Salir

Selección 1

Opciones:

Teclas	Acción
<div style="border: 1px solid black; padding: 5px; display: inline-block; margin-bottom: 10px;">↑</div> c 1 1	Sube el peso Baja el peso

Tabla 8.4.1

Selección 2

Opciones:





Teclas	Acción
	Gira derecha
	Gira izquierda
	Abre brazo
	Cierra brazo

Tabla 8.4.2

8.6 **FORMATOS** DB SALIDA

La **información** que se presenta al usuario del paquete de gráficas incluye una combinación de imágenes, menús, mensajes de salida y otras formas de diálogo generadas por el sistema. La pantalla se presenta con el siguiente formato:

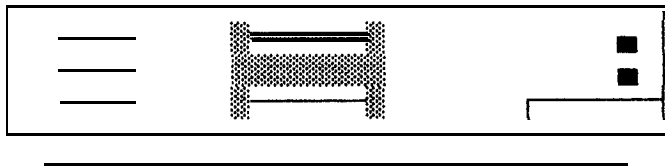
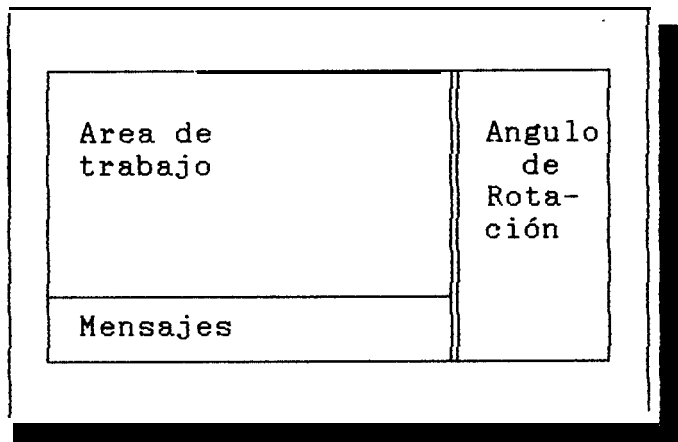


Figura 8.6 Formato de la pantalla

CAPITULO IX

APLICACION DEL **SISTEMA** DE ANIHACION DE GRAFICA

9.1 OBJETIVOS DEL **DISEÑO**

Como ya se mencionó en el capítulo VII el objetivo principal del paquete de gráficas tridimensional es contar con un sistema gráfico accesible, eficiente **que** propicie la creatividad del usuario siendo una herramienta útil para futuras aplicaciones de ingeniería, proyectos de dibujo. Es así que se ha implementado una aplicación del paquete de gráficas hacia un dispositivo externo conectado al puerto paralelo del computador. Este dispositivo permite el control de una grúa-robot, a la cual se le pueden dar órdenes para su posterior ejecución.

El programa principal se sirve de las cinco librerías gráficas descritas en el capítulo VIII, y un archivo **que** contiene todos los vértices, aristas y polígonos que definen la grúa.

En resumen el sistema controlador permitirá:

- Manejar por medio de una computadora personal a un sistema Grúa-Robot.

- Establecer la comunicación PC-GRUA por medio de los puertos paralelos de la PC.
- Permitir a **través** del teclado del computador el control de la GRUA-ROBOT y el control de su respectiva animación **gráfica** en tres dimensiones a través de la pantalla de video.

El sistema envía señales a través del puerto de la FC, a su vez estos **direccionan** a la interfase digital, y es aquí donde una vez procesadas estas señales se accionan los motores de la grúa.

9.2 DESCRIPCION DEL **SISTEMA** QUE CONTROLA EL DISPOSITIVO EXTERNO HOVIL

Como mencionamos anteriormente el programa **que** controla tanto la animación gráfica como el movimiento de la GRUA-ROBOT utiliza las cinco librerías gráficas: Tablas, Visible, Dos-d, Motion, Imagen (Capítulo VII). **Así** también se ha creado un archivo tipo texto que contiene los vértices, aristas y polígonos que definen la forma de la GRUA, esta **información** es procesada para la creación de la estructura de datos de la cual se sirven las librerías gráficas.

La Grua en mención puede ser controlada desde una consola de botoneras y/o desde cualquier computador personal perteneciente a la familia IBM_PC.

Fara este propósito se ha utilizado una interfase digital que permite establecer la comunicación entre las partes mencionadas.

La GRUA consiste de un dispositivo mecánico que es capaz de girar sobre su propio eje, bajar y subir un gancho y contraer o expandir un brazo.

Todas estas funciones se logran gracias al empleo de tres motores DC.

Por lo mencionado anteriormente, los códigos que son transmitidos al puerto son:

PUERTOS 7654 3 2 10	ACCION DE LA GRUA
1 1 1 1 1 1 1 0	BAJA GANCHO
1 1 1 1 1 1 0 1	SUBE GANCHO
1 1 1 1 1 0 1 1	EXPANSION DE BRAZO
1 1 1 1 0 1 1 1	CONTRACCION DE BRAZO
1 1 1 0 1 1 1 1	GIRO CONTRA RELOJ
1 1 0 1 1 1 1 1	GIRO A FAVOR DE RELOJ

El puerto se encera con del **codigo** 1 1 1 1 1 1 1

Cabe señalar que la para el movimiento de traslación en el cual sube y baja el gancho el factor de traslación en la dirección **positiva**(sube gancho) y negativa (baja gancho) del eje y puede ser modificado

de modo fácil, ya que es necesario coordinar el movimiento físico de la Grúa con el movimiento en la pantalla de video. La velocidad con que el gancho sube o baja en la gráfica está afectada directamente **por** la configuración del hardware. Esta misma observación se cumple para el caso del ángulo de rotación *angulo_y* en sentido en contra y a favor del reloj, y el ángulo *angulo_z* al abrir y cerrar el brazo.

A causa de que los motores DC tienen un retardo de tiempo pequeño en el momento inicial en que se le da la orden de movimiento, el cual no se ha simulado en la gráfica, la **coordinación** de movimientos no es en un 100%.

9.3 **MODELO** DEL USUARIO

Como se indicó en el Capítulo VIII, la interfase del usuario está proyectada directamente para la aplicación práctica de la Grúa-Robot. De tal manera **que** las opciones que se presentan únicamente son para:

- Traslación



Subir o Bajar Gancho

- Rotación



Abre
Brazo

Cierra
Brazo

Gira a la
Derecha

Giro a la
Izquierda

Para cancelar cualquier movimiento con la tecla ENTER

Para regresar al menú principal con el punto '.'



BIBLIOTECA

- _____
1. Se ha implementado una aplicación gráfica utilizando la Grúa-Robot y la interfase digital, dicha **apliación** se desarrollo a base de las librerías gráficas, lo cual demostró la eficiencia y facilidad de manejo del paquete de gráficas tridimensionales para aplicaciones de diseño futuras.
 2. Debido que los motores DC de la grúa no son motores de velocidad constante, es decir sufren un pequeño retardo al cambiar de movimiento, así también el frenado no es instantáneo, la coordinación con el despliegue gráfico no se lleva a cabo en un 100%.
 3. La comunicación de órdenes entre el computador y la interfase digital es rápida lo que puede asegurarse su **utilización** para fines prácticos.
 4. Debido a los grandes cálculos matemáticos que debe realizarse con los puntos **que** definen el despliegue de la figura, se hace necesario la utilización un co-procesador matemático.
 5. El sistema de control de la Grúa se representa únicamente para efectuar movimientos sin presencia de pesos en el gancho de la Grúa, sin embargo si se



desea mayor torque para los motores. se necesita rediseñar el circuito de fuerza, añadiendole ademas un sistema de frenado.

En base a la experiencia realizada se pueden establecer las siguientes recomendaciones:

1. Las aplicaciones del paquete de gráficas tridimensionales es ilimitada. Tomando en cuenta que su costo se reduce cada vez más. Es necesario profundizar en este aspecto' para la introducción masiva de autómatas en la industria ecuatoriana.
2. Debido a la capacidad de los motores para mejorar la coordinación entre la animación gráfica y los movimientos de la Grúa-Robot, se puede rediseñar un circuito de fuerza para motores de mayor potencia, así como un servomecanismo para estabilidad.
3. Debido a la diferencia de velocidades de procesamiento del hardware en donde se desarrolle el programa que controla la grúa, es necesario coordinar los movimientos mediante el reajuste en los ángulos de rotación y los factores de traslación para el caso del gancho.

APENDICE A

ALGORITMOS DE LIBRERIAS GRAFICAS
Y DEL PROGRAMA GRUA

U N I D A D T A B L A S

INTERFASE Unidades lista,files,cara_ocu

variables

xx,yy,zz,point,count,suma_s:tres_puntos;

ar_x,ar_y,ar_z:dos_puntos;

apuntador,smax,contador:integer;

puntero_s,z,puntero_e,puntero_v,puntero_inicial:nodo;

Procedure enlace-de-tablar (punt_s,punt_e:nodo)

Se sirve del procedimiento:

Procedure concat_s_con_e

Para: Concatenar la tabla poligonos *S* con la
 tabla aristas *E* y la tabla aristas *E* se
 concatena con la tabla **vertices V**.

