UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES
ARAGÓN

"DISEÑO Y CONSTRUCCIÓN DE UN DIGITALIZADOR 3D"

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A:

LAURA URIBE RODRÍGUEZ

ASESOR DE TESIS:
M. EN C. MARCELO PÉREZ MEDEL

MÉXICO, 2005.

AGRADECIMIENTOS

Al Creador, por la vida, y porque al haber hecho a los seres humanos a su imagen y semejanza nos permite reflejar aunque sea débilmente sus cualidades, como el amor.

A mi familia:

A mis papás: Yolanda y Guillermo, por haber contribuido a darme la vida, por todo el amor, la paciencia y dedicación que me han tenido, todo lo bueno que pueda haber en mí se los debo (lo malo es mi responsabilidad). Son un gran ejemplo para mí en todos los sentidos, y son las personas que más quiero. Este trabajo les ha costado al igual que a mí preocupaciones y enojos, gracias por estar siempre conmigo y me gustaría que se sintieran satisfechos y partícipes de este logro.

A mis hermanos, por haberme acompañado toda su vida, teniéndome mucha paciencia y amor, dejándome aprender de ustedes. A Ricardo, por ser siempre colaborador y entusiasta, a Guillermo, por compartir sus experiencias de vida y por saber explicar todo con lujo de detalle, a Cristina por ser mi compañera de locuras.

A mi tía Catty por estar presente en todos mis momentos importantes, a mi tío Carlos por los buenos consejos; a la familia Jaramillo Rodríguez por sus incansables palabras de aliento y su confianza en que este proyecto vería la luz.

A mis maestros:

En primer lugar, a mi queridísimo profesor Marcelo Pérez, por su gran paciencia, por su fe en mí, por su dedicación a.sus tesistas, por sus ideas, por su comprensión y por preocuparse por el bienestar de medio mundo.

A Ernesto Peñaloza por sus buenos consejos, por su ánimo constante, por infundirme amor por esta carrera y aliento en los momentos difíciles. Nunca se me va a olvidar su valiosa ayuda en mi primer trabajo.

Al maestro Antonio Carichola por su ayuda con la materia de Filtrado y modulación.

A mis compañeros de trabajo y amigos:

Por supuesto a Jesús Vera, un gran compañero en la carrera y en este trabajo. Te agradezco que hayas puesto tu inteligencia (que es sobresaliente) al servicio de mi tesis, tu tiempo y tu esfuerzo, a pesar de que el 90% de la veces lo hayas hecho a regañadientes.

Al Ing.Benito Toledano por darme asilo en su oficina y ayudarme con todo su ingenio, recursos y buenos consejos para que me titulara.

A Ely Velázquez mi querida compañera del HCC, infinitas gracias por todo el tiempo que atendiste el trabajo solita, te aseguro que sin tu cooperación no habría terminado.

A Daniel San tiago por todo tu apoyo en el HCC, y por tus explicaciones siempre pacientes y claras.

A Sandra Diego, por preguntarme sobre mis avances, darme ánimos y ayudarme en cuanto te era posible, a Karla Ortega y a Alejandro Rodríguez por darme un gran impulso con su buen ejemplo en la tesis de licenciatura y maestría respectivamente, a Lupita Munive por todos tus sediciosos consejos, a Susy por tu apoyo en el trabajo de la Secretaría General .

Al Lic. Samuel Zepeda, Biól. Martha Torija y Lic. Celia Vargas por las facilidades en el trabajo para poder avanzar y terminar este proyecto.

A la Máxima Casa de Estudios, la UNAM, por formarme como profesional y darme la oportunidad de ser parte de su planta académica.

ÍNDICE

INTRODUCCIÓN

La transformación de imágenes reales, físicas y con precisión infinita en imágenes finitas y con una precisión determinada que pueda ser procesada por una computadora se logra desde hace ya varios años a través de un digitalizador de imagen bidimensional (2D). Existen diferentes tipos de digitalizadores 2D:

- Plano o de mesa. Tiene la fuente de luz y el sensor CCD[1] acoplados en un brazo móvil que se desliza sobre el documento que se encuentra inmóvil sobre una placa de vidrio, una variante de éste es el de libros de trayectoria aérea, el cual permite escanear volúmenes encuadernados con las hojas hacia arriba gracias a que la fuente de luz y el sensor CCD se encuentran ensamblados en un brazo de trayectoria aérea.
- Con alimentador de hojas. En este tipo el sensor y la fuente de luz permanecen fijos mientras que lo que se mueve es el documento, ayudado por un transporte de rodillos, cinta, tambor o de vacío.
- De tambor. En lugar de utilizar el sensor CCD utiliza un sistema de tubos fotomultiplicadores (PMT) en el bloque lector. Un sistema de transmisión fotomecánico recorre la imagen punto por punto, obteniéndose así una gran resolución y gama dinámica entre bajas y altas luces. Produce una imagen en colores primarios, pero ésta puede ser convertida en CMYK[2] mientras el lector recorre la imagen.
- De mano o portátil. La mayoría de estos modelos carecen de un motor para pasar las hojas, es el mismo usuario el que debe deslizar el escáner sobre el original. Suelen conectarse al puerto paralelo de la computadora y otros modelos llevan su propia tarjeta para bus ISA.

[1] Un foto-sensor CCD (Charge Coupled Device) es un dispositivo que recibe luz que es enviada desde una imagen a través de un juego de espejos y la convierte en señales eléctricas controladas por la intensidad y el color de la imagen, de la misma manera en que funciona un ojo.

[2] El espacio de color CMYK es una variación del modelo CMY. Éste agrega negro (Cyan, Magenta, Yellow, and blacK). El espacio de color CMYK cierra la brecha entre la teoría y la práctica. En teoría el componente negro no es necesario. No obstante, la experiencia con varios tipos de tintas y papeles ha mostrado que cuando son mezclados cyan, magenta y amarillo en igual proporción el resultado es usualmente un café oscuro, no negro. Agregando tinta negra en la mezcla se soluciona este problema.

Sin embargo, el avance de la tecnología se ha dirigido también hacia la digitalización de imágenes tridimensionales (3D), es decir, poder producir un modelo de un objeto real en tercera dimensión que incluso pueda manipularse con un software especialmente hecho para trabajar con modelos de este tipo como 3D Studio.

Probablemente algún día exista la misma variedad de digitalizadores 3D que hay en el caso de los de 2D, cubriendo distintas necesidades con un costo accesible al público en general.

Actualmente está a la venta un escáner compacto 3D que permite digitalizar superficies a través de un sistema "táctil" (piezo eléctrico). Este sensor puede explorar los datos con una precisión del grosor de un cabello. Puesto que se trata de un escáner de tipo contacto, puede explorar vidrio y otros materiales transparentes, además de objetos blandos como pueden ser: elementos de barro, cera, alimentos, etc. Tiene un volumen máximo de exploración de 152.4 mm (X) x 101.6 mm (Y) x 60.5 (Z) mm y el peso máximo que soporta es de 500 mg. Pueden seleccionarse pasos de exploración hasta de 0.05 mm. Tiene un costo de unos 1540 euros .(Figura I.1)

Figura I.1 Picza Pix 4

Además de escanear por contacto se han utilizado otros métodos, por ejemplo inferir las formas tridimensionales a partir de la deformación de la sombra de una varilla que va rodando por encima del plano en el que el objeto está situado. Otro método involucra la proyección de rayas en blanco y negro alternadas sobre el objeto. Se obtienen nubes de puntos que son convertidas en mallas poligonales o en parches de splines. Otra alternativa es aquella en que los cuerpos modelados son representados por un árbol de operadores que dan una descripción lógica de la estructura del objeto. Los diseñadores pueden especificar, examinar y modificar el modelo.

En el presente trabajo se presenta otra opción para lograr digitalizar un objeto y crear su modelo correspondiente, pretendiendo hacerlo de una manera sencilla y con un costo menor.

CAPÍTULO 1. ANTECEDENTES DE MODELOS 3D

1.1 CONCEPTOS BÁSICOS

Píxel. Este nombre resulta de la contracción de las palabras inglesas *picture element.* Se define como la unidad mínima de un gráfico, sus propiedades son posición y color, los cuales pueden ser direccionados mediante el uso de un sistema de coordenadas cartesianas y un modelo de color respectivamente.

Aspect Ratio o Razón de Aspecto. Es la razón entre el ancho y el alto en pantalla. De esta manera, 4:3 significa que el tamaño horizontal es un tercio mayor que el tamaño vertical. Por ejemplo, el *aspect ratio* de la televisión estándar es de 4:3 (ó 1.33:1). La pantalla de cine panorámica tiene un *aspect ratio* de 2.35:1, esto significa que la pantalla es 2.35 veces más ancha que alta. Otra razón de aspecto habitual de cine es 1.85:1.

Mapa de bits. Es un arreglo[3] rectangular de pixeles que representan una imagen. Existen varios formatos para almacenar estos tipos de imágenes entre los que se destacan: .bmp, .gif, .jpg, .tga.

Paleta de colores. Es un conjunto de colores tomados del universo de colores (16.7 millones) y se seleccionan de acuerdo a la conveniencia del usuario para el dibujo o imagen a representar; a cada uno se le asigna un índice y éste se utiliza para invocar el color correspondiente.

[3] En computación un arreglo es una estructura de datos que permite almacenar información del mismo tipo; cada dato individual se accesa mediante un índice. El arreglo puede ser de 1 a n dimensiones.

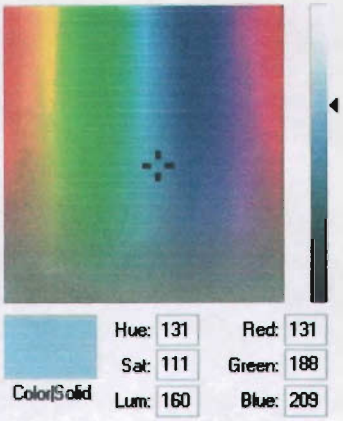


Figura1.1 Paleta de colores

Primitivas gráficas. Formas geométricas a partir de las cuales se pueden construir todas las demás, son: punto, línea, arco, texto. (Fig. 1.2)

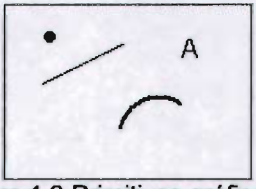


Figura 1.2 Primitivas gráficas

Profundidad del color. Es el número de colores posibles para cada píxel. La fórmula para calcular el número de colores es: $N_c = 2^{Nb}$ donde N_c es el número de colores y N_b el número de bits.

Número de bits por pixel	Exponente	Número de colores
1	2^1	2
2	2^2	4
4	2^4	16
8	2^8	256
24	2^{24}	16.7 millones

Tabla 1.1 Profundidad de color

Resolución. Cantidad de pixeles por unidad de área. A mayor resolución, mayor calidad de imagen. (Figuras 1.3 a 1.5). En las computadoras personales (PC) podemos encontrar típicamente cualquiera de las siguientes resoluciones:

Monitor	Resolución	Color
CGA	320X200	4
MCGA	320X200	256
VGA	640X480	16
SVGA	800X600	256
	1024X768	65536
	1200X1024	16.7 millones

Tabla 1.2 Resoluciones típicas en PC

Figura 1.3 Resolución 20 X 20

Figura 1.4 Resolución 70 x 70

Figura 1.5 Resolución 200 X 200

Conocer la resolución y la profundidad del color nos permite calcular el espacio requerido para almacenar una imagen. Por ejemplo, una profundidad de color de 4 requiere de 2 bits por píxel, y con una resolución de 320 X 200 tenemos que:

320 X 200 = 64 000 pixeles
X 2 bits/píxel
128 000 bits requeridos para almacenar la imagen

Conversión a bytes:

128 000 / 8 = 16 000 bytes

Para una profundidad de color de 16.7 millones ser requieren 24 bits, y una resolución de 1200 X 1024 tenemos que:

1200 X 1024 = 1 228 800 pixeles
X 24bits/píxel
29 491 200 bits

Conversión a bytes:

29 491 200 / 8 = 3 686 400 bytes

3 686 400 /1024 = 3600 KB requeridos para almacenar la imagen

Modelo RGB. Este modelo de color emplea síntesis aditiva, es decir, es la suma escalar de colores para obtener nuevos. El color de inicio es el negro, y la suma de todos los colores es el blanco. La combinación de los colores primarios (rojo, verde, azul) genera colores secundarios. Los monitores y las televisiones usan este modelo.(Figura 1.6)

Figura 1.6 Modelo de color RGB

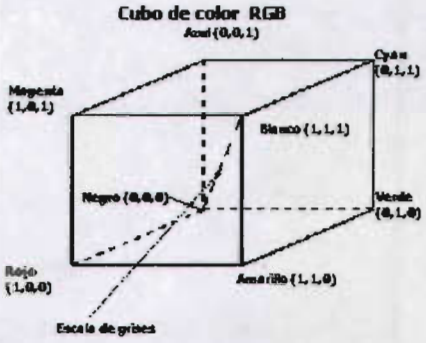


Figura 1.7 Cubo de color RGB

Cubo de color RGB. La diagonal representa los niveles de grises. El negro es (0,0,0) y el blanco es (1,1,1). en los vértices contiguos a el negro están los colores rojo, verde y

azul, y en los siguientes las combinaciones de estos colores entre sí. Todos los colores totalmente saturados se encuentran en la superficie del cubo. (Figura 1.7)

Modelo CMY. Modelo de color que emplea síntesis sustractiva, es decir, resta colores para obtener nuevos. El color de inicio es el blanco, y la sustracción de todos los colores da negro. La combinación de los colores primarios (cyan, magenta, amarillo) da los colores secundarios. Los *plotters*[4], las impresoras de inyección de tinta y láser a color utilizan este modelo. (Figura 1.8)

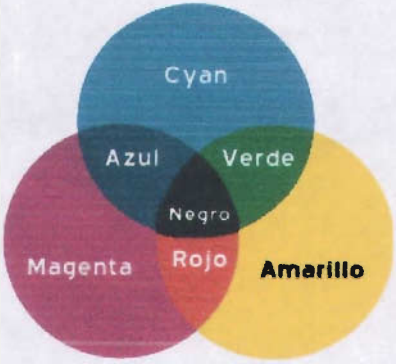


Figura 1.8 Modelo CMY

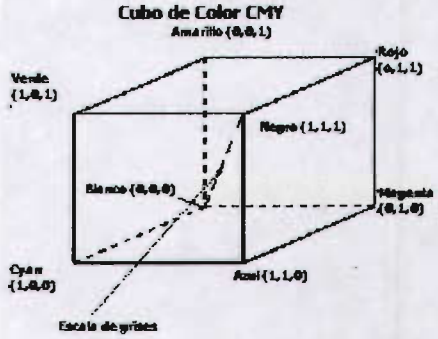


Figura 1.9 Cubo de Color CMY

Cubo de color CMY. La diagonal representa los niveles de grises. El blanco es (0,0,0) y el negro es (1,1,1). en los vértices contiguos a el negro están los colores rojo, verde y azul, y en los siguientes las combinaciones de estos colores entre sí. Todos los colores totalmente saturados se encuentran en la superficie del cubo. (Figura 1.9)

[4] Dispositivo de salida que usa una o más plumas que pueden ser levantadas, bajadas o movidas sobre un medio de impresión para dibujar gráficos o texto. Son usados principalmente en aplicaciones de ingeniería (planos, diseño)

1.2 VISUALIZACIÓN

Coordenadas polares. Por medio de un sistema de coordenadas en un plano, es posible localizar cualquier punto del plano. En el sistema polar, un punto se localiza especificando su posición relativa con respecto a una recta fija y a un punto fijo de esa recta. La recta fija se llama *eje polar*; el punto fijo se llama *polo*. Sea la recta horizontal OA el eje polar y el punto O el polo. Sea P un punto cualquiera en el plano coordenado. Tracemos el segmento OP y designemos su longitud por r. Llamemos θ al ángulo AOP. Evidentemente, la posición del punto P con relación al eje polar y al polo es determinada cuando se conocen r y θ. Estas dos cantidades se llaman coordenadas polares del punto P; en particular, r se llama radio vector y θ ángulo polar, ángulo vectorial o argumento de P. Las coordenadas polares de un punto se indican dentro de un paréntesis, escribiéndose primero el radio vector. Así las coordenadas de P se escriben (r, θ). La línea recta que pasa por el polo y es perpendicular al eje polar se llama el eje a 90° (Lehmann, 1965:237,238) (Fig. 1.10)

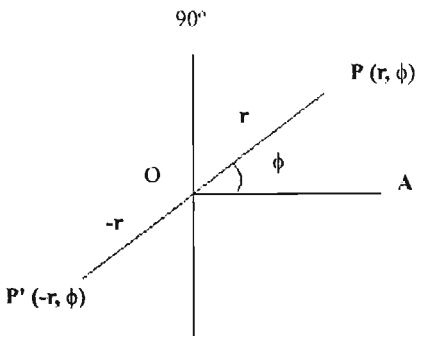


Figura 1.10 Coordenadas polares

Coordenadas cartesianas. Fue Descartes el primero que utilizó el método de las coordenadas para indicar la posición de un punto (en el plano o en el espacio), por eso se suele decir coordenadas cartesianas. Para explicarlo, tomaremos dos ejes X'X e Y'Y

cruzándose en ángulos rectos en un punto de origen común O. En cada eje escogemos una unidad conveniente de medición que puede o no ser la misma para ambos ejes. La línea horizontal X'X se llama el eje x, la línea vertical Y'Y se llama eje y. Las direcciones positivas se escogerán a la derecha para el primer eje y hacia arriba para el otro. Esta selección es arbitraria, pero es la usual. Tomando cualquier punto P, la distancia desde este punto hasta el eje x se llama la abscisa del punto y se señala con una x, su distancia desde el eje x se llama la ordenada y se señala con y. La abscisa y la ordenada juntas se llaman coordenadas del punto. El punto P cuya abscisa es x y cuya ordenada es y se designa por (x,y) o P(x,y), nombrándose siempre en primer lugar la abscisa. Obviamente, cuando un punto se encuentra a la derecha del eje y, su abcisa es positiva; cuando está a la izquierda su abscisa es negativa. Cuando un punto se encuentra arriba del eje x, su ordenada es positiva; si se encuentra debajo del eje x su ordenada es negativa. (Rider, 1940: 22,23) (Fig. 1.11)

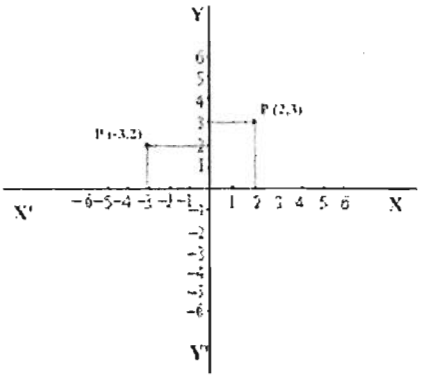


Figura 1.11 Coordenadas cartesianas

Coordenadas mundiales. Es cualquier sistema de coordenadas que el usuario encuentre conveniente para definir una imagen.

Ventana. Es un área especificada en coordenadas mundiales que se selecciona para desplegarla.

Puerto de vista. Área del dispositivo de salida en el que se coloca la ventana.

Tubería de vista. Es el conjunto de operaciones por las que tiene que pasar una imagen definida en coordenadas mundiales, para poder ser desplegada en un dispositivo de salida.

Transformaciones de modelado. Son las que se aplican en las definiciones de símbolos de las coordenadas de modelado para producir instancias[5] de los símbolos en coordenadas mundiales. Por lo general implican rotación, traslación y escalación para colocar un símbolo en coordenadas mundiales.

Transformaciones de vista. Es el mapeo de una parte de la escena de coordenadas mundiales a coordenadas de dispositivo.

Perspectiva. Para poder dibujar gráficos en la pantalla es necesario determinar como representar en dos dimensiones imágenes tridimensionales. Para ello se proyectan los puntos que forman la imagen sobre el plano, estableciendo un homomorfismo entre R3 y R2..

$$\pi: R^3 \rightarrow R^2$$

$$\pi\,(x,y,z) = (x,y).$$

Como la pantalla es un espacio plano, las coordenadas obtenidas con la aplicación sirven directamente para representar el punto.

Perspectiva caballera. Se caracteriza por ser oblicua y cilíndrica, el punto figura o cuerpo volumétrico se proyecta en uno de los planos coordenados con líneas de visión paralelas y no perpendiculares a dicho plano. El eje saliente suele proyectarse con un ángulo de 135° con el eje x, y se le aplica un coeficiente de reducción para lograr que la representación gráfica del objeto transmita la sensación de verlo en sus proporciones reales. Los valores más empleados para el coeficiente de reducción son ½, $^2/_3$ y ¾,

[5] Copia o referencia del modelo original.

aunque puede utilizarse cualquier otra fracción menor a la unidad para no generar desproporciones en el dibujo. (Fig. 1.12)

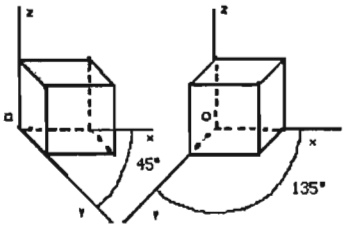


Figura 1.12 Perspectiva caballera

Perspectiva axonométrica. Se caracteriza porque las líneas de visión son ortogonales al plano de proyección. Con las proyecciones centrales y ortogonales se pueden conseguir diversas vistas del objeto desde diferentes posiciones, variando los ángulos que forman entre sí los ejes proyectados. (Fig. 1.13)

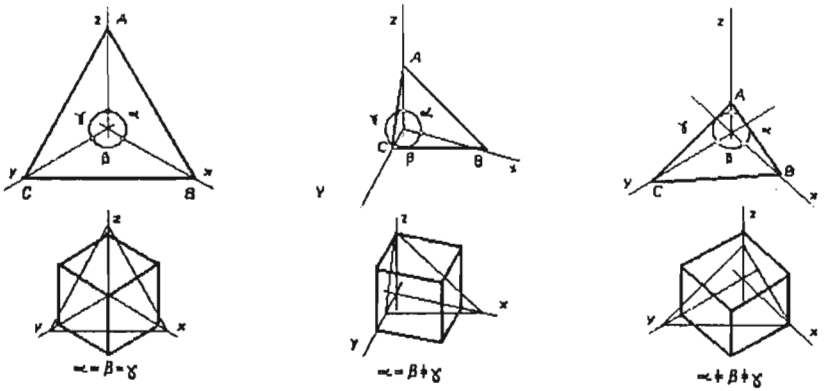


Figura 1.13 Perspectiva axonométrica

Perspectiva cónica. Se caracteriza por usar proyecciones cilíndricas, en donde las líneas de visión son paralelas y se supone al observador en un punto al infinito. En esta perspectiva los dibujos se parecen más a como los vemos, ya que las distancias se acortan con la profundidad y las paralelas dejan de serlo.

Proyección. Una vez que se convierten las descripciones de los objetos en una escena de coordenadas mundiales a coordenadas de vista , se pueden proyectar los objetos tridimensionales en el plano de visión bidimensional, para lo cual existen dos métodos básicos: la proyección paralela, en el que se transforman las posiciones de coordenadas en el plano de visión a lo largo de líneas paralelas, y la proyección en perspectiva, en la que se transforman las posiciones de los objetos en el plano de visión a lo largo de líneas convergentes en un punto llamado punto de referencia de proyección o centro de proyección. Una proyección paralela conserva las proporciones de los objetos, obteniendo vistas exactas; sin embargo no son representaciones realistas del objeto tridimensional, para lo cual es útil la proyección en perspectiva, si bien esta no conserva las proporciones relativas pues hace más pequeños los objetos más distantes.

1.3 TRANSFORMACIONES 3D

Traslación. Se aplica a un objeto para cambiar su posición a lo largo de la trayectoria de una línea recta de una dirección de coordenadas a otra. Se traslada un punto de la posición P = (x, y, z) a la posición P'= (x', y', z') con la operación de matriz

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

o

$$P' = T \bullet P$$

La traslación es una transformación de cuerpo rígido que mueve objetos sin deformarlos. (Figura 1.15) traslada cada punto del objeto la misma distancia, y luego se vuelven a unir entre sí. Se utilizan métodos similares para trasladar objetos curvos.

Escalación o escalamiento. Altera el tamaño de un objeto (Fig. 1.16). Dado un punto P (x, y, z) la expresión matricial para la transformación se puede describir como:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

o

$$P' = S \bullet P$$

Rotación. Se aplica una rotación en un objeto al cambiar su posición a lo largo de la de los ejes x y o z (Fig. 1.17). Para generar una rotación, especificamos un ángulo de rotación θ y la posición (x, y, z) del punto de rotación (o punto pivote) en torno al cual se gira el objeto. Las ecuaciones varían según el eje sobre el cual se quiera rotar, son:

Sobre el eje z:

$x' = x \cos\theta - y \operatorname{sen}\theta$

$y' = x \operatorname{sen}\theta + y \cos\theta$

$z' = z$

Sobre el eje y:

$x' = z \operatorname{sen}\theta + x \cos\theta$

$y' = y$

$z' = z \cos\theta - x \operatorname{sen}\theta$

Sobre el eje x:

$x' = x$

$y' = y \cos\theta - z \operatorname{sen}\theta$

$z' = y \operatorname{sen}\theta + z \cos\theta$

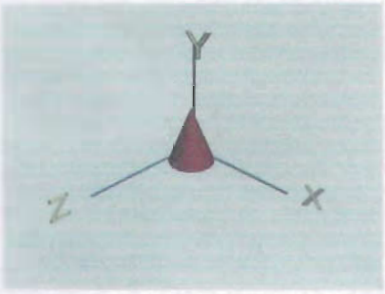


Figura 1.14 Objeto original

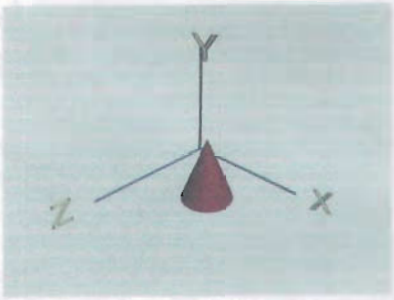


Figura 1.15 Traslación.

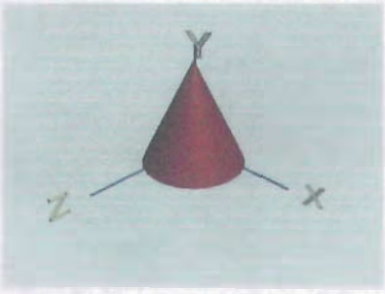


Figura 1.16 Escalación

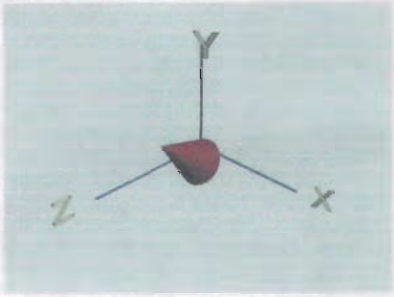


Figura 1.17 Rotación.

1.4 MODELADO DE SUPERFICIES

Modelado con mallas. Consiste en dividir en triángulos las superficies esto ofrece la ventaja de tener mayor facilidad para modificar o editar figuras, sin embargo ocupa mucho espacio.

Curvas de Bézier. Método para modelar superficies creado por el ingeniero francés Pierre Bézier para ser utilizado en el diseño de carrocerías de los automóviles Renault (Hearn, 1995 :342) Primero la curva se calcula en coordenadas polares, para el paso a tres dimensiones se añade una ecuación, para la coordenada z, idéntica a las de la x y y.

La curva se calcula por una aproximación con base en una serie de puntos de control, lográndose así curvas más suaves que si se calculara por interpolación. Bézier calcula

las coordenadas de un punto de la curva a partir del peso que los puntos de control que tienen sobre él. Ese peso viene dado por la distancia al punto de control y una distribución binomial ordinaria. El punto se obtiene sumando esos pesos.