Procedure crea-tablas

Crea las tres tablas: Poligonos *S*, Aristas ***E***, Vertices***V***

Procedure obtiene-vertices (punt_s:nodo);

Se sirve de:

Procedure concat (var apuntador:integer);

Saca de *s* cada poligono de *e* cada arista y de *v* cada **vertice**

Procedure por_columna(var apunta:integer);

Llena los vectores *XX*, *YY*, y *ZZ* con los **vertices** de cada poligono

Para:

Sacar de la base de datos hacia tres arreglos definidos cono *xx*, *YY*, *ZZ* todos los vertices que definen cada trazo de línea.

Procedure marca-cabezas (punt_a_e:nodo);

Marca las cabezas de cada lista de la tabla *E* de las aristas comunes para no trazarlas doblemente.

Function marcada (apunta:nodo) : de tipo boolean

Comprueba si está marcada la cabeza de la lista de polígonos E

Procedure **arreglo-de-vertices** (punt_v:nodo)

Obtiene solo los vertices de la tabla V y los guarda en tres arreglos: ar_X, ar_Y, ar_Z

Procedure **actualizavertices** (punter_v:nodo)

Actualiza la base de datos

Procedure **marca-cabezas-de-s** (vector-unitario:unidad; puntero-s)

Marca en cuando se trata de una cara oculta.

Algoritmos de los Procedimientos

Procedure **crea-tablas;;**

begin

crea(u);

Si es cabeza de lista entonces puntero_a_tabla:=u;

p:=u;

c^.ptr1:=u;

for i=1 to n do

begin

x:=arr_temp[i];

inserta(p,x);

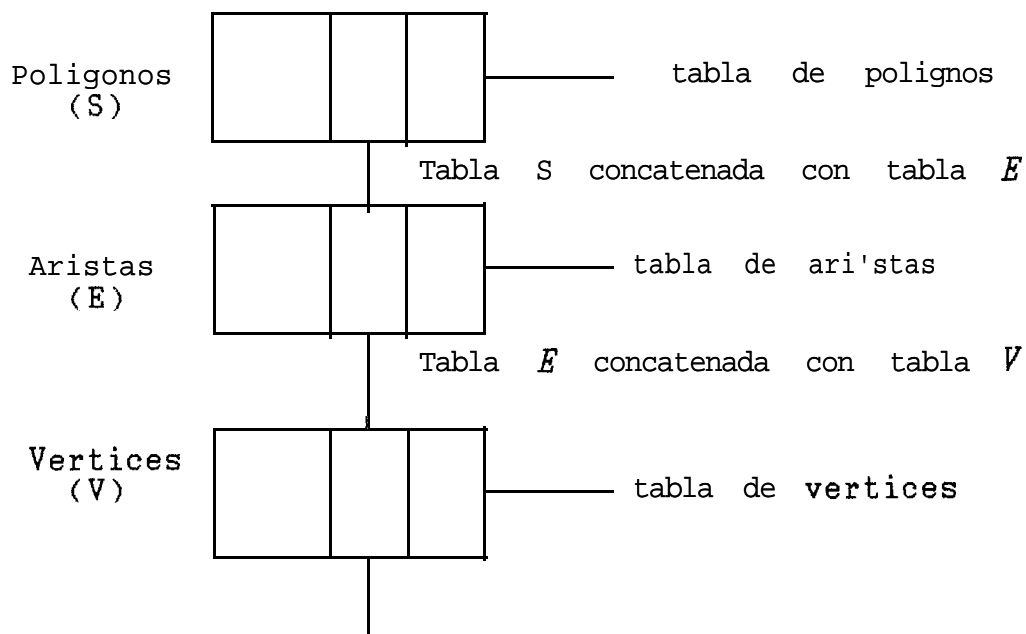
p:=p^.ptr2;

end;

```

o:=u;
end;{crea_tablas}

```



```

Procedure concat_s_con_e;

```

```

var

```

```

indice:integer;

```

```

casilla:real;

```

```

begin

```

```

    indice ← 1;

```

```

    mientras puntero_colum_s^.ptr2 diferente de nil hacer

```

```

        begin

```

```

            guardar el dato en casilla

```

```

            mientras (indice se diferente de casilla) hacer

```

```

                begin

```

```

        indice:=indice+1;
        puntero_fila_e:=puntero_fila_e^.ptr1;
    end;

```

```

    puntero_colum_s^.ptr2^.ptr1:=puntero_fila_e;
    indice:=1;
    puntero_colum_s:=puntero_colum_s^.ptr2;
    puntero_fila_e:=pointer_e;
    end;

```

```

    puntero_fila_s:=puntero_fila_s^.ptr1;
    puntero_colum_s:=puntero_fila_s;

```

```

end; {procedimiento concat-s-con-e}

```

Procedure enlace de tablas;

var

```

    puntero_fil_s,puntero_col_s:nodo;
    puntero_fil_e,puntero_col_e:nodo;

```

begin

```

    puntero-fil-s :=punt_s;

```

```

    puntero_col_s:=punt_s;

```

```

    puntero_fil_e:=punt_e;

```

```

mientras puntero_fil_s^.ptr1 sea diferente de nil

```

hacer

```

    concat-s-con-e (puntero-fil-s, puntero-col-s,
                    puntero-fil-e, punt_e);

```

```

                    unt_e);

```



```

        por_columna(apuntador,punt_v_temp);
        punt_e_temp:=punt_e_temp^.ptr2;
        apuntador :=apuntador+1;
    end;
end;
    puntero-colum-s :=puntero_colum_s^.ptr2;
end; {mientras}
    puntero_fila_s:=puntero_fila_s^.ptr1;
    puntero-colum-s :=puntero_fila_s;
end; {procedimiento concat}

```

Procedure obtiene-vertices;

```

begin
    apuntador:=1;
    puntero-fil-s :=punt_s;
    puntero_col_s:=punt_s;
    mientras puntero_fil_s^.ptr1<>nil hacer
    concat(apuntador,puntero_fil_s,puntero_col_s);
end; {obtiene-vertices}

```

Procedure arreglo_de_vertices;

```

begin
    arrow:=1;
    puntero-fil-v :=punt_v;
    puntero_col_v:=punt_v;
    mientras puntero_col_v<> nil hacer
        begin

```

```

        ar_x[arrow]:=puntero_col_v^.ptr2^.dato;
        puntero_col_v:=puntero_col_v^.ptr2;
        ar_y[arrow]:=puntero_col_v^.ptr2^.dato;
        puntero_col_v :=puntero_col_v^.ptr2;
        ar_z[arrow]:=puntero_col_v^.ptr2^.dato;
        puntero_col_v:=  puntero_col_v^.ptr2;
        arrow:=arrow+1;
        puntero_fil_v :=puntero_fil_v^.ptr1;
        puntero_col_v :=puntero_fil_v;
    end;

        ar_x[arrow]:=puntero_col_v^.ptr2^.dato;
        puntero_col_v :=puntero_col_v^.ptr2;
        ar_y[arrow]:=puntero_col_v^.ptr2^.dato;
        puntero_col_v :=puntero_col_v^.ptr2;
        ar_z[arrow]:=puntero_col_v^.ptr2^.dato;
        puntero_col_v:=  puntero_col_v^.ptr2;
    end; {procedimiento arreglo-de-vertices}

```

```

Procedure  actualizavertices;

```

```

var

```

```

actual_fil_v,actual_col_v:nodo;

```

```

begin

```

```

    actual:=1;

```

```

    actual_fil_v:=punter_v;

```

```

    actual_col_v:=punter_v;

```

```

    mientras actual_fil_v^.ptr1 <> nil hacer

```

```

        begin

```



```
        actual_col_v^.ptr2^.dato:= ar_x[actual];
        ananza actual-col-v;
        actual_col_v^.ptr2^.dato:=ar_y[actual];
        avanza actual-col-v;
        actual_col_v^.ptr2^.dato:=ar_z[actual];
        avanza actual-fil-v;
        actual_col_v:=actual_fil_v;
        actual:=actual+1;
    end;
end; {actualiza_vertices}

procedure marca-cabezas-de-s;
var
marca-s:nodo;
begin
    marca_s^.dato:=vector_unitario[i];
    marca-s :=marca_s^.ptr1;
end;
```

U N I D A D V I S I B L E

INTERFASE Lista:

Type

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

unidad=array[1..num_max_tab,num_max_pol] of real;

var

a,b,c,d:real;

temp,temp_xx,temp_yy,temp_zz:lados;

smax:integer;

vector-unitario:unidad;

point,count,suma_s,fil_x,fil_y,fil_z:tres_puntos;

Procedure obitene_vertices_dos (punt_s:nodo)

Se sirve de:

```
Procedure concat_dos (var puntador: integer;  
                      var punter_fila_s,  
                      punter_colum_s:nodo);
```

Controla los punteros por cada columna.

```
Procedure por_columna_dos(var punta:integer;  
                          var puntero_aux_v:nodo);
```

Guarda en un arreglo temporal los tres

vértices no colineales.