Por tanto, dados n+1 puntos de control, P_0, P_1,...,P_n, por sus coordenadas cartesianas (xi,yi), el polinomio de Bézier

$$P(\Theta)=\sum_{i=0}^{n} P_i B_{i,n}(\Theta)$$

Donde

$$B_{i,n}(\theta) = \binom{n}{i} \theta^i \cdot (1-\theta)^{n-i}$$

En componentes:

$$x(\Theta)=\sum_{i=0}^{n} x_i B_{i,n}(\Theta) \qquad y(\Theta)=\sum_{i=0}^{n} y_i B_{i,n}(\Theta)$$

El número de vértices n+1 determina el orden del polinomio. Para aumentar el nivel de detalle debe aumentarse el orden del polinomio aproximado. Todos los puntos de la curva están afectados por todos de control, así que la variación de un solo punto de control hace que toda la curva se modifique. [Escribano, 1995:26,27)

Propiedades de las curvas de Bézier.

- Pasa por el primer y último punto de control
- Cada punto de control ejerce atracción especial por el tramo de la curva más próximo a él.
- Pueden ser cerradas y cortarse a sí mismas
- Al alterarse la abscisa de un punto de control, la curva experimentará un estiramiento de abscisas, sin modificarse las ordenadas, siendo esto válido también para éstas.

- Al aumentar la multiplicidad de un punto de control, la curva tiende a aproximarse más a él. (Escribano,1995:28)

Curvas B-Spline. En éstas el grado del polinomio se puede establecer de manera independiente de la cantidad de puntos de control y permiten un control local sobre la forma de una curva. La desventaja es que son más complejas que las Bézier.
La expresión general para el cálculo para las coordenadas de los puntos de la curva es:

$$P(\Theta)=\sum_{k=0}^{n} P_k N_{k,t}(\Theta)$$

donde P_k es un conjunto de entrada de n+1 puntos de control. El grado de los polinomios es $N_{k,t}(\theta)$ es t-1. Esto significa que el grado de la curva resultante no depende de los puntos de control sino estará definido de forma independiente. Es común emplear t = 4, obteniendo polinomios de tercer grado, las splines cúbicas.

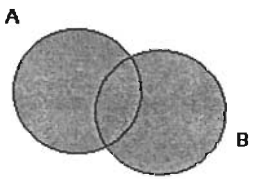
La multiplicidad de puntos de control, como en el caso de Bézier, acerca la curva a ese punto, pero tiene además otras consecuencias. Un vértice de multiplicidad 2 hace que la curva en él sólo presente continuidad de primer orden, perdiéndose suavidad, y una multiplicidad de 3 hace que se dé sólo continuidad de orden 0 (de posición) produciéndose vértices. La flexibilidad de control es una de las razones de la difusión de las B-splines.

1.5 MODELADO DE SÓLIDOS

Instanciación de primitivas. Todo modelador de sólidos debe contener una librería de primitivas sólidas, es decir un conjunto de objetos definidos por el programa que puede usarse con sólo especificarse las dimensiones de la primitiva.

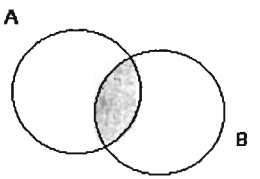
Operaciones booleanas. Es un método para combinar objetos y obtener otros. Los operadores pueden ser unión, diferencia e intersección. Por medio de estos se pueden

obtener sólidos complicados y difíciles de representar por otros medios. Para ilustrarlo con conjuntos, unión es lo que está en A y está en B (Figura 1.18), intersección es lo que está en dos conjuntos dados a la vez (Figura 1.19) y diferencia es sustraer un conjunto de otro. (Figura 1.20).



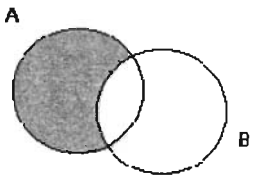
A U B

Figura 1.18 Unión



A ∩ B

Figura 1.19 Intersección



A - B = A - A ∩ B

Figura 1.20 Diferencia

Modelo de mallas poligonales. Es en el que se presenta al sólido por un conjunto de líneas, típicamente las aristas del objeto. En la malla debe almacenarse una lista de vértices y de polígonos. Cada polígono almacena los índices de los vértices que lo conforman. Las desventajas de este método son: que presenta un bajo nivel de detalle , ocupa mucho espacio, desconocimiento del volumen real del objeto, incapacidad para reconocer perfiles curveados, dificultades para el cálculo de propiedades físicas, incapacidad para la aplicación de modelos de iluminación y sombreado de imagen y sus ventajas son: facilidad para modelar, más eficaz para datos de ingeniería y ciencia, posibilidad de aprovechar las tarjetas aceleradoras de gráficos, que están optimizadas para generar triángulo. Por ejemplo, para la creación de la malla para un cilindro, el primero paso es considerar a un cilindro como una sucesión de circunferencias sobre una trayectoria lineal. Después se calcula la altura de las circunferencias, considerando que la altura del cilindro es 'h' y que la altura de las circunferencias variará de 0 a 'h' a través de Z, obteniéndose la siguiente fórmula:

$$Z_i = h/(n\_niveles-1)*n$$

Donde 'n_niveles' es el número de circunferencias a lo largo de la altura del cilindro y n variará de 0 a 'n_niveles'-1. Esto nos proporciona los puntos y los polígonos de la pared

del cilindro. Dado lo anterior, se pueden crear polígonos tomando cada punto p de una circunferencia del nivel n y se une con el punto p+1 del mismo nivel y con el p de la circunferencia del nivel n+1, formando un triángulo, de esta forma cada punto es tocado por dos triángulos que son: (p, n), (p+1, n) y (p, n+1), (p+1, n), (p, n+1) y (p+1, n+1). (Figura 1.21). A continuación se crean las tapas polares, para las que se calculan los puntos máximos (0,0,radio) y mínimo (0,0,-radio) y se unen con líneas a los vértices del primer y último nivel. Los triángulos están formados por la unión de P_n, P_{n+1} y P_{polar} (Figura 1:22) (Pérez, 2002:67-70). El resultado final es (Figura 1.23):

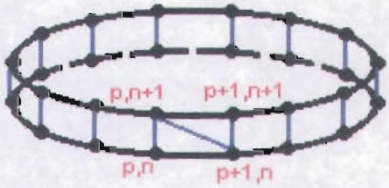


Fig. 1.21 Polígonos

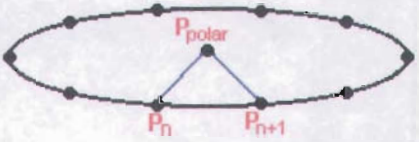


Fig. 1.22 Tapas polares

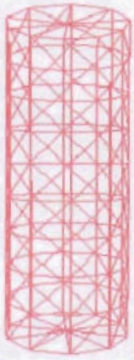


Fig 1.23 Malla completa

Modelo de esquemas o alámbrico. Es en el que se presenta al sólido por un conjunto de líneas, típicamente las aristas del objeto. Es considerado en muchos casos como obsoleto, por las siguientes razones: ambigüedad en la representación en pantalla, desconocimiento del volumen real del objeto, incapacidad para reconocer perfiles curveados, dificultades para el cálculo de propiedades físicas, incapacidad para la aplicación de modelos de iluminación y sombreado de imagen. (Escribano, 1995:39)

Modelo B-rep. B-rep viene de la expresión *boundary representation*, es decir, representación de fronteras. En este se definen los objetos en función de la superficie frontera: vértices, aristas y caras.

Los B-reps primordialmente tratan con poliedros, pues en algunos casos restringen a que las caras sean sólo triángulos y polígonos convexos; en estos sistemas las superficies curvas y formas libres se poligonalizan, lo que lleva a una pérdida de resolución y el aumento en las necesidades de memoria para el almacenamiento.

Lo primero que debe hacerse es comprobar que la frontera que el usuario introduce corresponde a un sólido, esta suele hacerse con la fórmula de Euler, que describe la relación invariante entre el número de vértices, caras y aristas de un poliedro.

Aunque esa verificación no es suficiente en sí misma, por ello Baumgart introdujo los operadores de Euler. Esta herramienta se aplica a los poliedros y se obtienen poliedros de Euler. Esto facilita la tarea al usuario pues se le pueden ofrecer las primitivas y las herramientas para modificarlas, añadiendo y eliminando vértices, aristas y caras.

Los B-reps son los más precisos para la representación interna de un sólido, permiten el acceso fácil y rápido a la información del modelo en las últimas fases de diseño, sin embargo presenta dificultad para representarlo en pantalla, no es fácil de utilizar para el usuario, ni intuitivo y es complejo de implementar. (Escribano, 1995:40,41)

Modelado C-Rep. Su nombre viene de representación constructiva y es más conocido como CGS (Geometría Sólida Constructiva).Es uno de los métodos más usados para la implementación de modeladores de sólidos. Ofrece primitivas sólidas que se combinan con operadores booleanos. Un modelador CGS almacena la información en un árbol binario, donde los nodos son las operaciones binarias y las hojas, las primitivas (Figura 1.24). Estos árboles son fáciles de representar en pantalla, y al tener la información completa sobre la masa del objeto, es apropiado para el cálculo de propiedades físicas,

sin embargo es lento para las fases finales del proceso de diseño y facturación pues, para conocer el contorno exacto del sólido, debe resolver cada vez el árbol de operaciones booleanas.

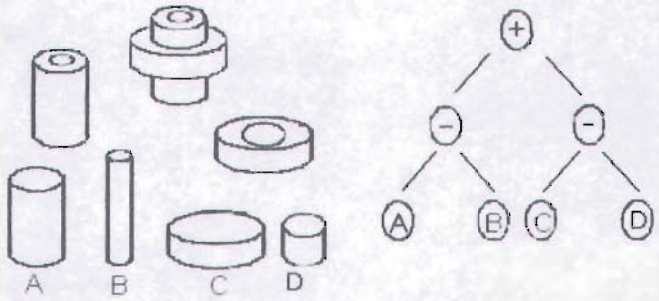


Figura 1.24 Modelado CGS

Barridos. Consiste en recorrer una región con una superficie plana; el barrido puede ser traslacional con dirección perpendicular al plano de la superficie, o rotacional, entre otros. Esto permite generar volúmenes difíciles de lograr con operaciones booleanas. Un ejemplo sencillo de barrido traslacional lo tenemos al trasladar un rectángulo en el eje de las x, formando un paralelepípedo. (Figuras 1.25 y 1.25). En la figura 1.27 se ilustra el barrido rotacional.

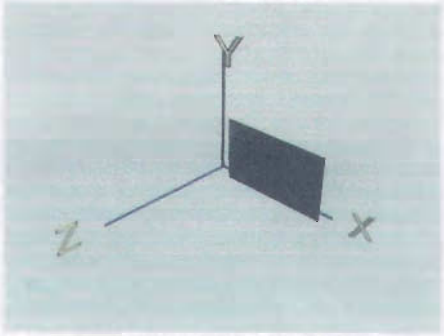


Figura 1.25 Rectángulo.

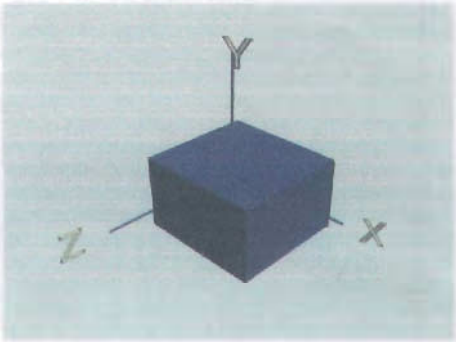


Figura 1.26 Paralelepípedo que resulta del barrido traslacional del rectángulo

Figura 1.27 Barrido rotacional

Modelado de sólidos con campos escalares. Las primitivas de modelado basadas en campos escalares interaccionan deformándose al estar cerca unas de otras, logrando formas complejas difíciles de obtener en otros métodos. Para ilustrarlo, al aproximar dos esferas, se obtiene una forma parecida a un cacahuate, diferente lo que resultaría de una unión booleana simple (Figuras 1.28 a 1.30) Es especialmente útil para modelar formas orgánicas, por ejemplo, animales, rostros, flujos de fluidos y similares. Algunos métodos de modelado basados en campos escalares son Blobby model, Soft objects, Metaballs y Supermetaballs(Escribano, 1995:46,47) .

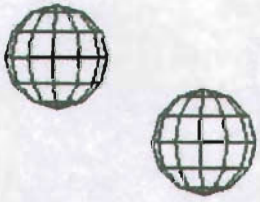


Figura 1.28 Esferas

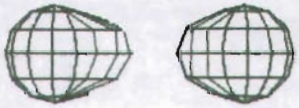


Figura 1.29 Al aproximarse empiezan a deformarse

Figura 1.30 Se unen formando un sólido.

CAPÍTULO 2. ADQUISICIÓN Y PROCESAMIENTO DE IMÁGENES

2.1 SISTEMAS DE PERCEPCIÓN DE IMÁGENES.

Un sistema de percepción de imágenes consta básicamente de:

1) Una fuente de iluminación. Esta puede ser interna o externa, dependiendo de si pertenece a un sistema pasivo o uno activo. En los primeros sistemas la fuente de iluminación es externa al sistema, por lo que no se tiene control sobre ella. Los activos cuentan con su propia fuente de iluminación.

2) Una escena. Es una sección del paisaje que se selecciona arbitrariamente con el sensor remoto y contiene un conjunto de objetos que interaccionan con la luz de la fuente, ya sea reflejándola, transmitiéndola o absorbiéndola.

3) Un sensor remoto, un dispositivo que captura la luz proveniente de la escena. Un ejemplo de un sensor remoto natural es el ojo, que deja pasar la luz a través de la córnea, y luego es regulada por el iris que actúa como un diafragma que se abre o cierra de acuerdo a la iluminación de la escena, el cristalino la dirige a la retina, en donde las células son detectores fotosensibles y convierten la luz en señales electroquímicas, éstas se mandan al cerebro por medio del nervio óptico, obteniéndose una imagen. Un sensor artificial sigue los mismos principios, sin embargo, éste puede almacenar de manera permanente la imagen y es capaz de percibir luz invisible para el ojo humano.

IMAGEN DIGITAL. Es un arreglo bidimensional de valores, donde cada valor está expresado como f(x,y) donde x,y son los índices del arreglo. Los elementos de este arreglo son pixeles.

FORMATOS DE IMAGEN

Una imagen puede almacenarse en un archivo siguiendo diferentes formatos. Siempre utiliza una cabecera que identifica el tipo de archivo y contiene información necesaria para saber el tamaño de la imagen o el número de colores. A continuación se encuentran los datos, por lo general comprimidos con el algoritmo correspondiente a ese formato, una vez descomprimidos los datos indican el color de cada pixel de la imagen. También se incluye una tabla que asocia a cada color las correspondientes cantidades de rojo, verde y azul, a la que se le conoce como paleta de colores. De manera que cada archivo gráfico tiene una cabecera, los datos de los pixeles, y la paleta de colores. Ha diferentes tipos de formatos: BMP, PPM, PCX, GIF, JPG. A continuación se da más información sobre cada uno de ellos.

BMP (Windows BitMaP). Es el formato de archivo más simple, pues prácticamente no usa compresión, consiste en una cabecera, y a continuación los datos de cada pixel, comenzando por la línea de más abajo y terminando con la superior, pixel por pixel de izquierda a derecha. Tiene como ventaja su sencillez, y como desventaja el gran tamaño del archivo que se genera

GIF. (Graphic Interchange Format) realizado por Compuserve. Utiliza el algoritmo de compresión LZW (Lempel-Ziv-Welch) modificado, mucho más complejo que el RLE (Run Length Encoded)[6]. Consiste en detectar las repeticiones de color y de ciertas secuencias, mediante un diccionario que se va construyendo, Además permite definir un color transparente, por lo que resulta útil para páginas Web.

[6] Está basado en sustituir la información gráfica de píxeles que se repiten por el valor del color de uno de ellos y la posición de cada uno de los puntos que lo utilizan.

JPG. Utiliza un complejo algoritmo de compresión con pérdida de información. Gracias a esto este formato es uno de los que más comprimen. Las modificaciones realizadas son inapreciables si se trata de una fotografía escaneada. Sin embargo, en el caso de un dibujo la diferencia es más perceptible. No permite definir colores como transparentes.

PPM (Portable Pix Map) Es sin compresión, utiliza el modelo RGB y se puede almacenar en modo texto.

VECINDAD. Es un pixel y sus vecinos. Al conjunto de 4 vecinos que comparten un vértice con el pixel se le conoce como ND. Los vecinos que comparten una arista (lado) con el pixel son vecinos N4. El conjunto que resulta de la unión de los conjuntos ND y N4 es conocido como N8. La distancia que existe del pixel a un vecino N4 es de 1, la distancia a un vecino ND es $\sqrt{2}$.

	ND_1	$N4_1$	ND_2	
	$N4_2$	**Pixel**	$N4_4$	
	ND_3	$N4_3$	ND_4	

Dado un pixel P (x,y), tenemos que:

Vecinos ND	Vecinos N4
ND_1 (x-1,y-1)	$N4_1$ (x,y-1)
ND_2 (x+1,y-1)	$N4_2$ (x-1,y)
ND_3 (x-1,y+1)	$N4_3$ (x,y+1)
ND_4 (x+1,y+1)	$N4_4$ (x+1,y)

2.3 PROCESAMIENTO DIGITAL DE IMÁGENES (PDI)

Se encarga del estudio de las metodologías para modificar y analizar imágenes por medio de una computadora digital. Algunas de las áreas del PDI son:

- Realce: Mejora la apariencia de la imagen para su despliegue, o bien resalta cierta información de interés para un posterior análisis.
- Restauración: Elimina o minimiza degradaciones en una imagen.
- Análisis: Produce una descripción de la imagen a partir de medidas cuantitativas. (Aguilar, 1994:1)

Elementos básicos de un sistema de PDI

1. Adquisición. Puede realizarse con cámaras de video, digitalizadores u otros. Cuando el Sistema de percepción de imágenes nos proporciona una imagen analógica en video, debe ser digitalizada con un convertidor A/D. Por lo general un digitalizador tiene 2 fases: muestreo y cuantización. En el muestreo se debe dividir la imagen en un número finito de elementos. El número de elementos depende de la cantidad de información que se quiera conservar. El tamaño de cada elemento define la resolución del sistema. A mayor resolución, mayor contenido informativo de la imagen. En la cuantización la información de cada elemento de la imagen se traduce en una señal eléctrica. El rango de la señal se divide en un número finito de pasos, distribuidos uniformemente, de modo que a cada elemento de la imagen le corresponde un valor entero. El número de niveles de gris está dado por 2^n donde n es el número de bits del convertidor. Entre más pasos existan en el rango mayor es la resolución de la cuantización.

2. Almacenamiento. Una imagen puede guardarse en la memoria de la computadora o en algún dispositivo magnético.

3. Despliegue. Por lo general se desplegará la imagen en el monitor o en la impresora. Para las pantallas de tubo de rayos catódicos (CRT) barre y excita a una pantalla de fósforo que produce puntos de luz proporcional al nivel de gris que corresponde a cada punto. En las pantallas planas de cristal líquido y de plasma sólo requieren de un barrido para desplegar la imagen, sin embargo el cambio de un pixel requiere barrer de nuevo la pantalla.

4. Procesamiento. Hay diversas técnicas, y herramientas para modificar una imagen, de acuerdo al objetivo que se tenga al procesarla es el método a seguir.

2.4 HISTOGRAMA

El histograma es una representación visual simple de la frecuencia con la que ocurren los valores de los tonos en una imagen. Es una función que muestra para cada valor de pixel D, el número de pixeles H(D) que tienen ese valor en la imagen. A H(D) también se le llama la frecuencia. Al graficar, D es el eje horizontal (abcisas) y H(D) es el eje vertical (ordenadas). A continuación se presenta un ejemplo de una imagen en tonos de grises (Figura 2.1)con su respectivo histograma (Figura 2.2):

Figura 2.1 Imagen original

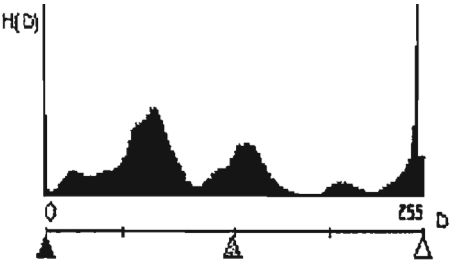


Figura 2.2 Histograma de 2.1

Figura 2.3 Imagen clara

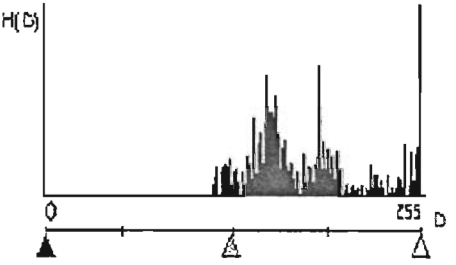


Figura 2.4 Histograma de 2.3

Figura 2.5 Imagen obscura con alto contraste

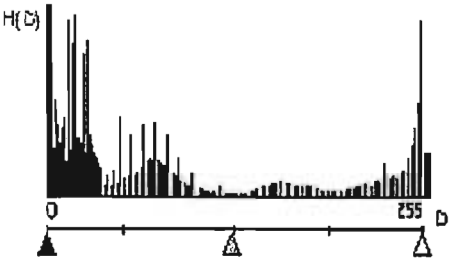


Figura 2.6 Histograma de 2.5

2.5 FILTROS

Un filtro es una operación que se realiza sobre los pixeles de la imagen para modificar su aspecto, por ejemplo, elimina el ruido, resalta los bordes, etc. Existen filtros en el dominio de la frecuencia y en el dominio espacial. Los primeros usan la transformada de Fourier, en el caso de los filtros de dominio espacial el valor de cada pixel de la imagen filtrada no sólo depende del pixel correspondiente de la imagen original, sino de sus vecinos. (Aguilar, 1994:25)

KERNEL. Es una matriz cuadrada de dimensión impar (generalmente de 3 X 3, o 5 X 5), que se coloca como máscara a través de toda la imagen.

2.5.1 FILTROS DE CONVOLUCIÓN.

Consiste en sustituir un pixel por la suma ponderada de sus vecinos. La orilla de la imagen no se convoluciona.

PASA-BAJA. Dejan intacto el contenido de baja frecuencia de la imagen y atenúan el de alta. Su principal utilidad es suavizar el ruido (Figura 2.7 y 2.8). Al aplicar los siguientes kerneles se obtiene un efecto pasa – baja:

$$1/9\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}\quad 1/16\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}\quad 1/8\begin{bmatrix}0&1&0\\1&4&1\\0&1&0\end{bmatrix}$$

PASA – ALTA. Algunos de éstos acentúan los detalles de alta frecuencia sin modificar los de baja. Las áreas planas no se afectan, sin embargo los cambios grandes de intensidad se amplifican (Figura 2.8 y 2.9). Los siguientes kerneles actúan como un filtro pasa –alta:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Figura 2.7 Filtro Pasa-baja

Figura 2.8 Imagen original

Figura 2.9 Filtro Pasa-alta

OPERADORES DIFERENCIALES. Se usan para realzar los bordes antes del proceso de obtención de características. Los coeficientes de los kerneles suman cero, eliminando las zonas uniformes de la imagen. Entre éstos se cuentan el operador Laplaciano y el Gradiente.

LAPLACIANO. Este operador genera una buena definición de bordes en cualquier dirección. Se resaltan las zonas con cambios marcados de intensidad. Es muy sensible al ruido, por lo que en algunas técnicas de segmentación se usa para determinar si un

pixel está del lado claro u obscuro de un borde(Figura 2.x y 2.x). El kernel aproximado para este operador es:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

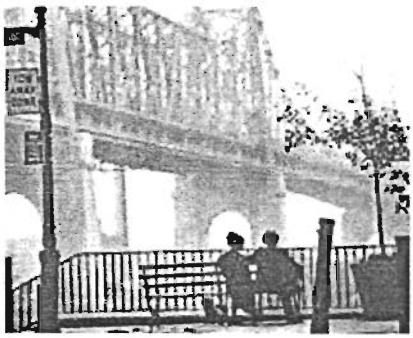


Figura 2.10 Imagen original

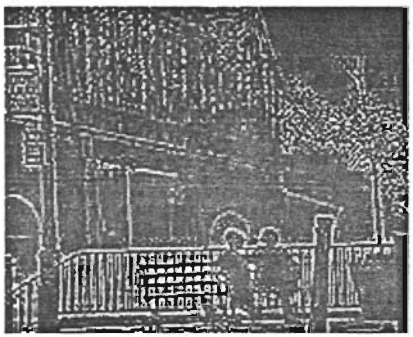


Figura 2.11 Filtro laplaciano aplicado a 2.10

GRADIENTE. Para este operador se requiere un kernel para la dirección horizontal y otro para la dirección vertical. Si se utiliza sólo uno de ellos se resaltan sólo los bordes verticales u horizontales según sea el caso. Algunos operadores gradiente más comunes son: Prewitt, Sobel (Figuras 2.12 a 2.15) e Isotrópico, a continuación se muestran los kerneles correspondientes (Tabla 2.1):

	Gx	Gy
Prewitt	$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$
Sobel	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$
Isotrópico	$\begin{bmatrix} -1 & \sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ \sqrt{2} & 0 & \sqrt{2} \\ -1 & 0 & 1 \end{bmatrix}$

Tabla 2.1 Operadores gradiente

Figura 2.12 Imagen original

Figura 2.13 Filtro Sobel para bordes horizontales

Figura 2.14 Filtro Sobel para bordes verticales

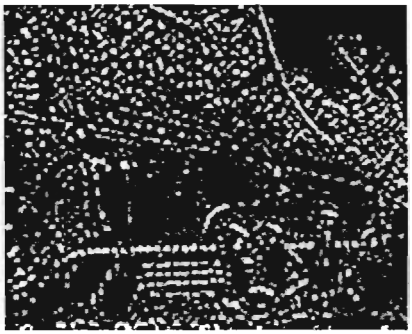


Figura 2.15 Gradiente

2.5.2 FILTRO MEDIANA

Consiste en ordenar de menor a mayor los valores de los pixeles de la vecindad tomando la mediana (o el valor de en medio). Así si un pixel tiene un valor disparado afecta menos a los demás, por lo que es útil para quitar el ruido, siempre y cuando el número de pixeles ruidosos no sea igual o mayor a la mitad del número total de pixeles, pues muy probablemente dejará pasar los pixeles ruidosos. A continuación se muestra un ejemplo (Figuras 2.16 y 2.17):

Figura 2.16 Imagen original

Figura 2.17 Filtro mediana aplicado a 2.16

2.6 CLASIFICACIÓN.

Esta fase incluye analizar la información de la imagen para poder para poder ordenar y determinar a que clase pertenecen los objetos que la componen. Este proceso consta de 3 partes: segmentación, extracción de características y asignación de patrones a una clase específica.

SEGMENTACIÓN. Consiste en separar un objeto del fondo y de otros objetos. Existe la segmentación por umbral y por fronteras. En la segmentación por umbral se encuentra a los pixeles que pertenecen al objeto, con ayuda del histograma se puede contar cuantos pixeles hay en un cierto rango, determinando el área bajo la curva. En la segmentación por frontera se hace un barrido de la imagen hasta encontrar un punto de otro color, se revisa a sus vecinos para determinar cual de ellos es frontera y se sigue el barrido a través de toda la frontera, guardando la posición de los pixeles, lo que permite dibujar el objeto segmentado.

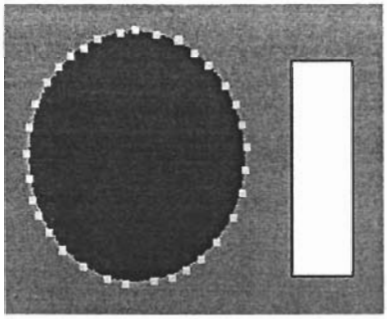


Figura 2.18 Imagen original

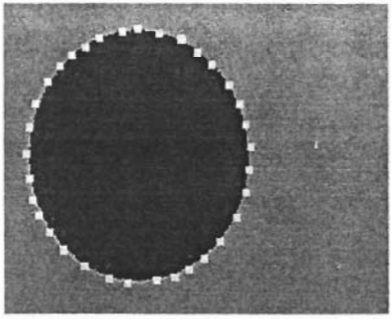


Figura 2.19 Objeto segmentado de 2.18

El grado de interacción del usuario en el proceso de segmentación se puede clasificar en: manual, semiautomática y automática.