Para obtener los tres **vertices** no colineales de una cara que son utilizados para obtener la **ecuación** del plano de cada cara o polígono.

Procedure a_b_c_d (tempox,tempoy,tempoz:arreglos
temporales de los tres vértices no
colineales);

Calcula los valores de A,B,C,D para la ecuación del
plano $Ax + By + Cz + D = 0$.

Procedure ecuaciondelplano (vector-unitario:
vector encerado que va a contener las caras'ocultas);

Function cara-opuesta (cara:integer):integer;

Regresa la cara opuesta del polígono correspondiente, ya
que la proyección empleada es la proyección oblicua
reflejando los puntos en el plano $z=0$.

Procedure obitene_vertices_tres (punt_s,punt_v:
nodo);

Se sirve de:

Procedure concat_tres(var smax: integer;var
punter_fila_s,
punter_colum_s:nodo);

Controla la búsqueda de las aristas
visibles

Procedure por_columna_tres
(contador:integer;
punt_aux_v:nodo; var
count:tres_puntos);

Suma los valores de las coordenadas z de los

polígonos.

```

Procedure    fill_array (point:tres_puntos;
                          puntr_v:nodo);

```

Llena en tres arreglos
 (fil_x,fil_y,fil_z, con únicamente los
 vértices de los polígonos visibles de
 la figura.

Para:

1.- Obtener cuál(es) polígono(s) están más cercanos al plano de proyección para obtener de éste la, cara opuesta, ya que los puntos **rotan** hacia el plano de proyección y puedan ser visibles en dos dimensiones a través del dispositivo de video.

2.- Guardar los puntos que definen únicamente las caras que no son ocultas en tres arreglos.

Algoritmos de Procedimientos y Funciones

```

procedure fill_array
var
puntr_v_aux:nodo;
begin
    j:=1;
    puntr_v_aux:=puntr_v;
    i:= 1;
    repetir

```

```

mientras point[j]<>i do
  begin
    i:=i+1;
    puntr_v_aux:= puntr_v_aux^.ptr1;
  end;
  fil_x[j]:=puntr_v_aux^.ptr2^.dato;
  fil_y[j]:=puntr_v_aux^.ptr2^.ptr2^.dato;
  fil_z[j]:=puntr_v_aux^.ptr2^.ptr2^.ptr2^.dato;
  j:=j+1;
  i:=1;
  hasta j=5;
end;

```

Procedure por-columnatres

```

begin
  count[contador]:=valor de la coordenada z;
end;

```

Procedure concat_tres;

```

var
  punt_e_temp,punt_v_temp:nodo;
begin
  encera_point(point);
  contador:=1;
  mientras punter_colum_s^.ptr2 <> ni.1 hacer
    begin
      Se obtiene la coordenada z
      por_columna_tres(contador,coordenada z);
    end;
end;

```

```

        contador := contador + 1;
        siguiente arista
        avanza puntercolumns;
    end; {while}
    suma-coordenadas-z;
    punter_fil_s := punter_fil_s^.ptr1;
    punter_colum_s := punter_fil_s;
end; {procedimiento concat_tres}

```

Procedure obtiene-verticestres;

var

```

    punter_fil_s, punter_col_s : nodo;

```

begin

```

    punter_fil_s := punt_s;

```

```

    punter_col_s := punt_s;

```

```

    mientras punter_fil_s^.ptr1 <> nil hacer

```

```

        concat_tres(smax, punter_fil_s, punter_col_s);

```

end; {obtiene-vertices-tres}

Procedure por_columna_dos;

begin

```

    temp_xx[a] := puntero_aux_v^.ptr2^.dato;

```

```

    puntero_aux_v := puntero_aux_v^.ptr2;

```

```

    temp_yy[a] := puntero_aux_v^.ptr2^.dato;

```

```

    puntero-aux-v := puntero_aux_v^.ptr2;

```

```

    temp_zz[a] := puntero_aux_v^.ptr2^.dato;

```

```

    if a = 3 then

```

```

        a_b_c_d(temp_xx, temp_yy, temp_zz);

```

```
end;{por_columna_dos}
```

```
Procedure concat_dos;
```

```
var
punt_e_temp,punt_v_temp:nodo;
begin
    puntador:=1;
    punt_e_temp:=punter_colum_s^.ptr2^.ptr1;
    dato_1:=punt_e_temp^.ptr2^.dato;
    dato_2:=punt_e_temp^.ptr2^.ptr2^.dato;
    almacena las coordenadas de dato-1 y dato-2
    por_columna_dos(puntador,punt_v_temp);
    punter_colum_s:=punter_colum_s^.ptr2;
    punt_e_temp:=punter_colum_s^.ptr2^.ptr1;
    dato_3:=punt_e_temp^.ptr2^.dato;
    si dato-3 es el tercer vertice no colineal
    entonces
        begin
            punt_v_temp:=punt_e_temp^.ptr2^.ptr1;
            {almacena las coordenadas de dato_3}
            por_columna_dos(puntador,punt_v_temp);
            puntador :=puntador+1;
        end;
        punt_e_temp:=punt_e_temp^.ptr2;
    end;
    punter_fila_s:=punter_fila_s^.ptr1;
    punter_colum_s:=punter_fila_s;
```

```
end: {procedimiento concat}
```

```
Procedure obitenegertices-dos;
```

```
var
```

```
    pun_fil_s,pun_col_s:nodo;
```

```
begin
```

```
    puntador:=1;
```

```
    pun_fil_s :=primer puntero de S;
```

```
    pun_col_s:=primer puntero de S;
```

```
    mientras no sea fin de tabla hacer
```

```
        concat_dos(puntador,pun_fil_s,pun_col_s);
```

```
end; {obtiene_vertices}
```

```
Procedure a_b_c_d;
```

```
var
```

```
x1,x2,x3,y1,y2,y3,z1,z2,z3:real;
```

```
begin
```

```
    x1:=tempox[1];
```

```
    y1:=tempoy[1];
```

```
    z1:=tempoz[1];
```

```
    x2:=tempox[2];
```

```
    y2:=tempoy[2];
```

```
    z2:=tempoz[2];
```

```
    x3:=tempox[3];
```

```
    y3:=tempoy[3];
```

```
    z3:=tempoz[3];
```

```
    A:=y1*(z2-z3) + y2*(z3-z1) +y3*(z1-z2);
```



```

B:=z1*(x2-x3) + z2*(x3-x1) +z3*(x1-x2);
C:=x1*(y2-y3) + x2*(y3-y1) +x3*(y1-y2);
D:=-x1*(y2*z3-y3*z2) - x2*(y3*z1-y1*z3) -
    x3*(y1*z2-y2*z1);

```

almacena los valores de A,B,C,D

Procedure ecuacion_del_plano

```

const
    equis=40;
    ye=25;
    zeta=40;
begin
    for i:=1 to mar-index-2 do
        begin
            resultado:= A[i,1]*equis + B[i,2]*ye + C[i,3]*zeta +
                D[i,4]; {ecuacion del plano}
            if resultado>0 then
                vector_unitario[i]:=cara visible;
        end;
        cara_mas_cerca_del_plano_proyeccion= find_max();
        vector_unitario[cara_mas_cerca]:=cara no visible;
        cara opuesta es una cara visible
    end;
end;

```

UNIDAD DOS-D

INTERFASE Unidades Tablas, Lista

Type

```
vector_de_enteros_dobles=array[1..num_max_tab,1..num_max  
                                quntos] of integer;
```

Var

```
vector_x,vector_y,vec_x,vec_y:vector_de_enteros_dobles;
```

Procedure transforma-paras-coordenados

```
(i: integer; xx,yy,zz:tres_por_tres;var  
                                vect_x,vect_y:vector_de_enteros_dobles);
```

Efectúa la transformación de las coordenadas mundiales en coordenadas del dispositivo de video, almacena los valores en los vectores **vect_x** y **vect_y**.

Algoritmo del Procedimiento

Const

```
profundidad=1;
```

begin

```
for j:=1 to numero-de-puntos do
```

```
begin
```

```
{Aplicando la fórmula  $x' = x + z * profundidad * \cos \phi$ 
```

```
 $y' = y + z * profundidad * \sin \phi$ 
```

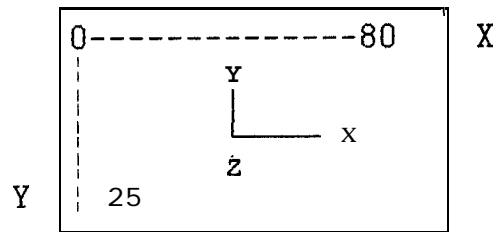
```
Definida para la proyección oblicua de los puntos  
sobre el plano de proyección}
```

```
x1:= xx[i,j] + zz[i,j] * profundidad * cos  $\phi$ ;
y1:= yy[i,j] + zz[i,j] * profundidad * cos  $\phi$ ;
```

{Transformación a coordenadas bidimensionales)

```
x1:=abs(x1+40)*8;
```

```
si (y1<=0) entonces
  y1:= abs (y1+12.5)*19;
sino
  y1:=(abs(y1-12.5))*19
```



```
end;
```

U N I D A D M O T I O N :

INTERFASE Unidades Tablas, Lista:

type

matriz= array[1..4,1..4] of real;

var

t:matriz;

Procedure identidad (m:matriz);

Transforma a matriz identidad

Procedure combine-transformaciones (t,m:matriz)

Eslabona la matriz t con m y el resultado lo guarda en t

Procedure escala (sx,sy,sz:factores de escalacion;

xf,yf,zf: punto de referencia)

Realiza la escalación de la figura

Procedure traslada (tx,ty,tz: factores de

traslación);

Realiza la traslación de la figura

Procedure transforma_puntos (señal:integer; var

a_x,a_y,a_z:tres_puntos);

Actualiza los arreglos que contienen los vértices con los nuevos valores luego de efectuarse las **transformaciones** respectivas.

Procedure rota_x (a:ángulo de rotación en el eje X);

Procedure rota-x (a:ángulo de rotación en el eje Y);

Procedure rota-z (a:ángulo de rotación en el eje Z);

Procedure rota-con-eje arbitrario (a:ángulo de rotación; xx1, yy1, zz1, xx2, yy2, zz2: coordenadas del eje de rotación);

ALGORITMOS DE LOS PROCEDIMIENTOS

Procedure Transforma_Puntos;

begin

for k:=1 to señal do

begin

tempx:=a_x[k] * t[1,1] + a_y[k] *t[2,1] + t[3,1]*
a_z[k] + t[4,1];

a_y[k]:=a_x[k]*t[1,2] + a_y[k]*t[2,2] + t[3,2]*a_z[k]
+ t[4,2];

a_x[k] :=tempx;

a_z[k]:=a_x[k]*t[1,3] + a_y[k]*t[2,3] + t[3,3]*a_z[k]
+ t[4,3];

end;

end;

Procedure combine_transformations;

begin

for r:=1 to 4 do

for d:=1 to 4 do

```
temp[r,d]:=t[r,1]*m[1,d] + t[r,2] *m[2,d] +t[r,3]
           *m[3,d] +t[r,4]*m[4,d];
```

```
for r:=1 to 4 do
```

```
  for d:=1 to 4 do
```

```
    t[r,d]:=temp[r,d];
```

```
end;
```

Procedure Escala;

```
begin
```

```
  identidad(m);
```

```
    m[1,1]:=sx;
```

```
    m[2,2]:=sy;
```

```
    m[3,3]:=sz;
```

```
    m[4,1]:=(1 - sx) *xf;
```

```
    m[4,2]:=(1 - sy) *yf;
```

```
    m[4,3]:=(1 - sz) *zf;
```

```
  combine-transformaciones(t,m);
```

```
end;
```

Procedure traslada;

```
begin
```

```
  m[4,1]:=tx;
```

```
  m[4,2]:=ty;
```

```
  m[4,3]:=tz;
```

```
  combine-transformaciones(t,m);
```

```
end;
```

Procedure rota_x;

begin

 identidad(m);

 a:=radian_equivalente(a);

 m[2,2]:=cos(a);

 m[2,3]:=sen(a);

 m[3,2]:=-sen(a);

 m[3,3]:=cos(a);

 combine-transformaciones(t,m);

end;

Procedure rota_y;

begin

 identidad(m);

 a:=radian_equivalente(a)

 m[1,1]:=cos(a);

 m[3,1]:=sen(a);

 m[1,3]:=-sen(a);

 m[3,3]:=cos(a);

 combine_transformaciones(t,m);

end;

Procedure rota_z;

var

 ca,sa:real;

begin

```

    identidad(m);
    a:=radian_equivalente(a);
    ca:=cos(a);
    sa:=sin(a);
    m[1,1]:=ca;
    m[1,2]:=sa;
    m[2,1]:=-sa;
    m[2,2]:=ca;
    combine-transformaciones(t,m);
end;

```

Procedure rotacion_con_eje_arbitrario;

var

a,b,c,d,long:real;

begin

identidad(m);

a:=xx2-xx1; { determina el vector unitario

b:=yy2-yy1; paralelo al eje de rotación)

c:=zz2-zz1;

long:= raiz (a²+b² +c²)

{traslada hacia el origen}

traslada{-xx1, -yy1, -zz1};

identidad(m);

m[2,2]:= c/d; m[2,3]:= b/d;

m[3,2]:= -b/d; m[3,3]:= c/d;

combine-transformaciones(t,m);

{giro en torno al eje x para quedar en el plano **xz**}

identidad(m);

m[1,1]:= d; m[1,3]:= a;

m[3,1]:= -a; m[3,3]:= d;

combine_transformaciones(t,m);

{rotación en torno al eje y para alinear con el
eje z}

identidad(m);

rota-z(a);

combine-transformaciones(t,m);

{rotación inversa en torno al eje y}

identidad(m);

m[1,1]:= d; m[1,3]:= -a;

m[3,1]:= a; m[3,3]:= d;

combine-transformaciones(t,m);

{rotación inversa en torno al eje x}

identidad(m);

m[2,2]:= c/d; m[2,3]:= -b/d;

m[3,2]:= b/d; m[3,3]:= c/d;

combine_transformaciones(t,m);

{traslación inversa que devuelve el vector a su
posición original}

traslada(xx1,yy1,zz1)

combine_transformaciones(t,m);

end;

U N I D A D I M A G E N

INTERFASE Dos-d, Tablas, Graph, Visible

var

gd,gm:integer;

polypoints:array[1..numpoints] of pointtype;

Procedure ini moda grafico;

Inicializa el sistema gráfico y pone al hardware en modo gráfico.

Procedure traza-lineas (par-x, par-y:

vector-de-enteros-dobles);

Obtiene el punto inicial y el punto final de cada arista de la figura de los vectores que contienen las coordenadas bidimensionales. El trazo se realiza por medio del procedimiento del Pascal `Line(X1,X2,Y1,Y2)`.

Procedure llena_poligono (par_x,par_y:

vector-de-enteros-dobles;

vector-unitario: unidad);

Algoritmos de los Procedimientos

Procedimiento traza-líneas

begin

repetir

line(par_x1,par_y1,par_x2,par_y2);

hasta fin de arreglos;

end;

```

Procedure Fill_format(opcion: integer);
{establece el tipo de relleno de cada polígono visible}
begin
  case opcion of
    2: setfillstyle(closedotfill,1);
    3: setfillstyle(interleavefill,1);
    1: setfillstyle(closedotfill,1);
  end;
end;

```

```

Procedure llena_poligono

```

```

begin
  k:=0;
  pik:=1;
  for i:=1 to as do
  Si es una cara visible entonces
  begin
    {llena al arreglo de polypoints con los vértices
    que definen cada poligono}
    polypoints.x:=par_x;
    polypoints.y:=par_y;
    {establece el formato de salida;
    fillpoly(numpoints,polypoints); {procedimiento de
    Fascal}
  end;
end;

```

UNIDAD LISTA;

type

```
nodo = apuntador a celda( ^celda) ;  
    celda = registro  
        dato:real;  
        ptr1,ptr2 son de tipo nodo  
    end;
```

tres_puntos=array[1..6] of real;

var

```
u,p,l,t,o,k:nodo;  
x:real;
```

Function vacia (recibe el nodo u y retorna un valor de cierto si está vacia la celda o falso si no es asi):
boolean;

Procedure crea(crea un nodo u);

Procedure inserta(inserta una nueva celda en la lista);

Algoritmos de funciones y procedimientos

Function vacia;

begin

```
    si u^.ptr2=nil  
    entonces vacia=cierto;
```

```

    sino vacia=false;
end;

```

```

Procedure crea;

```

```

begin

```

```

    new(u);

```

```

    u^.ptr2=nulo;

```

```

    u^.ptr1=nulo;

```

```

end;

```

```

Pocedure inserta;

```

```

var

```

```

    temp:nodo;

```

```

begin

```

```

    new(temp);

```

```

    temp^.dato=x; {a la nueva celda le damos en su campo
                    dato el valor de x}

```

```

    temp^.ptr2=p^.ptr2;

```

```

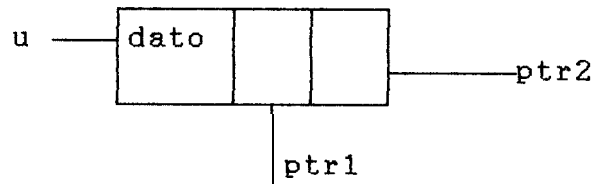
    p^.ptr2=temp;

```

```

end;