Manual: El usuario realiza la segmentación el mismo con ayuda de una herramienta informática. En ésta se seleccionan manualmente las fronteras de regiones que se desea segmentar, es preciso pero muy lento y es impracticable cuando el número de imágenes es muy alto.

Semiautomática o interactiva: la computadora realiza el proceso, pero el usuario interviene en determinados momentos para definir parámetros o efectuar correcciones. Es el método empleado generalmente.

Automática: La computadora realiza todo el proceso.

EXTRACCIÓN DE CARACTERÍSTICAS. En este paso se estudian los parámetros medibles del objeto o patrón, obteniendo un conjunto de medidas llamado vector de características, entre estas pueden estar: largo, ancho, perímetro, área, rectangularidad y circularidad.

Rectangularidad: Nos indica que tan parecido a un rectángulo es el objeto. Es igual al área del objeto entre el área del rectángulo que lo encierra, el máximo valor de rectangularidad es 1.

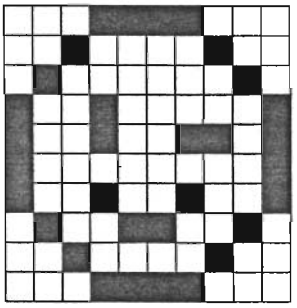
2.7 COMPRESIÓN

Es el proceso mediante el cual se reduce el tamaño eliminando las redundancias, existen 2 métodos: por pixeles adyacentes (RLE) y por patrones (LZH).

COMPRESIÓN POR PIXELES ADYACENTES

El método de pixeles adyacentes consiste en describir una imagen considerando la repetición consecutiva de pixeles del mismo color. Ejemplo:

Para comprimir esta imagen en blanco y negro tomamos el primer pixel de izquierda a derecha y de arriba abajo, vemos cuantas veces aparece y se anota el valor y el color, hasta encontrar un pixel de diferente color, y el proceso se repite hasta abarcar la totalidad de la imagen. Es necesario conocer las dimensiones originales de la imagen para su futura descompresión.



Por lo que la imagen anterior quedaría como:

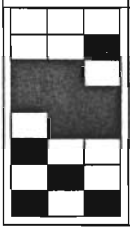
3B 4N 5B 1N 4B 1N 3B 1N 6B 1N 1B 1N 2B
1N 5B 2N 2B 1N 2B 2N 1B 2N 8B 2N 2B 1N
2B 1N 2B 1N 1B 1N 2B 2N 2B 1N 3B 1N 4B
1N 5B 4N 3B

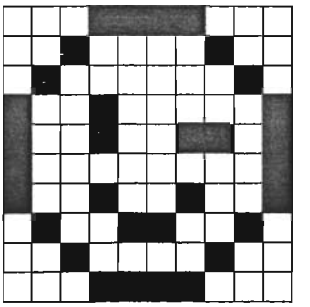
Considerando que para almacenar la cantidad de puntos junto con su color requeriríamos por lo menos 9 bits (8 para el tamaño y uno para el color), los datos de la imagen anterior requerirían 378 bits de espacio (42 X 9)

COMPRESIÓN POR PATRONES.

Debido a que hay imágenes cuya complejidad es tan grande que el método por pixeles adyacentes resulta ineficiente (donde no son muchos los puntos continuos del mismo color y por lo que los datos resultantes serían más que los de la imagen original), es necesario utilizar alternativas como la compresión por patrones la cual se basa en tomar un bloque de información y asignarle un código (cuyo tamaño obviamente es menor al tamaño del bloque). Ejemplo:

Primero elegimos el tamaño del bloque (que en este caso será de 3 pixeles), y generamos una tabla, que contendrá en una columna los diferentes bloques que encontremos y en otra los valores asignados.

Bloque	Código
	000 001 010 011 100 101 110 111



La imagen anterior quedaría como:

000	011	101	000	101	001	000	101	000	110	101	101
000	010	110	100	100	000	000	001	101	101	101	111
001	101	101	110	000	101	000	011	101			

Si utilizamos 48 bits para almacenar la tabla (3 bits para el bloque y 3 bits para su valor) la imagen anterior comprimida requeriría 141 bits para su almacenamiento, comparados con los 369 bits utilizando el método anterior, por lo que se evidencia que el método es más eficiente.

Referencias fotográficas:

Figura 2.1	www.woodrising.com/ gesg/book.htm (Sierra Gorda) D. Neil Bird. 1997 Copyright © Woodrising Consulting Inc.
Figura 2.8	www.jornada.unam.mx/1998/nov98/ 981128/naturaleza.jpg J. Guadalupe Pérez, Abraham Paredes
Figuras 2.10, 2.12	http://www.otrocampo.com/3/paisaje.html

CAPÍTULO 3. DISEÑO Y CONSTRUCCIÓN DEL EQUIPO

3.1 BASES TEÓRICAS.

MOTORES PASO A PASO.

Son motores que por cada pulso que reciben se mueven un paso que puede ser de 90°, o tan pequeño como 1.8° de manera que se requieren 4 o 200 pasos respectivamente para dar un giro de 360°, por ello, son ideales para sistemas que requieren movimientos precisos.

Están constituidos por un rotor, que en este caso es un imán permanente, y un estator compuesto de bobinas, toda excitación para las bobinas será externa y manejada por un controlador, al alimentar estas bobinas se atraerá el polo del rotor con respecto al polo generado por la bobina y éste permanecerá en esta posición atraído por el campo magnético de la bobina hasta que desaparezca el campo magnético y se active otra bobina haciendo avanzar o retroceder el rotor.

Los motores paso a paso de imán permanente pueden dividirse en 2 grupos: unipolares y bipolares. Los bipolares cuentan por lo general con 4 hilos de salida y para moverse se requiere invertir la corriente que circula en las bobinas en una secuencia determinada. Cada vez que se invierte la polaridad el eje se mueve un paso, La tabla 3.1 muestra la secuencia para mover motores paso a paso bipolares.

Paso	Terminales

	A	B	C	D
1	V+	V-	V+	V-
2	V+	V-	V-	V+
3	V-	V+	V-	V+
4	V-	V+	V+	V-

Tabla 3.1 Secuencia para mover motores de paso bipolares

Los unipolares tienen 5 o 6 hilos de salida y son más simples de controlar. Existen 3 tipos de secuencias para mover un motor de este tipo, en la primera, conocida como de paso simple, se activa una sola bobina a la vez, lo que causa que el torque de paso y la retención sea menor en comparación con las otras secuencias (Tabla 3.2).

Paso	Terminales			
	A	B	C	D
1	**On**	Off	Off	Off
2	Off	**On**	Off	Off
3	Off	Off	**On**	Off
4	Off	Off	Off	**On**

Tabla 3.2 Secuencia simple

La segunda secuencia, de paso doble, activa 2 bobinas en cada paso, es la más común y logra un alto torque de paso y retención (Tabla 3.3).

Paso	Terminales			
	A	B	C	D
1	**On**	**On**	Off	Off
2	Off	**On**	**On**	Off
3	Off	Off	**On**	**On**
4	**On**	Off	Off	**On**

Tabla 3.3 Secuencia de paso doble.

La tercera secuencia es una combinación de las 2 anteriores, se le conoce como secuencia del tipo de medio paso, pues brinda un movimiento igual a la mitad del paso real, para ello se activan 2 bobinas, luego 1 solamente y así sucesivamente. La secuencia completa consta 8 movimientos en vez de 4 (Tabla 3.4).

Paso	Terminales			
	A	B	C	D
1	**On**	Off	Off	Off
2	**Onf**	**On**	Off	Off
3	Off	**On**	Off	Off
4	Off	**On**	**On**	Off
5	Off	Off	**On**	Off
6	Off	Off	**On**	**On**
7	Off	Off	Off	**On**
8	**On**	Off	Off	**On**

Tabla 3.4 Secuencia del tipo de medio paso

CIRCUITO L239B

Según la hoja de especificaciones de este circuito, es un Driver Push-Pull de 4 canales, capaz de proporcionar una corriente de salida de hasta 1 A por canal. Cada canal es controlado por señales de entrada compatibles TTL y cada pareja de canales dispone de una señal de habilitación que desconecta las salidas de los mismos. Dispone de un pin para la alimentación de las cargas que se están controlando, de forma que dicha alimentación es independiente de la lógica de control. Además tiene alta inmunidad al ruido y protección contra sobre-temperaturas. En la tabla 3.5 se describen las funciones de los 16 pines con los que cuenta.

Entre las aplicaciones de este circuito pueden mencionarse: girar 2 motores en un único sentido, controlar el giro de un motor en 2 sentidos, y el control de un motor paso a paso.

Pin	Nombre	Descripción
1	Chip Enable 1	Habilitación de los canales 1 y 2
2	Input 1	Entrada del canal 1
3	Output 1	Salida del canal 1
4	GND	Tierra de alimentación
5	GND	Tierra de alimentación
6	Output 2	Salida del canal 2
7	Input 2	Entrada del canal 2
8	Vs	Alimentación de las cargas

9	Chip Enable 2	Habilitación de los canales 3 y 4
10	Input 3	Entrada del canal 3
11	Output 3	Salida del canal 3
12	GND	Tierra de alimentación
13	GND	Tierra de alimentación
14	Output 4	Salida del canal 4
15	Input 4	Entrada del canal 4
16	Vss	Alimentación

Tabla 3.5 Descripción de los pines del L293B

PUERTO PARALELO.

Los métodos básicos para la transmisión de datos son por un lado la transmisión en serie, que consiste en enviar datos a razón de un bit a la vez a través de un cable; y la transmisión en paralelo, en la que se envían *n* número de bits por medio de *n* cables al mismo tiempo, la mayoría de los sistemas paralelos utilizan 8 líneas de datos para enviar 1 byte cada vez, aunque el estándar SCSI permite transferencia desde 8 hasta 32 bits en paralelo. La comunicación puede ser unidireccional o bidireccional. En la comunicación unidireccional hay una parte transmisora y otra receptora, la primera envía los datos y la segunda, una vez que éstos están disponibles los recibe e indica que está lista para obtener más información.

El puerto paralelo de una PC por lo general utiliza un conector tipo hembra de 25 pines (DB25). Para entender mejor la función de cada uno de estos pines, es necesario definir algunos términos, que tienen origen en el idioma inglés; por ejemplo: *handshaking* se

refiere a las operaciones entre la parte transmisora y la receptora. Para implementarlo se requieren líneas adicionales, una, la línea del estrobo o estrosboscopio (*strobe)* que utiliza la parte transmisora para comunicarle a la receptora la disponibilidad de la información, además la línea de admisión (*acknowledge*) que utiliza la parte receptora para indicarle a la transmisora que ha tomado la información y está lista para recibir más datos y la tercera llamada *busy* que puede ser utilizada por la parte receptora para indicar que está ocupada y que la transmisora no debe enviar más datos por el momento. En la tabla 3.5 se muestra la configuración estándar de los 25 pines del puerto paralelo de una PC.

Pin	Entrada/Salida	Polaridad Activa	Descripción
1	Salida	0	Strobe
2 - 9	Salida	-	Línea de datos (bit 0 pin 2, bit 7 pin 9)
10	Entrada	0	Línea acknowledge
11	Entrada	0	Línea busy
12	Entrada	1	Línea Falta de papel
13	Entrada	1	Línea Select
14	Salida	0	Línea Autofeed
15	Entrada	0	Línea Error
16	Salida	0	Línea Init
17	Salida	0	Línea Init
18 - 25	-	-	Tierra eléctrica

Tabla 3.5 Configuración estándar del puerto paralelo de una PC

3.2 DIGITALIZADOR 3D

El digitalizador 3D se compone básicamente de un soporte de acrílico con un plato que gira en donde se coloca el objeto a escanear, gracias a que está unido al motor paso a paso que se fija con tornillos a la base, y un circuito CIAT-MEX LB-71003 de Robot Maxinez que controla el movimiento del motor (Fig. 3.3). Este soporte (Fig. 3.1, Fig. 3.2) fue construido en el Centro Tecnológico de la FES Aragón.

El software envía códigos de control al circuito a través del puerto paralelo utilizando los pines del 2 al 5 que pertenecen a la línea de datos y la tierra; para ello, se modificó un cable común para impresora, conservando el conector DB25 macho, que va al puerto paralelo de la PC y cambiando el conector *Centronics* hembra de 36 pines por un DB9 hembra.

El circuito consta de un L239B y ocho diodos (para las bobinas) que sirven para proteger el circuito contra las corrientes que se producen en el momento del arranque del motor. A su vez este circuito está montado sobre una *master board,* que consta de dos conectores DB9, un macho para el cable que viene del puerto paralelo, y un hembra para enviar la información correspondiente al motor paso a paso. También tiene la entrada para la fuente de alimentación. Las entradas I1 a I4 reciben los datos que vienen de los pines 2 a 5 del puerto paralelo, y las salidas O1 a O4 van a los hilos del motor. La tierra del motor y la del circuito van a la tierra de la fuente de alimentación. El Vs y Vss van al positivo de la fuente de alimentación de 5 V. Los hilos del motor están conectados en un DB9 macho.

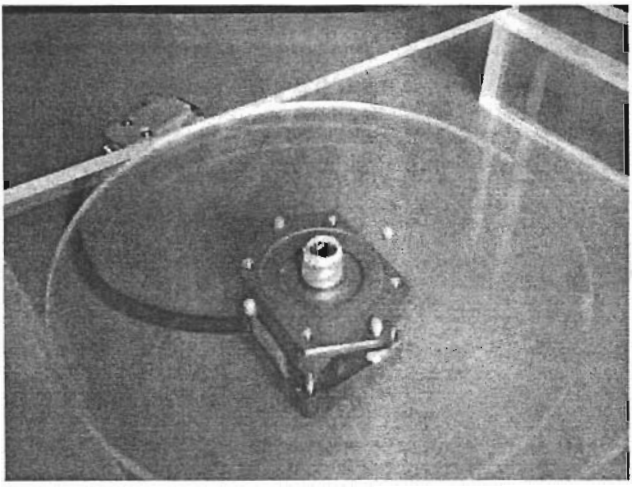


Figura 3.2.1 Plato giratorio unido al motor paso a paso

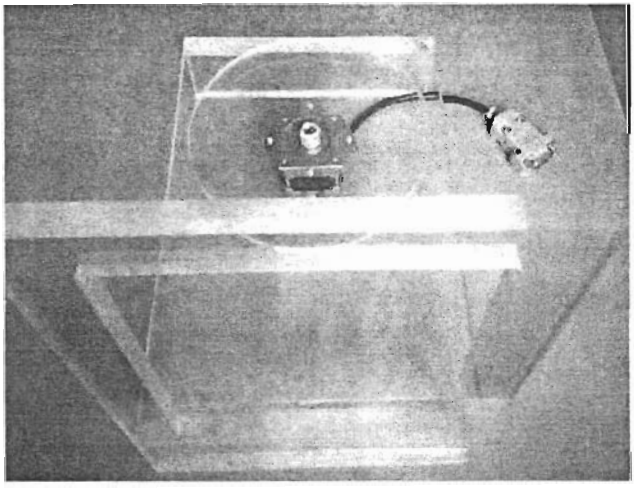


Figura 3.2 Soporte de acrílico

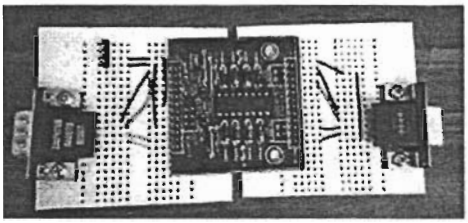
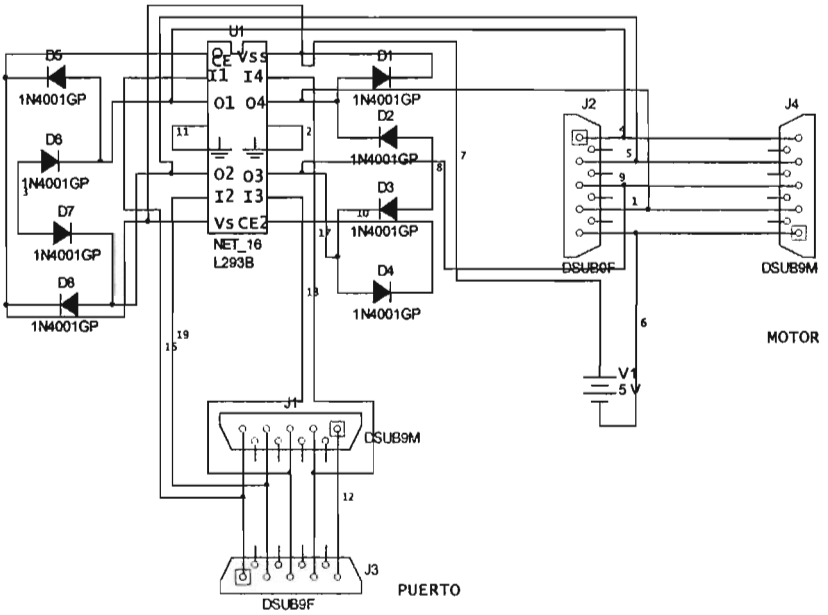


Figura 3.3 Circuito CIAT-MEX LB-71003 montado en una master board con las conexiones correspondientes



I1..I4 Entradas del puerto

O1..O4 Salidas al motor

Vs,Vss Voltaje de entrada

Figura 3.4 Diagrama eléctrico

CAPÍTULO 4. . DESARROLLO DEL SOFTWARE

El software está hecho en Borland Delphi 7.0. A continuación se describirá brevemente cada una de las funciones principales del programa en el orden en que se deben ejecutar para conseguir los resultados deseados, presentando además el código fuente correspondiente para tener una idea más clara de cómo funciona el software.

En primer lugar se sacan las fotos, para esto se usa una aplicación hecha en Delphi llamada BZHVideo[7], modificada de manera que saque las fotos y gire el motor simultáneamente. El motor recibe una secuencia de 8 datos para moverse, girando 360° en 192 pasos, por lo que el programa da la opción de sacar como máximo 192 fotos y como mínimo 1, pasando por todos los submúltiplos de 192, en el entendido de que entre más fotos se saquen el modelo será más fiel al original. La pantalla inicial se muestra en la figura 4.1. Cada fotografía recibe el nombre de Test junto con el número consecutivo correspondiente y le asigna la extensión bmp (Ej. Test027.bmp, Figura 4.4). Todas tienen una resolución de 320 X 240.

[7] BZH video es una aplicación bajada de Internet, y que para su funcionamiento correcto requiere instalar en Delphi los componentes contenidos en el paquete DSPACK231, también obtenido en Internet

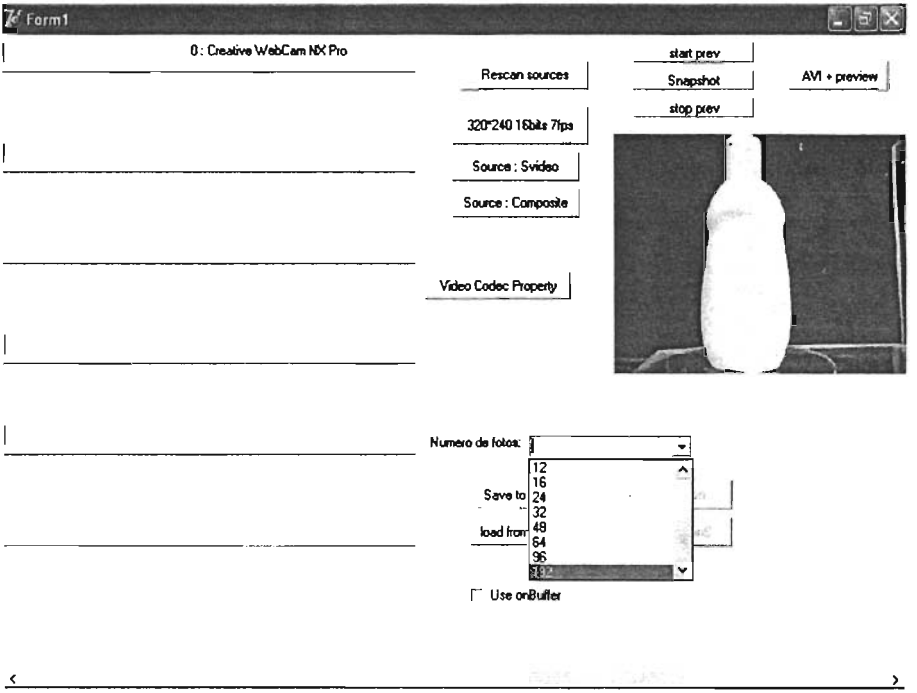


Figura 4.1 Pantalla inicial del software que saca fotos y mueve el motor

Código fuente:

```
function Out32(wAddr:word;bOut:byte):byte; stdcall; external 'inpout32.dll';


procedure TForm1.Timer1Timer(Sender: TObject);
var
  bWriteMe, bErr : byte;
  s : string;
begin
  if (Contador = 1) or ((Contador mod NumIntervalos) = 0) then
  begin
    Inc(NumFotos);
    s := Inttostr(Numfotos);
    while length(s) < 3 do
```

```
    s := '0' + s;
   BzhVideo1.SavePicture('test'+s +'.bmp', 50);
  end;

 i:=i+1;

  Inc(Contador);



  case i of
    1:bWriteMe:= 1;
    2:bWriteMe:= 5;
    3:bWriteMe:= 7;
    4:bWriteMe:= 6;
    5:bWriteMe:= 2;
    6:bWriteMe:= 10;
    7:bWriteMe:= 11;
    8:begin
       bWriteMe:=9;
       i:=0;
      end;
 end;

   bErr:=(Out32($378,bWriteMe));
  if (Contador > 192)
  begin
    Timer1.Enabled:=False;
    Button5.Enabled:=False;
    Numfotos:=0;
  end;
 end;
```

Una vez que se han obtenido las fotos con el objeto a escanear debe sacarse una foto del fondo únicamente, renombrándose diferencia.bmp, con el fin de sacar la diferencia entre las fotos con objeto y la foto sin objeto (Figura 4.3) obteniendo como resultado el objeto segmentado (Figura 4.5). Para la segmentación y los pasos siguiente se usa la aplicación Escaner3D, cuya pantalla inicial se presenta en la figura 4.2. El nombre que reciben las nuevas imágenes es diferencia más número consecutivo. (Ejemplo: diferencia027.bmp).

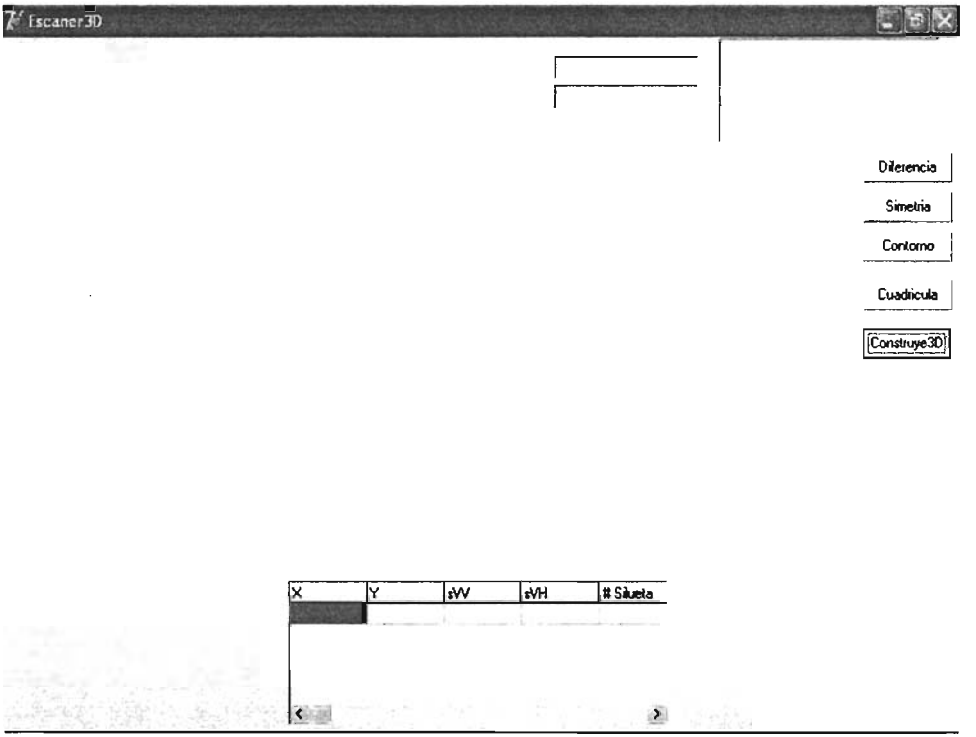


Figura 4.2 Pantalla inicial del software del digitalizador 3D

Código fuente del procedimiento diferencia:

```
Procedure TForm1.DIferenciaClick(Sender: TObject);
  Function DithColor(RGBPixel : TColor) : TColor;
```

```
  begin
    Result: = ((RGBPixel and $000000FF{%R}) +
            ((RGBPixel and $0000FF00{%G}) shr  8) +
            ((RGBPixel and $00FF0000{%B}) shr 16)) div 3;
  end;

Var
   x,y,Umbral, contador:integer;
   Diff:TColor;
   cad:string;
begin
  Umbral:=30;
  LimpiaImagen (Image1);
  LimpiaImagen(Image2);
  Image1.Picture.LoadFromFile ('c:\borland\imagenes\diferencia.bmp');
  for contador:=1 to Numfotos do
  begin
    If contador <= 9 then
      cad:='00'+ inttostr (contador)+'.bmp'
    else
      If contador <=99 then
        cad:='0'+inttostr(contador)+'.bmp'
      else
        cad:=inttostr(contador)+ '.bmp';
    Image2.Picture.LoadFromFile('c:\borland\Imagenes\test'+cad);

    for y:=0 to Image1.Height-1 do
     for x:=0 to Image1.Width-1 do
      begin
        Diff := Abs(DithColor(Image1.Picture.Bitmap.Canvas.Pixels[x, y]) -
                DithColor(Image2.Picture.Bitmap.Canvas.Pixels[x, y]));
        If Diff > Umbral then
          Image3.Canvas.Pixels[x,y]:=clBlack
```

```
      else
        Image3.Canvas.Pixels[x,y]:=clWhite;
      end;
    Image3.Picture.SaveToFile('c:\borland\Imagenes\diferencia'+cad);
  end;//for
end;
```

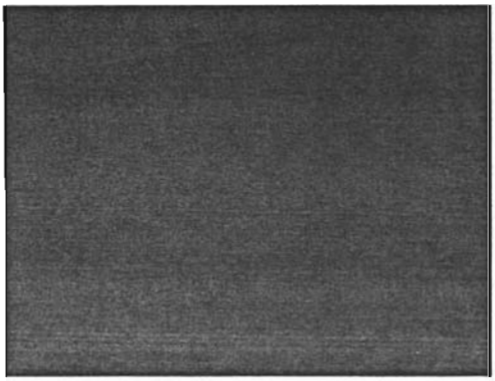


Figura 4.3 Imagen del fondo

Figura 4.4 Imagen con objeto (test027.bmp)

Figura 4.5 Objeto segmentado (diferencia027.bmp)

Cuando se pone el objeto en el plato que gira, no siempre se coloca exactamente en el centro, de manera que conforme el motor da vuelta el objeto puede parecer más pequeño o más grande según se acerque o se aleje de la cámara, y puede quedar más cargado hacia la derecha o la izquierda. Por ello, debe darse clic en el botón que dice "Simetría" que manda llamar un procedimiento en que se centran las imágenes y tomando como referencia la primera fotografía se ajustan todas las demás al mismo tamaño vertical, pues horizontalmente el objeto varía de ancho según la perspectiva que se tenga. El nombre que reciben estas imágenes es simétrica y el número consecutivo. (Ejemplo: simétrica027.bmp, figura 4.6)