```




```

                                puntero-v});

    end;

identidad(t);

repeat

    inicia_modos_grafico;

    menu_inicial;

    PRINCIPAL;

    readln(opcion);

    case opcion of

        1:  TRASLACION  DEL  GANCHO

            SUBIR  GANCHO:  begin

                enviar señal al puerto SUBE-GRUA;

                repetir

                    CLEAR  PANTALLA;

                    traslada(0,0.5,0);

                    transforma_ puntos(ar_x,ar_y,ar_z);

                    PRINCIPAL;

                    hasta presionar 'ENTER'  ;

                    encera-puerto;

                end;

            BAJA  GANCHO  :  begin

                CLEAR  PANTALLA;

                enviar señal al puerto BAJA-GRUA;

                Repetir

                    traslada(0,-0.5,0);

                    transforma_puntos(ar_x,ar_y,ar_z);

                    PRINCIPAL;

```



```

hasta presionar 'ENTER'
encera-puerto
end;

```

2: ROTACION DE LA GRUA

```

ABRE BRAZO: begin
    enviar señal al puerto ABRIR-BRAZO;
    repetir
    CLEAR PANTALLA;
    rota_z(10);
    transforma_puntos(8,ar_x,ar_y,ar_z);
    PRINCIPAL;
    hasta presionar 'ENTER'
    encera-puerto;
end;

```

```

CIERRA BRAZO:begin
    enviar señal al puerto CERRAR_BRAZO;
    repetir
    CLEAR PANTALLA;
    rota-z(-10);
    transforma_puntos(8,ar_x,ar_y,ar_z);
    PRINCIPAL;
    hasta presionar 'ENTER'
    encera-puerto;
end;

```

```

GIRO EJE Y+: begin
    enviar señal al puerto de Giro-y-(+);
    repetir

```

```
CLEAR PANTALLA:
rota_y(10)
transforma_puntos(8,ar_x,ar_y,ar_z);
PRINCIPAL;
hasta Presionar 'ENTER'
encera-puerto
end;

GIRO EJE Y-: begin
enviar señal al puerto de Giro-y(-);
repetir
CLEAR PANTALLA;
rota-y(-10);
transforma_puntos(8,ar_x,ar_y,ar_z);
PRINCIPAL;
hasta presionar 'ENTER';
enceraquerto;
end;

4: SALIR
end.
```

AE'ENDICE B

LISTADOS DE LIBRERIAS GRAFICAS
Y DEL PROGRAMA GRUA

```

unit tablas:
interface
uses lista, visible:
var
    ar_x,ar_y,ar_z:tres_por_tres;
    n,m,j,apuntador,smax,contador:integer;
    puntero_s,z,puntero_e,puntero_v,puntero_inicial:nodo;
    band,uno:boolean;
    pointers:pointer_array;
procedure concat_s_con_e(var puntero-fila-s,
    puntero_colum_s, puntero-fila-e:nodo;
    pointer_e:nodo);
procedure enlace_de_tablas(punt_s,punt_e:nodo);
procedure crea_tablas(var puntero_a_tabla:nodo;
    dos:boolean; n:integer; arr_temp:tipo);
procedure valores_iniciales(var bandera:boolean;var
    cc:nodo);
procedure obtiene_vertices(punt_s:nodo;trie:integer);
procedure concat(var apuntador:integer;var
    puntero_fila_s,puntero_colum_s:nodo;trie:integer);
procedure por_columna(var apunta:integer;var
    puntero_aux_v:nodo; trie:integer);
procedure encera_cabezas(punt_a_s:nodo);
procedure marca_cabezas(punt_a_e:nodo);
function marcada(apunta:nodo):boolean;
procedure encera_vectores(var vx,vy,vz:tres_puntos);
procedure arreglo_vertices(punt_v:nodo; tables:integer);

```

```

    var ar_x,ar_y,ar_z: tres-por-tres);
procedure actualiza_vertices(punter_v:nodo;
    orh:integer; ar_x, ar_y, ar_z: tres-por-tres);
procedure marca-cabezas-de-s(vector_unitario:unidad;
    puntero-s:nodo; numero:integer; yes:integer);
procedure marca_cabezas_de_v(pun_v_ini:nodo);

implementation

procedure encera-vectores;
begin
for i:=1 to 50 do
begin
    vx[i]:=0;
    vy[i]:=0;
    vz[i]:=0;
end;
end;

procedure crea-tablas;

begin
    crea(u);
    if (band) then puntero_a_tabla:=u;
    band:=false;
    p:=u;
    c^.ptr1:=u;
    for i:=1 to n do
        begin

```

```

        x:=arr_temp[i];
        inserta(p,x);
        p:=p^.ptr2;
    end;
c:=u;
end;{crea_tablas}

procedure  valores-iniciales;
begin
    bandera:=true;
    cc:=nil;
end;

procedure  concat_s_con_e;
var
    indice:integer;
    casilla:real;
begin
    indice:=1;
    while puntero_colum_s^.ptr2 <> nil do
        begin
            casilla:=puntero_colum_s^.ptr2^.dato;
            while (indice<>casilla) do
                begin
                    indice:=indice+1;
                    puntero-fila-e:=
                        puntero_fila_e^.ptr1;

```

```

        end:
            puntero_colum_s^.ptr2^.ptr1:=
                puntero-fila-e;
            indice:=1;
            puntero-columns :=puntero_colum_s^.ptr2;
            puntero_fila_e:=pointer_e;
        end;
        puntero-fila-s :=puntero_fila_s^.ptr1;
        puntero_colum_s:=puntero_fila_s;
    end; {procedimiento concat_s_con_e}

procedure enlace-de-tablas;
    var
        puntero_fil_s,puntero_col_s:nodo;
        puntero_fil_e,puntero_col_e:nodo;
begin
    puntero-fil-s :=punt_s;
    puntero_col_s:=punt_s;
    puntero_fil_e:=punt_e;
    while puntero_fil_s^.ptr1<>nil do
        concat_s_con_e(puntero_fil_s, puntero-col-s,
            puntero-fil-e ,punt_e);
    end;{enlace_tablas}

function marcada;
begin
    if apunta^.dato=0 then
        marcada:=FALSE

```

```
else
  marcada:=TRUE;
end;

procedure marca-cabezas;
begin
  punt_a_e^.dato:=1;
end;

procedure encera-cabezas;
var
  punt_cab:nodo;
begin
  punt_cab:=punt_a_s;
  punt_cab^.dato:=0;
  while punt_cab^.ptr1<>nil do
  begin
    punt_cab^.ptr1^.dato:=0;
    punt_cab:=punt_cab^.ptr1;
  end;
end;

procedure por-columna;
var
  ind:integer;
begin
  xx[trie,apunta]:=puntero_aux_v^.ptr2^.dato;
  puntero_aux_v:=puntero_aux_v^.ptr2;
```



```

yy[trie,apunta]:=puntero_aux_v^.ptr2^.dato;
puntero_aux_v:=puntero_aux_v^.ptr2;
zz[trie,apunta]:=puntero_aux_v^.ptr2^.dato;

end;

procedure concat;
var
.j:integer;
punt_e_temp,punt_v_temp:nodo;
count:tres_puntos;
begin
if marcada(puntero_fila_s) then
while puntero_colum_s^.ptr2 <> nil do
begin
punt_e_temp:=puntero_colum_s^.ptr2^.ptr1;
if not(marcada(punt_e_temp)) then
begin
marca_cabezas(punt_e_temp);
for j:=1 to 2 do
begin
punt_v_temp:=punt_e_temp^.ptr2^.ptr1;
por_columna(apuntador,punt_v_temp,trie);
punt_e_temp:=punt_e_temp^.ptr2;
apuntador :=apuntador+1;
end;
end;
puntero_colum_s:=puntero_colum_s^.ptr2;

```

```

        end:
            puntero_fila_s:=puntero_fila_s^.ptr1;
            puntero_colum_s:=puntero_fila_s;
        end;

procedure  obtiene-vertices;
var
    puntero_fil_s,puntero_col_s:nodo;
begin
    apuntador:=1;
    puntero-fil-s :=punt_s;
    puntero_col_s:=punt_s;
    while puntero_fil_s^.ptr1<>nil do
concat(apuntador,puntero_fil_s,puntero_col_s,trie);
concat(apuntador,puntero_fil_s,puntero_col_s,trie);
end; {obtiene_vertices}

procedure  arreglo-deuertices;
var
    puntero_fil_v,puntero_col_v:nodo;
    arrow:integer;
begin
    arrow:=1;
    puntero-fil-v :=punt_v;
    puntero_col_v:=punt_v;
    while puntero_col_v<> nil do
begin
    ar_x[tables,arrow]:=puntero_col_v^.ptr2^.dato;

```

```

    puntero_col_v:=puntero_col_v^.ptr2;
    ar_y[tables,arrow]:=puntero_col_v^.ptr2^.dato;
    puntero-col-v :=puntero_col_v^.ptr2;
    ar_z[tables,arrow]:=puntero_col_v^.ptr2^.dato;
    puntero-col-v:= puntero_col_v^.ptr2;
    arrow:=arrow+1;
    puntero-fil-v :=puntero_fil_v^.ptr1;
    puntero-col-v :=puntero_fil_v;
    end;
end; {procedimiento arreglo_de_vertices}

```

```

procedure marca-cabezas-de-v;

```

```

var

```

```

y:integer;

```

```

punt_v_t:nodo;

```

```

begin

```

```

    punt-v-t :=pun_v_ini;

```

```

    punt_v_t:=punt_v_t^.ptr1;

```

```

    punt_v_t^.dato:=1;

```

```

    punt-v-t :=punt_v_t^.ptr1;

```

```

    punt_v_t^.dato:=1;

```

```

    punt-v-t :=punt_v_t^.ptr1;

```

```

    punt-v-t: =punt_v_t^.ptr1;

```

```

    punt_v_t:=punt_v_t^.ptr1;

```

```

    punt_v_t^.dato:=1;

```

```

    punt-v-t :=punt_v_t^.ptr1;

```

```

    punt_v_t^.dato:=1;

```

```
end;
```

```
procedure  actualiza-yertices;
```

```
var
```

```
actual:integer;
```

```
actual_fil_v,actual_col_v:nodo;
```

```
begin
```

```
    actual:=1;
```

```
    actual_fil_v:=punter_v;
```

```
    actual_col_v:=punter_v;
```

```
    while actual_fil_v^.ptr1 <> nil do
```

```
        begin
```

```
            if not(marcada(actual_fil_v)) then
```

```
                begin
```

```
                    actual_col_v^.ptr2^.dato:= ar_x[orh,actual];
```

```
                    actual-col-v :=actual_col_v^.ptr2;
```

```
                    actual_col_v^.ptr2^.dato:=ar_y[orh,actual];
```

```
                    actual_col_v:=actual_col_v^.ptr2;
```

```
                    actual_col_v^.ptr2^.dato:=ar_z[orh,actual];
```

```
                end;
```

```
                actual_fil_v:=actual_fil_v^.ptr1;
```

```
                actual-col-v:=actualfilu;
```

```
                actual:=actual+1;
```

```
            end; {while}
```

```
            if not(marcada(actual_fil_v)) then
```

```
                begin
```

```
                    actual_col_v^.ptr2^.dato:= ar_x[orh,actual];
```

```

    actual_col_v:=actual_col_v^.ptr2;
    actual_col_v^.ptr2^.dato:=ar_y[orh,actual];
    actual-col-v :=actual_col_v^.ptr2;
    actual_col_v^.ptr2^.dato:=ar_z[orh,actual];
    actual-fil-v :=actual_fil_v^.ptr1;
    actual_col_v:=actual_fil_v;
    end
end;  {actualiza-vertices}

procedure  marca-cabezas-de-s;
var
    is:integer;
    marca-s:nodo;
begin
    marca_s:=puntero_s;
    for is:=1 to numero do
        begin
            marca-s-. dato:=vector_unitario[yes,is];
            marca_s:=marca_s^.ptr1;
        end;
    end;
begin
end.

```

```
unit Visible;

interface

uses crt, lista;

type

  arreg=array[1..num_max_pol,1..4] of real;
  lados=array[1..3] of real;
  unidad=array[1..num_max_tab,1..num_max_pol] of real;

var

  mar:arreg;
  puntador:integer;
  temp:lados;
  temp_xx,temp_yy,temp_zz:lados;
  mar-index:integer;
  vector-unitario:unidad;
  point,count,suma_s:tres_puntos;
  smax,contador,di_ar:integer;
  punter_v:nodo;
  xx,yy,zz:tres_por_tres;

procedure obtiene_vertices_dos(punt_s:nodo);
procedure concat_dos(var puntador:integer;var
punter_fila_s,punter_colum_s:nodo);
procedure por-columna-dos(var punta:integer;var
puntero_aux_v:nodo);
```

```

procedure                                encera_vector_unitario(var
vect_unitario:unidad);
procedure  a_b_c_d(tempox,tempoy,tempoz: lados);
procedure                                ecuacion_del_plano(marte:arreg;var
vector-unitario:unidad;tnt:integer);
function   cara-opuesta(anterior:integer):integer;
function   busqueda_de_vertice (value:real; indx:integer;
var point:tres_puntos):boolean;
function   find_max (max_s:tres_puntos; unimax:integer)
:integer;
procedure                                inserta_nuevo_vertice(data1:real;var
point:tres_puntos;contador:integer);
procedure  obtiene-vertices-tres (punt_s,punt_v :nodo;
tnt: integer);
procedure                                concat_tres(var                smax:integer;var
punter_fila_s,punter_colum_s,punter_v:nodo;tnt:integer);
procedure                                suma-coordenadas-z                (count:tres_puntos;
six:integer; smax:integer);
procedure                                por-columna-tres                (contador:integer;
punt_aux_v:nodo; var count:tres_puntos);
procedure  encera_point(var points:tres_puntos);
procedure  ordena_point(var point:tres_puntos);
procedure  fill_array(point:tres_puntos;puntr_v:nodo;var
d_ar,tnt:integer);

implementation

function  cara-opuesta;

```

```
type
point_type=array[1..num_max_pol] of integer;
var
  point_matrix: point_type;
  omd:integer;
begin
  point_matrix[1]:=3;
  point_matrix[2]:=4;
  point_matrix[3]:=1;
  point_matrix[4]:=2;
  point_matrix[5]:=6;
  point_matrix[6]:=5;
  cara_opuesta:=point_matrix[anterior];
end;

procedure enceragoint;
begin
  for i:=1 to num_max_pol do
    points[i]:=0;
  end;

function busqueda_de_vertice;
var
  ym:integer;

begin
```



```
busqueda_de_vertice:=FALSE;
  for ym:=1 to indx do
    if (point[ym]=value) then
      busqueda_de_vertice:=TRUE
    end;
end;
```

```
function find_max;
var
maximo:real;
maxi:integer;
begin
  j:=0;
  maximo:=max_s[1];
  for i:=1 to unimax do
    if max_s[i]>maximo then
      maximo:=max_s[i];
    repeat
      j:=j+1;
    until maximo=max_s[j];
  find_max:=j;
end;
```

```
procedure inserta-nuevo-vertice;
begin
  point[contador]:=data1;
```

```
end;
procedure ordenaoint;
var
  jj, kk: integer;
  auxi: real;
begin
  for jj:=1 to 3 do
    for kk:=jj+1 to 4 do
      if point[kk]> point[jj] then
        begin
          auxi:=point[jj];
          point[jj]:=point[kk];
          point[kk]:=auxi;
        end;
      end;
    end;
  end;
end;
```

```
procedure fill_array;
var
  j1: integer;
  ii: real;
  puntr_v_aux: nodo;
begin
  j1:=1;
  puntr_v_aux:=puntr_v;
  ii:= 1;
  repeat
    while point[j1]<>ii do
```

```

begin
    ii:=ii+1;
    puntr_v_aux:= puntr_v_aux^.ptr1;
end;

fil_x[tnt,d_ar]:=puntr_v_aux^.ptr2^.dato
fil_y[tnt,d_ar]:=puntr_v_aux^.ptr2^.ptr2^.dato;
fil_z[tnt,d_ar]:=puntr_v_aux^.ptr2^.ptr2^.ptr2^.dato;
    j1:=j1+1;
    d_ar:=d_ar+1;
    ii:=1;
    puntr_v_aux:=puntr_v;
until j1=5;
end;

procedure suma-coordenadas-z;
begin
    for j:=1 to six do
        suma_s[smax]:=suma_s[smax]+count[j];
    end;
procedure por-columna-tres;
var
ind :: integer;

begin
    count[contador]:=punt_aux_v^.ptr2^.ptr2^.ptr2^.dato;
end;

```

```

procedure concat_tres;
var
j:integer;
punt_e_temp,punt_v_temp:nodo;
data1:real;
begin
    encera_point(point);
    contador:=1;
    while punter_colum_s^.ptr2 <> nil do
        begin
            punt_e_temp:=punter_colum_s^.ptr2^.ptr1;
            for j:=1 to 2 do
                begin
                    data1:=punt_e_temp^.ptr2 .dato;
                    if          not(busqueda_de_vertice
                    (data1,contador-1,point)) then
                        begin
inserta_nuevo_vertice(data1,point,contador);
punt_v_temp:=punt_e_temp^.ptr2^.ptr1;
por_columna_tres(contador,punt_v_temp,count);
contador:=contador+1;
                        end;{if}
                            punt_e_temp:=punt_e_temp^.ptr2;
                        end; {for}
                            punter_colum_s:=punter_colum_s^.ptr2;
                    end; {while}