Código fuente del procedimiento Simetría:

```
Procedure TForm1.SimetriaClick(Sender: TObject);
 Var
 x, y, MaxIzqO, MaxDerO, contImag,
 Arriba_base,Abajo_base,MaxAbajoO,
MaxArribaO, h0 ,w1 ,h1:integer;
 ar1:real;

begin
h0:=0;
for contImag:=1 to Numfotos do
 begin
    If contImag <= 9 then
      cad:='00'+ inttostr (contImag)+'.bmp'
    else
      If contImag <=99 then
        cad:='0'+inttostr(contImag)+'.bmp'
      else
        cad:=inttostr(contImag)+ '.bmp';
```

```
If contImag <= 3 then
   cad1:='0'+inttostr (contImag+96)+'.bmp'
else
   cad1:=inttostr (contImag+96)+'.bmp';
 Image3.Picture.LoadFromFile('c:\borland\Imagenes\diferencia'+cad);

 If (contImag=1) then
  begin
   Arriba_base:=Image3.Height;
   Abajo_base:=0;
   for y:=0 to Image3.Height-1 do
    for x:=0 to Image3.Width-1 do
     If Image3.Canvas.Pixels[x,y]=ClBlack then
        begin
         If y>Abajo_base then
          Abajo_base:=y;
         If y<Arriba_base then
          Arriba_base:=y;
        end;
   h0:=Abajo_base-Arriba_base;
  end;

MaxIzqO:=Image3.Width;
MaxDerO:=0;
MaxAbajoO:=0;
MaxArribaO:=Image3.Height;

for x:=0 to Image3.Width-1 do
   for y:=0 to Image3.Height-1 do
    begin
     If Image3.Canvas.Pixels[x,y]=clBlack then
      begin
       If x>MaxDerO then
```

```
            MaxDerO:=x;
           If x<MaxIzqO then
            MaxIzqO:=x;
           If y>MaxAbajoO then
            MaxAbajoO:=y;
           If y<MaxArribaO then
            MaxArribaO:=y;
          end;
       end;
    w1:=MaxDerO-MaxIzqO;
    h1:=MaxAbajoO-MaxArribaO;
    ar1:=w1/h1;
    w1:=round(h0*ar1);
    //Pinta en la imagen 1 la imagen 3 al tamaño deseado

    BitBlt(Image3.Canvas.Handle,0,0,MaxDerO+1,MaxAbajoO+1,
        Image3.Canvas.Handle,
        MaxIzqO,MaxArribaO,srcCopy);

    Image3.Width:=(MaxDerO+2)-MaxIzqO;
    Image3.Height:=(MaxAbajoO+2)-MaxArribaO;

    Image3.Picture.Bitmap.Width:=Image3.Width;
    Image3.Picture.Bitmap.Height:=Image3.Height;
    Image3.Canvas.StretchDraw(Rect(0, 0, Image3.Width, Image3.Height), Image3.Picture.Bitmap);
    MaxIzqO:=(Image1.Width-w1) div 2;
    Arriba_base:= (Image1.Height-h0) div 2;

Image1.Canvas.StretchDraw(Rect(MaxIzqO,Arriba_base,MaxIzqO+2+w1,Arriba_base+2+h0),Image3.Picture.Bitmap)
;
    //Limpia imagen 3
    Image3.Width:=Image1.Width;
    Image3.Height:=Image1.Height;
```

```
Image3.Picture.Bitmap.Width :=Image3.Width;
Image3.Picture.Bitmap.Height :=Image3.Height;

Image3.Canvas.Pen.Mode:= pmCopy;
Image3.Canvas.Brush.Style :=bsClear;
LimpiaImagen(Image3);

Image3.Canvas.Brush.Style:=bsSolid;
Image3.Canvas.Brush.Color:=clBlack;
Image3.Canvas.Pen.Color:=clBlack;

Image1.Picture.SaveToFile('c:\borland\Imagenes\simetrica'+cad);
//Restaura Imagen1

Image1.Canvas.Pen.Mode:= pmCopy;
Image1.Canvas.Brush.Style :=bsClear;
LimpiaImagen(Image1);
Image1.Canvas.Brush.Style:=bsSolid;
Image1.Canvas.Brush.Color:=clBlack;
Image1.Canvas.Pen.Color:=clBlack;

//Restaura Imagen2
 Image2.Width:=Image1.Width;
 Image2.Height:=Image1.Height;
 Image2.Picture.Bitmap.Width :=Image2.Width;
 Image2.Picture.Bitmap.Height :=Image2.Height;

 Image2.Canvas.Pen.Mode:= pmCopy;
 Image2.Canvas.Brush.Style :=bsClear;
 LimpiaImagen(Image2);
 Image2.Canvas.Brush.Style:=bsSolid;
 Image2.Canvas.Brush.Color:=clBlack;
 Image2.Canvas.Pen.Color:=clBlack;
```

```
    //Restaura imagen 3
    Image3.Canvas.Brush.Style :=bsClear;
    LimpiaImagen(Image3);
    Image3.Canvas.Brush.Style:=bsSolid;
    Image3.Canvas.Brush.Color:=clBlack;
    Image3.Canvas.Pen.Color:=clBlack;
  end;
end; //funcion
```

Teniendo las fotos de tamaño uniforme y centradas se saca entonces el contorno a cada una de ellas. Este procedimiento genera los archivos contorno más el número consecutivo. (Ejemplo: contorno027.bmp). En la figura 4.7 se puede observar el resultado.

Código fuente del procedimiento Contorno:

```
Procedure TForm1.ContornoClick(Sender: TObject);
  Function DithColor(RGBPixel : TColor) : TColor;
    Begin
      Result:= ((RGBPixel and $000000FF{%R}) +
            ((RGBPixel and $0000FF00{%G}) shr  8) +
            ((RGBPixel and $00FF0000{%B}) shr 16)) div 3;
    end;

  Var
    x,y,umbral,cont:integer;
    Diff, DifContDer,DifContIzq,DifContArr,DifContAba:TColor;
    Contorno:Boolean;
  Begin
    Umbral:=40;
    for cont:=1 to Numfotos do
```

```
begin
    If cont <= 9 then
     cad:='00'+ inttostr (cont)+'.bmp'
    else
     If cont <=99 then
      cad:='0'+inttostr(cont)+'.bmp'
     else
      cad:=inttostr(cont)+ '.bmp';
   Image1.Picture.Bitmap.LoadFromFile('c:\borland\Imagenes\simetrica'+cad);

   LimpiaImagen(Image2);
   Image2.Canvas.Brush.Color:=clBlack;
   For x:=1 to Image1.Width-2 do
    For y:= 1 to Image1.Height-2 do
     begin
      Diff := Abs(DithColor(Image1.Picture.Bitmap.Canvas.Pixels[x, y]) -
           DithColor(ColorFondo));
      If Diff > Umbral then
       begin
        DifContArr:= Abs (DithColor(Image1.Picture.Bitmap.Canvas.Pixels[x, y-1]) -
              DithColor(ColorFondo));
        DifContAba:= Abs (DithColor(Image1.Picture.Bitmap.Canvas.Pixels[x, y+1]) -
              DithColor(ColorFondo));

        If (DifContArr < Umbral) or (DifContAba<Umbral) then
         Image2.Canvas.Pixels[x,y]:=clBlack
        else
         If Contorno then
          Image2.Canvas.Pixels[x,y]:=clWhite//Image1.Canvas.Pixels[x,y]
         else
          Image2.Canvas.Pixels[x,y]:=clWhite;
       end
     end;
```

```
    For y:=1 to Image1.Height-2 do
     For x:= 1 to Image1.Width-2 do
      Begin
       Diff:= Abs(DithColor(Image1.Picture.Bitmap.Canvas.Pixels[x, y]) -
            DithColor(ColorFondo));
       If Diff > Umbral then
        begin
         DifContDer:= Abs (DithColor(Image1.Picture.Bitmap.Canvas.Pixels[x-1, y]) -
                DithColor(ColorFondo));
         DifContIzq:= Abs (DithColor(Image1.Picture.Bitmap.Canvas.Pixels[x+1, y]) -
                DithColor(ColorFondo));
//         Contorno:=True;
         If (DifContDer < Umbral) or (DifContIzq<Umbral) then
          Image2.Canvas.Pixels[x,y]:=clBlack
         else
        end
       end;
     Image2.Picture.SavetoFile('c:\borland\diferencia\contorno'+cad);
    end;
end;
```

Figura 4.6 Imagen simétrica

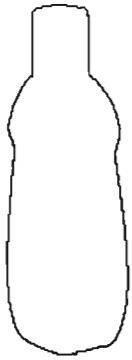


Figura 4.7 Contorno de la imagen (Contorno027.bmp)

Cada imagen contiene una gran cantidad de puntos, de manera que no sería óptimo trabajar con cada uno de ellos. Por eso, el siguiente procedimiento hace una cuadrícula de referencia y toma sólo aquellos puntos que coincidan con la cuadrícula. Así surgen las imágenes llamadas cuadrícula más el número consecutivo. (Ejemplo : cuadricula027.bmp) Véase Figura 4.8

Código fuente del procedimiento cuadrícula:

```
Procedure TForm1.CuadriculaClick(Sender: TObject);
Function LineaLibre(x1,y1,x2,y2:integer):boolean;
Var
 cfuncion,menor, mayor:integer;
Begin
  Result:=true;
  If (x1=x2) then
    begin
      If (y1<y2) then
       begin
         Mayor:=y2;
         Menor:=y1;
       end
      else
       begin
         Mayor:=y1;
         Menor:=y2;
      end;
       for cfuncion:=Menor to Mayor do
         begin
           If Image3.Canvas.Pixels[x1,cfuncion] = clblack then
             result:=False;
         end;
    end;
```

```
If (y1=y2) then
  begin
   If (x1<x2) then
    begin
     Mayor:=x2;
     Menor:=x1;
    end
    else
     begin
      Mayor:=x1;
      Menor:=x2;
     end;
   for cfuncion:=Menor to Mayor do
    begin
     If Image3.Canvas.Pixels[cfuncion,y1]= clBlack then
      Result:=False;
    end;
  end;
end;

Var
 NcuadroX, NcuadroY,CuentaX,CuentaY,Cinterno,x,y,cont:integer;
 EncontradoArIz,EncontradoAbDe:boolean;
 InicioC,FinalC:TPoint;
begin

 for cont:=1 to Numfotos do
  begin
    If cont <= 9 then
     cad:='00'+ inttostr (cont)+'.bmp'
    else
     If cont <=99 then
      cad:='0'+inttostr(cont)+'.bmp'
```

```
      else
        cad:=inttostr(cont)+ '.bmp';

{inicializa image3}

    LimpiaImagen(Image3);
    Image3.Canvas.Pen.Color:=clYellow;
    for x:= 0 to (Image3.Width div Factor) do
      begin
        Image3.Canvas.MoveTo(x*Factor,0);
        Image3.Canvas.LineTo(x*Factor,Image3.Height);
      end;
    for y:= 0 to (Image3.Height div Factor) do
      begin
        Image3.Canvas.MoveTo(0,y*Factor);
        Image3.Canvas.LineTo(Image3.Width,y*Factor);
      end;
    Image1.Canvas.Pen.Color:=clBlack;
    {termina inicializar image3}
Image2.Picture.LoadFromFile('c:\borland\Imagenes\contorno'+cad);
 NCuadroX:=Image2.Width div Factor;
 NCuadroY:=Image2.Width div Factor;
 For CuentaY:=0 to NCuadroY-1 do
   For CuentaX:= 0 to NCUadroX-1 do
     begin
       InicioC.X:=CuentaX*Factor;
       InicioC.Y:=CuentaY*Factor;
       FinalC.X:= InicioC.X+factor;//CuentaX*Factor+Factor+Factor;
       FinalC.Y:=InicioC.Y+factor;//CuentaY*Factor+Factor;
       encontradoArIz:=false;
       encontradoAbDe:=false;
```

```
If LineaLibre(InicioC.X,InicioC.Y,FinalC.X,InicioC.Y) then
  begin
   for Cinterno:=InicioC.X+1 to FinalC.X do
     begin
      If not (EncontradoArIz) then
       If (Image2.Canvas.Pixels[Cinterno,InicioC.Y]=clBlack) then
         begin
          If LineaLibre (Cinterno-Factor,InicioC.Y,Cinterno+Factor,InicioC.Y) then
          begin
           Image3.Canvas.Pixels[Cinterno,InicioC.Y]:=clBlack;
           EncontradoArIz:=true;
          end;
         end;
     end; //LineaLibre
  end;
If LineaLibre(InicioC.X,FinalC.Y,FinalC.X,FinalC.Y) then
  begin
   for Cinterno:=InicioC.X+1 to FinalC.X do
     begin
      If not (EncontradoAbDe) then
       If (Image2.Canvas.Pixels[Cinterno,FinalC.Y]=clBlack) then
         begin
          If LineaLibre (Cinterno-Factor,FinalC.Y,Cinterno+Factor,FinalC.Y) then
          begin
           Image3.Canvas.Pixels[Cinterno,FinalC.Y]:=clBlack;
           EncontradoAbDe:=true;
          end;
         end;
     end;
  end; //Linealibre
 encontradoAbDe:=false;
```

```
    If LineaLibre(FinalC.X,InicioC.Y,FinalC.X,FinalC.Y) then
      begin
       for CInterno:=InicioC.Y+1 to FinalC.Y-1 do
        begin
         If not (EncontradoAbDe) then
          If (Image2.Canvas.Pixels[FinalC.X,Cinterno]=clBlack) then
            begin
             Image3.Canvas.Pixels[FinalC.X,Cinterno]:=clBlack;
             EncontradoAbDe:=true;
            end;
        end;
      end; //Linealibre
  end;//For NcuentaX,NcuentaY

   Image3.Picture.SaveToFile('c:\borland\Imagenes\cuadricula'+cad);
 end; //For fotos
end; //Funcion
```

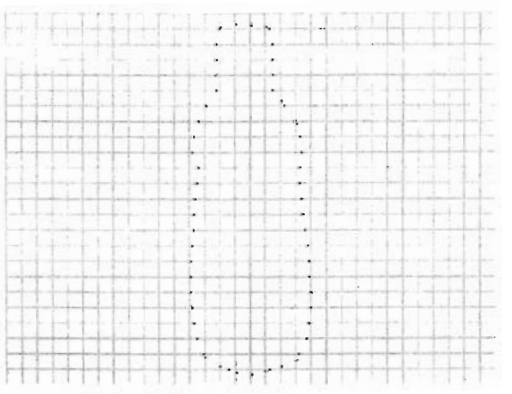


Figura 4.8 Imagen cuadriculada (cuadricula 027.bmp)

Todo lo anterior es un proceso de preparación de las imágenes para dar paso al procedimiento más importante de la aplicación, el responsable de generar un modelo en 3D. Éste manda llamar una a una las imágenes, hace un barrido hasta que encuentra el primer punto negro que pertenece a una figura y a partir de allí empieza a recorrer el contorno de la figura utilizando para ello los contornos originales , cuando encuentra un punto que coincide con la cuadrícula creada en el paso anterior lo guarda en un StringGrid (estructura de Delphi para mostrar datos en una tabla). Al terminar cada figura recorre el StringGrid, tomando cada coordenada que allí se encuentra en 2D, convirtiéndola a 3D, rotándola los grados que sean convenientes según se encontraba el objeto en la fotografía original y cambiándola de coordenadas globales a locales. Después, las guarda en un archivo de texto llamado coordenadas.obj como vértices. Con ayuda del StringGrid determina también cuáles vértices deben unirse para formar nuevamente el contorno del objeto. Una vez hecho esto con cada una de las imágenes el programa recorre nuevamente el StringGrid, en esta ocasión para determinar como se unirán horizontalmente cada uno de los puntos. Esto también se guarda en el archivo *obj*.