```

```

        suma-coordenadas-z(count,contador-1,smaxI;
        smax:=smax+1;

        ordena_point(point);
        fill_array(point,punter_v,di_ar,tnt);
        punter_fila_s:=punter_fila_s^.ptr1;
        punter_colum_s:=punter_fila_s;
    end; {procedimiento concat_tres}

procedure   obtienevertices-tres;
var
    punter_fil_s,punter_col_s  :nodo;
begin
    smax:=1;
    di_ar:=1;
    encera_point(point);
    encera_point(suma_s);
    punter_fil_s:=punter_s;
    punter_col_s:=punter_s;
    punter_v:=punter_v;
    while punter_fil_s^.ptr1<>nil do

concat_tres(smax,punter_fil_s,punter_col_s,punter_v,tnt)
concat_tres(smax,punter_fil_s,punter_col_s,punter_v,tnt)
end;   {obtieneyertices-tres}

procedure   por-columna-dos;

```

```

var
ind,index:integer;
begin
    begin
        temp_xx[punta]:=puntero_aux_v^.ptr2^.dato;
        puntero_aux_v:=puntero_aux_v^.ptr2;
        temp_yy[punta]:=puntero_aux_v^.ptr2^.dato;
        puntero-aux-v:=puntero_aux_v^.ptr2;
        temp_zz[punta]:=puntero_aux_v^.ptr2^.dato;
        if punta=3 then
            a_b_c_d(temp_xx,temp_yy,temp_zz);
        end;
    end;
end;
procedure concat_dos;
var
j,excel:integer;
punt_e_temp,punt_v_temp:nodo;
dato_1,dato_2,dato_3:real;
    begin
        puntador:=1;
        punt_e_temp:=punter_colum_s^.ptr2^.ptr1;
        dato_1:=punt_e_temp^.ptr2^.dato;
        dato_2:=punt_e_temp^.ptr2^.ptr2^.dato;
        for j:=1 to 2 do
            begin
                punt_v_temp:=punt_e_temp^.ptr2^.ptr1;
                por_columna_dos(puntador,punt_v_temp);
            end;
        end;
    end;
end;

```

```

        punt_e_temp:=punt_e_temp^.ptr2;
        puntador:=puntador+1;
    end; {for}
    punter_colum_s:=punter_colum_s^.ptr2;
    punt_e_temp:=punter_colum_s^.ptr2^.ptr1;
    for j:=1 to 2 do
    begin
        dato_3:=punt_e_temp^.ptr2^.dato;

        if (dato_3<>dato_1) and (dato_3<>dato_2)
        then
        begin
            punt_v_temp:=punt_e_temp^.ptr2^.ptr1;
            por_columna_dos(puntador,punt_v_temp);
            puntador:=puntador+1;
        end;
        punt_e_temp:=punt_e_temp^.ptr2;
    end;
    punter_fila_s:=punter_fila_s^.ptr1;
    punter_colum_s:=punter_fila_s;
end; {procedimiento concat}

```

```

procedure  obtiene-vertices-dos;

```

```

var

```

```

    pun_fil_s,pun_col_s:nodo;

```

```

begin

```

```

    mar_index:=1;

```

```

puntador:=1;
pun_fil_s :=punt_s;
pun_col_s:=punt_s;

while pun_fil_s^.ptr1<>nil do
concat_dos(puntador,pun_fil_s,pun_col_s);
concat_dos(puntador,pun_fil_s,pun_col_s);
end; {obtiene_vertices_dos}

procedure encera-vector-unitario;
begin
for i:=1 to num_max_tab do
for j:=1 to num_max_pol do
vect_unitario[i,j]:=0;
end;
procedure a_b_c_d;
var
x1,x2,x3,y1,y2,y3,z1,z2,z3:real;
a,b,c,d:real;
begin
x1:=tempox[1];
y1:=tempoy[1];
z1:=tempoz[1];
x2:=tempox[2];
y2:=tempoy[2];
z2:=tempoz[2];
x3:=tempox[3];

```



```

y3:=tempoy[3];
z3:=tempoz[3];
A:=y1*(z2-z3) + y2*(z3-z1) +y3*(z1-z2);
B:=z1*(x2-x3) + z2*(x3-x1) +z3*(x1-x2);
C:=x1*(y2-y3) + x2*(y3-y1) +x3*(y1-y2);
D:=-x1*(y2*z3-y3*z2)      -      x2*(y3*z1-y1*z3)      -
x3*(y1*z2-y2*z1);
mar[mar_index,1]:=A;
mar[mar_index,2]:=B;
mar[mar_index,3]:=C;
mar[mar_index,4]:=D;
mar-index :=mar_index+1;
end;

```

```

procedure ecuacion_del_plano;

```

```

const

```

```

    equis=40;

```

```

    ye=25;

```

```

    zeta=40;

```

```

var

```

```

    resultado:real;

```

```

    int,platini,michel:integer;

```

```

    car_oculta:array[1..num_max_pol] of integer;

```

```

begin

```

```

    int:=1;

```

```

    for i:=1 to mar-index-2 do

```

```

        car_oculta[i]:=0;

```

```
for i:=1 to mar-index-2 do
begin
  resultado:=   marte[i,1]*equis   +   marte[i,2]*ye   +
marte[i,3]*zeta + marte[i,4];
  if resultado>0 then
    vector-unitario[tnt,i]:=1;
end;

  platini :=find_max(suma_s,smax-1);
vector-unitario[tnt,platini]:=0;
  michel :=cara_opuesta(platini);
  vector-unitario[tnt,michel]:=1;

endd;
begin
end.
```

```

unit dos-d;
interface
uses tablas, lista, printer, crt;

type
  vector_de_enteros=array[1..50] of integer;
  vector_de_enteros_dobles=array[1..num_max_tab,1..25] of
integer;
var
vector_x,vector_y,vec_x,vec_y:vector_de_enteros_dobles;
gm,gd:integer;

procedure                transforma-pares-coordenados
(marca,tnt:integer;      xx,yy,zz:tres_por_tres;var
vect_x,vect_y : vector-de-enteros-dobles);

implementation

  vect_x,vect_y:vector_de_enteros);}

procedure  transforma-pares-coordenados;
const
profundidad=1;
  var
  x1,y1:real;
  k:integer;
begin

```

```
for k:=1 to marca do
begin
  x1:= xx[tnt,k] + zz[tnt,k] * profundidad * 0.71;
  y1:= yy[tnt,k] + zz[tnt,k] * profundidad * 0.71;
  x1:=(x1+40)*8;
  x1:=abs(x1);
  if (y1<=0) then
    begin
      y1:=abs(y1);
      y1:=(y1+12.5)*19;
    end
    else
      y1:=(abs(y1-12.5))*19;
  vect_x[tnt,k]:=round(x1);
  vect_y[tnt,k]:=round(y1);
end;
end;

begin
end.
```

```

unit motion;

interface

uses tablas, lista;

type

    matrix= array[1..4,1..4] of real;

var

    t,m:matrix;

    i,j:integer;

procedure identidad(var m:matrix);

procedure combine-transformaciones(var t,m:matrix);

procedure escala(sx,sy,sz:real;xf,yf,zf:real);

procedure traslada(tx,ty,tz:real);

procedure transforma_puntos(signal:integer; var
    a_x,a_y,a_z:tres_por_tres;mac:integer);

procedure transforma_puntos_y(signal:integer; var
    a_x,a_y,a_z:tres_por_tres;mac:integer);

procedure rota_x(a:real);

procedure rota_y(a:real);

procedure rota_z(a:real);

procedure rota-con-eje-arbitrario (xx1,yy1,zz1, xx2,
    yy2, zz2, angle:real);

implementation

procedure transforma-puntos;

var

    k:integer;

    tempx:real;

```

```

begin
  for k:=1 to signal do
    begin
      tempx:=a_x[mac,k] * t[1,1] + a_y[mac,k] *t[2,1] +
        t[3,1]* a_z[mac,k] + t[4,1];
      a-y[mac,k]:=a_x[mac,k]*t[1,2] + a_y[mac,k]*t[2,2] +
        t[3,2]*a_z[mac,k] + t[4,2];
      a_x[mac,k]:=tempx;
      a_z[mac,k]:=a_x[mac,k]*t[1,3] + a_y[mac,k]*t[2,3] +
        t[3,3]*a_z[mac,k] + t[4,3];
    end;
  end;

procedure identidad(var m:matrix);
begin
  for i:=1 to 4 do
    for j:=1 to 4 do
      if i=j then m[i,j]:=1
      else m[i,j]:=0;
    end;
  end;

procedure combine-transformaciones;
var
  temp:matrix;
begin
  for i:=1 to 4 do
    for j:=1 to 4 do
      temp[i,j]:=t[i,1]*m[1,j] + t[i,2] *m[2,j] +t[i,3]

```

```

        *m[3,j] +t[i,4]*m[4,j];
    for i:=1 to 4 do
        for j:=1 to 4 do
            t[i,j]:=temp[i,j];
        end;
    end;

procedure escala;
begin
    identidad(m);
    m[1,1]:=sx;  m[2,2]:=sy;
    m[3,3]:=sz;  m[4,1]:=(1 - sx) *xf;
    m[4,2]:=(1 - sy) *yf;  m[4,3]:=(1 - sz) *zf;
    combine-transformaciones(t,m);
end;

procedure traslada;
begin
    identidad(m);
    m[4,1]:=tx; m[4,2]:=ty; m[4,3]:=tz;
    combine-transformaciones(t,m);
end;

procedure rota-x;
var
    ca,sa:real;
function radian_equivalent(a:real) :real;
begin
    radian_equivalente:=a*3.1416 /180

```

```

end;
begin
    identidad(m);
    a:=radian_equivalente(a);
    ca:=cos(a);    sa:=sin(a);
    m[2,2]:=ca;   m[2,3]:=sa;
    m[3,2]:=-sa; m[3,3]:=ca;
    combine_transformations(t,m);
end;

    procedure rota_y;
var
    ca,sa:real;
function radian_equivalent(a:real) :real;
begin
    radian_equivalent:=a*3.1416 /180
end;
begin
    identidad(m);
    a:=radian_equivalent(a);
    ca:=cos(a); sa:=sin(a);
    m[1,1]:=ca;   m[3,1]:=sa;
    m[1,3]:=-sa; m[3,3]:=ca;
combine_transformations(t,m);
    end;

    procedure rota_z;
var

```



```

    ca,sa:real;
function radian_equivalent(a:real):real;
begin
    radian_equivalent:=a*3.1416 /180
end;
begin
    identidad(m);
    a:=radian_equivalent(a);
    ca:=cos(a); sa:=sin(a);
    m[1,1]:=ca; m[1,2]:=sa;
    m[2,1]:=-sa; m[2,2]:=ca;
    combine_transformations(t,m);
end;

procedure rotacion_con_eje_arbitrario;
var
a,b,c,d,length:real;
begin
    identidad(m);
    a:= xxz-xx1;  b:= yy2-yy1;  c:=zz2-zz1;
    length:= sqrt(a*a +b*b +c*c);
    a:= a/ length; b:=b / length;  c:=c /length;
    d := sqrt(b*b + c*c);
    traslada(-xx1,-yy1,-zz1);
    identidad(m);
    m[2,2]:= c/d;  m[2,3]:=b/d;
    m[3,2]:= -b/d;  m[3,3]:= c/d;

```

```
combine-transformaciones(t,m);
identidad(m);
m[1,1]:= d; m[1,3]:=a;
m[3,2]:= -a; m[3,3]:= d;
combine_transformaciones(t,m);
rota_z(angle);
identidad(m);
m[1,1]:=d; m[1,3]:= -a; m[3,1]:=a; m[3,3]:= d;
combine_transformaciones(t,m);
identidad(m);
m[2,2]:= c/d; m[2,3]:= -b/d;
m[3,2]:= b/d; m[3,3]:= c/d;
combine-transformaciones(t,m);
translada(xx1,yy1,zz1);
end; {rotacion_con_eje_arbitrario}
begin
end.
```

```

unit Imagen;
interface
uses dos_d,tablas, graph,crt,visible;

const
numpoints=4;

var
gd,gm:integer;

polypoints:array[1..numpoints] of pointtype;

procedure ini_modo_grafico;
procedure menu_inicial;
procedure menu_traslacion;
procedure menu_rotacion;
procedure traza-lineas (ap,tnt:integer; par_x,par_y:
vector-de-enteros-dobles);
procedure llenaqoligono (ap,tnt:integer; par_x,par_y:
vector-de-enteros-dobles; vector-unitario:
unidad;as:integer);
procedure clear;

implementation

procedure ini_modo_grafico;
begin
gd:=detect;
initgraph(gd,gm, '');
if graphresult <> grok then

```

```
halt(1);
end
procedure clear;
begin
    setViewport(150,5,600,440,true);
    clearViewport;
    SetViewport(0,0,getmaxx,getmaxy,true);
end;

procedure menu_inicial;
begin
repeat
    rectangle(0,0,GetmaxX,GetMaxY);
    OuttextXY(5,5,'GRAFICACION 3D');
    OuttextXY(5,415,'1-Traslacion    2-Rotacion    3-ClearScr
4-Salir');
    OutTextXY(5,430,'
    ');
until Keypressed;
end;

procedure menu_traslacion;
begin
    rectangle(0,0,GetmaxX,GetMaxY);
```

```
    OuttextXY(5,430 ,#24);
    OutTextXY(30,430,#25);
end;

procedure menu_rotacion;
begin
    rectangle(0,0,GetmaxX,GetMaxY);
    OuttextXY(5,430,#24);
    OuttextXY(20,430 ,#25);
    OutTextXY(35,430,#26);
    OutTextXY(50,430,#27);

end;

procedure traza-lineas;
var
    i1:integer;
begin
    i1:=1;
    repeat
        line(par_x[tnt,i1],par_y[tnt,i1],par_x[tnt,i1+1],par_y[t
nt,i1+1]);
        i1:=i1+2;
    until i1=ap+1;
end;

procedure fill_format(option:integer);
begin
```

```

case option of
2: setfillstyle(closedotfill,1);
   { setfillstyle(solidfill,1);}
3: setfillstyle(interleavefill,1);
1: setfillstyle(closedotfill,1);
end; end;

procedure llena_poligono
var
k,i,by,pik:integer;
begin
k:=0; pik:=1;
for i:=1 to as do
if vector_unitario[tnt,i]=1 then
begin
for by:=1 to numpoints do
begin
k:=k+1;
polypoints[by].x:=par_x[tnt,k];
polypoints[by].y:=par_y[tnt,k];
end;
fill_format(pik);
fillpoly(numpoints,polypoints);
pik:=pik+1;
end {end if} else k:=k+4;
end;
begin
end.

```

```

unit lista;
interface
const
num_max_pol=6;
num_max_tab=16;
num_max_vertices=25;
type
    nodo = ^celda;
    celda=record
        dato:real;
        ptr1,ptr2:nodo
    end;

    tres_puntos=array[1..num_max_pol] of real;
    tres_por_tres=array[1..16,1..num_max_vertices] of
real;
    pointer_array=array[1..50] of nodo;

var
    u,p,l,t,o,k:nodo;
    x:real;
    i,j:integer;
    fil_x,fil_y,fil_z:tres_por_tres;

function vacia(u:nodo):boolean;
procedure crea(var u:nodo);

```

```
procedure inserta(var p:nodo; x:real);
```

```
implementation
```

```
function vacia;
```

```
begin
```

```
    if u^.ptr2=nil
```

```
        then vacia:=true
```

```
        else vacia:=false
```

```
end;
```

```
procedure crea;
```

```
begin
```

```
    new(u);
```

```
    u^.ptr2:=nil;
```

```
        u^.ptr1:=nil;
```

```
end;{CREA}
```

```
procedure inserta;
```

```
var
```

```
    temp:nodo;
```

```
begin
```

```
    new(temp);
```

```
    temp^.dato:=x;
```

```
    temp^.ptr2:=p^.ptr2;
```

```
    p^.ptr2:=temp
```



```
end;{INSERTA}
```

```
begin
```

```
end.
```

```
unit uasm;
interface

  procedure enceraquerto;
  procedure arriba;
  procedure SUBE;
  procedure BAJA;
  procedure abajo;
  procedure izquierda;
  procedure derecha;
```

```
implementation
```

```
proecedure encera-puerto;
```

```
begin
```

```
  asm
```

```
    mov    dx,3BCh
```

```
    mov    al,11111111b
```

```
    out    dx,al
```

```
  end;
```

```
end;
```

```
procedure arriba;
```

```
begin
```

```
  asm
```

```
    mov    dx,3BCh
```

```
    mov    al,11111011b
```

```
    out    dx,al
```

```
end;
end;
procedure abajo;
begin
  asm
    mov    dx,3BCh
    mov    al,11110111b
    out   dx,al
  end;
end;
procedure izquierda;
begin
  asm
    mov    dx,3BCh
    mov    al,11101111b;
    out   dx,al
  end;
end;
procedure derecha;
begin
  asm
    mov    dx,3BCh
    mov    al,11011111b;
    out   dx,al
  end;
end;
end;
procedure sube;
```

```
begin
```

```
asm
```

```
    mov    dx,3BCh
```

```
    mov    al,11111101b;
```

```
    out   dx,al
```

```
end;
```

```
end;
```

```
procedure baja;
```

```
begin
```

```
asm
```

```
    mov    dx,3BCh
```

```
    mov    al,11111110b;
```

```
    out   dx,al
```

```
end;
```

```
end;
```

```
begin
```

```
end.
```

BIBLIOGRAFIA

1. Hearn, D., y M. Baker, "Gráficas por Computadora", 1988.
2. **Foley**, J. D., y A. Van Dam, "Fundamentals of Interactive Computer Graphics", 1984.
3. Weistock, N., "Computer Animation" , 1987.
4. Berger, M. , "**Graficación** por Computador", 1991.
5. Manuales **de Turbo** Pascal version 8.0, 1990.