Al terminar una aplicación de java toma el archivo *obj* y lo grafica, permitiéndonos ver nuestro modelo en 3D desde todos los ángulos.

Código fuente del procedimiento Contruye3D:

```
Procedure TForm1.Construye3DClick(Sender: TObject);
Var
  string:textfile;
  Intervalo,Grados,Z:real;
  SoloPunto,Primera,Guarda,Continua:boolean;
```

```
  Anterior,P2D:TPoint;
  PConvertido, PRotado:MTPoint;
  contImag,contInt,x,y,PPunto,OPPunto,
  NvoPPunto, recorre, Temp, TempImagen,
  TempPunto,renglon,columna:integer;

Function LeadingChar(i : Integer; len : Byte; c : Char) : string;
begin
  Result := IntToStr(i);
  While Length(Result) < len do
    Result := c + Result;
end;


 Begin
  {inicializa archivo obj}
  assignfile (obj, 'c:\j2sdk1.4.1_01\demo\applets\WireFrame\models\coordenadas.obj');
  rewrite(obj);
  Intervalo:=360 / 192;
  TempPunto:=1;
  Continua:=False;
  for contImag:=1 to Numfotos do
    begin
      If contImag <= 9 then
        cad:='00'+ inttostr (contImag)+'.bmp'
      else
        If contImag <=99 then
          cad:='0'+inttostr(contImag)+'.bmp'
        else
          cad:=inttostr(contImag)+ '.bmp';

     Image2.Picture.LoadFromFile('c:\borland\Imagenes\contorno'+cad);
     Image3.Picture.LoadFromFile('c:\borland\Imagenes\cuadricula'+cad);
```

```
ContBIf:=0;
ListaBIf:=TList.Create;
Anterior.X:=0;
Anterior.Y:=0;
Cont_Silueta:=0;
FillChar(Coord, SizeOf(Coord), 0);
Guarda:=False;
While PuntoNegro(Coord) do {While llama función que encuentra el primer punto negro}
 begin {While Punto Negro}
  SoloPunto:=True;
  Anterior:=Coord;
  Cont_Silueta:=Cont_Silueta+1;
  Image2.Canvas.Pixels[Coord.X,Coord.Y]:=clWhite;

While PuntoNegroAdy(Coord,CAdyacente) do {While que llama función de punto negro adyacente}
    begin {While punto negro ady}

     If Image3.Canvas.Pixels[CAdyacente.X,CAdyacente.Y]=clBlack then
      Guarda:=True;

     If Guarda then
      begin
            SoloPunto:=false;
            Image3.Canvas.Pixels[CAdyacente.X,CAdyacente.Y]:=clBlack;
            StringGrid1.Cells[0,StringGrid1.RowCount-1]:=inttostr(CAdyacente.X);
            StringGrid1.Cells[1,StringGrid1.RowCount-1]:=inttostr(CAdyacente.Y);
            StringGrid1.Cells[4,StringGrid1.RowCount-1]:=inttostr(cont_silueta);
            StringGrid1.Cells[5, StringGrid1.RowCount-1]:=inttostr(contImag);
            StringGrid1.RowCount:=StringGrid1.RowCount+1;
            Anterior:=CAdyacente;
            Guarda:=False;
      end; //Guarda
```

```
        Coord:=CAdyacente;
        Image2.Canvas.Pixels[CAdyacente.X,CAdyacente.Y]:=clWhite;
end; {While punto negro ady}
      Memo1.Lines.Add('Aqui acaba una figura');
      If not (SoloPunto)  then
       begin  //not SoloPunto
        StringGrid1.RowCount:=StringGrid1.RowCount-1;
        Primera:=true;
        If CasillaLibreVert(PPunto) then
        begin
         recorre:=PPunto;
         While (recorre<=StringGrid1.RowCount-1) do
          begin
           If (StringGrid1.Cells[4,recorre]=StringGrid1.Cells[4,PPunto]) and (recorre<StringGrid1.RowCount) then
            If not (Primera) then

              StringGrid1.Cells[2,recorre-1]:=inttostr(recorre);
           If (StringGrid1.Cells[4,recorre]>StringGrid1.Cells[4,PPunto]) then
            begin
              StringGrid1.Cells[2,recorre-1]:=inttostr(PPunto);
              PPUnto:=recorre;

            end;
           If (recorre=StringGrid1.RowCount-1) then
            StringGrid1.Cells[2,recorre]:=inttostr(PPunto);
           Inc (Recorre);
           If (Primera) then
            Primera:=false;
          end;
         end;
        Grados:=Intervalo*contImag;
        PPunto:=TempPunto;
```

```
      for recorre:=PPunto to (StringGrid1.RowCount-1) do    //Escribe en archivo las coordenadas x,y
       begin //for escribe x,y
        PConvertido:=Convierte(strtoint(StringGrid1.Cells[0,recorre]),strtoint(StringGrid1.Cells[1,recorre]));
        PRotado.X := Round(PConvertido.X * cos(Grados/(360/(2*PI))));
        Z := Round(PConvertido.X * sin(Grados/(360/(2*PI))));
        writeln(obj,'v ',floattostr(PRotado.X),' ',floattostr(PConvertido.Y),' ', Z:0:3);
      end; //for escribe x,y
      write(obj,'l ');
      for recorre:=PPunto to (StringGrid1.RowCount-1) do    //Escribe en archivo las uniones verticales
       begin  //for escribe uniones verticales
        write (obj,(StringGrid1.Cells[2,recorre]),' ');
       end; //for escribe uniones verticales
      writeln(obj);
      TempPunto:=StringGrid1.RowCount;
    end;  //If not solopunto
  end; {While Punto Negro}
end;//for 1 a Numfotos
Memo1.Lines.Add('Aqui acaba el for');
Memo1.Lines.SaveToFile('todoslosadyacentes.txt');
assignfile (stringG, 'c:\borland\Imagenes\string.txt');
 rewrite(stringG);
 for renglon:= 0 to StringGrid1.RowCount do
  begin
   for columna:= 0 to StringGrid1.ColCount do
    begin
     write (stringG,StringGrid1.Cells[columna,renglon]+#9);
    end;
   writeln (stringG);
  end;
 Closefile(stringG);

   for recorre:=1 to (StringGrid1.RowCount-1) do    //Escribe en archivo las uniones verticales
     begin  //for escribe uniones verticales
```

```
            write (obj,({StringGrid1.Cells[2,recorre]}recorre),' ');
          end;
        end; //for escribe uniones verticales}
      While CasillasLibres (OPPunto) do
      begin  {une horizontales}
       Temp:=OPPunto;
       write (obj,'l ');
       write (obj,inttostr(OPPunto)+' ');
       While CasillaLibreAdy (OPPunto,NvoPPunto) do
        begin
         StringGrid1.Cells[3,OPPunto]:=inttostr(NvoPPunto);
         write (obj,StringGrid1.Cells[3,OPPunto]+' ');
         OPPunto:=NvoPPunto;
        end;
       StringGrid1.Cells[3,OPPunto]:=inttostr(Temp);
       write (obj,StringGrid1.Cells[3,OPPunto]);
       writeln (obj);
      end; {une horizontales}

     closefile (obj);

     assignfile (stringG, 'c:\borland\Imagenes\string.txt');
     rewrite(stringG);
     for renglon:= 0 to StringGrid1.RowCount do
      begin
       for columna:= 0 to StringGrid1.ColCount do
        begin
         write (stringG,StringGrid1.Cells[columna,renglon]+#9);
        end;
       writeln (stringG);
      end;
     Closefile(stringG);
end;
```

A continuación se presentan otros procedimientos y funciones que se mandan llamar a lo largo del programa acompañadas de una breve descripción de lo que hacen.

La función booleana Punto Negro hace un barrido por la imagen, si encuentra un punto negro devuelve un verdadero y en una variable las coordenadas correspondientes al punto encontrado.

Código fuente:

```
Function TForm1.PuntoNegro(var Coordenada:TPoint):Boolean;
 Var
   i,j:integer;
   vDistancia:real;
 begin
   vDistancia:=Image1.Width*2;
   Result:=False;
   for j:=0 to Image1.Width-1 do
     for i:=0 to Image1.Height-1 do
       if (Image2.Canvas.Pixels[i,j]=clBlack) and not Result then
         begin
           if Distancia(Coordenada.X,Coordenada.Y,i,j)< vDistancia then
             begin
               vDistancia:=Distancia(Coordenada.X,Coordenada.Y,i,j);
               Coordenada.X:=i;
               Coordenada.Y:=j;
               Result:=true;
             end;
         end;//I
 end; //Funcion punto negro
```

La función booleana Punto Negro Adyacente busca un punto negro adyacente a las coordenadas que se le envían como parámetros, y además determina cual de ellos permite seguir el contorno de la figura.

Código fuente:

```
function TForm1.PuntoNegroAdy(const Coord_Ant:TPoint;var Coord_Ady:TPoint):Boolean;
  var
    contador:integer;
    contador_sec:integer;
    Point_Temp:TPoint;
  begin


    BuscaPunto(Coord_Ant,Coord_Ady,contador,Image2);
     if (contador >=2) then
       begin
        BuscaPunto (Coord_Ady,Point_Temp,contador_sec,Image2);
        If (contador_sec=0) then
          begin
            Image2.Canvas.Pixels[Coord_Ady.X,Coord_Ady.Y]:=clWhite;
            BuscaPunto(Coord_Ant,Coord_Ady,contador,Image2);
          end
        else
          begin
            BuscaPunto(Coord_Ady,Point_Temp,contador_sec,Image1);
            if (contador_sec=0) then
              begin
                Image2.Canvas.Pixels[Coord_Ady.X,Coord_Ady.Y]:=clWhite;
                BuscaPunto(Coord_Ant,Coord_Ady,contador,Image2);
              end;
          end;
       end;
     Result:=(contador>0);
  end;  //funcion Punto NegroAdyacente
```

El procedimiento Busca Punto busca en los píxeles adyacentes a las coordenadas x y y dadas si hay puntos negros, en el orden de las manecillas del reloj. Devuelve las coordenadas del último punto encontrado y en una variable cuantos puntos adyacentes existen en total.

```
procedure TForm1.BuscaPunto(const Coord_Ant:TPoint;var Coord_Ady:TPoint;var contador:integer;const
Imagen:TImage);
  begin
    contador:=0;
    if (Coord_Ant.Y>0) then
      If Imagen.Canvas.Pixels[Coord_Ant.X,Coord_Ant.Y-1] = clBlack then
        begin
          Coord_Ady.X:=Coord_Ant.X;
          Coord_Ady.Y:=Coord_Ant.Y-1;
          Inc(contador);
        end;
    if (Coord_Ant.X<(Image1.Width-1)) and (Coord_Ant.Y>0) then
      if Imagen.Canvas.Pixels[Coord_Ant.X+1,Coord_Ant.Y-1] = clBlack then
        begin
          Coord_Ady.X:=Coord_Ant.X+1;
          Coord_Ady.Y:=Coord_Ant.Y-1;
          Inc(contador);
        end;
    if (Coord_Ant.X<(Image1.Width-1)) then
      if Imagen.Canvas.Pixels[Coord_Ant.X+1,Coord_Ant.Y] = clBlack then
        begin
          Coord_Ady.X:=Coord_Ant.X+1;
          Coord_Ady.Y:=Coord_Ant.Y;
          Inc(contador);
        end;
```

```
if (Coord_Ant.X<(Image1.Width-1)) and (Coord_Ant.Y<(Image1.Height-1)) then
  if Imagen.Canvas.Pixels[Coord_Ant.X+1,Coord_Ant.Y+1] = clBlack then
    begin
      Coord_Ady.X:=Coord_Ant.X+1;
      Coord_Ady.Y:=Coord_Ant.Y+1;
      Inc(contador);
    end;
if (Coord_Ant.Y<(Image1.Height-1)) then
  if Imagen.Canvas.Pixels[Coord_Ant.X,Coord_Ant.Y+1] = clBlack then
    begin
      Coord_Ady.X:=Coord_Ant.X;
      Coord_Ady.Y:=Coord_Ant.Y+1;
      Inc(contador);
    end;
if (Coord_Ant.X>0) and (Coord_Ant.Y<(Image1.Height-1)) then
  if Imagen.Canvas.Pixels[Coord_Ant.X-1,Coord_Ant.Y+1] = clBlack then
    begin
      Coord_Ady.X:=Coord_Ant.X-1;
      Coord_Ady.Y:=Coord_Ant.Y+1;
      Inc(contador);
    end;
if (Coord_Ant.X>0) then
  if Imagen.Canvas.Pixels[Coord_Ant.X-1,Coord_Ant.Y] = clBlack then
    begin
      Coord_Ady.X:=Coord_Ant.X-1;
      Coord_Ady.Y:=Coord_Ant.Y;
      Inc(contador);
    end;
```

```
  if (Coord_Ant.X>0)and (Coord_Ant.Y>0) then
   if Imagen.Canvas.Pixels[Coord_Ant.X-1,Coord_Ant.Y-1] = clBlack then
     begin
       Coord_Ady.X:=Coord_Ant.X-1;
       Coord_Ady.Y:=Coord_Ant.Y-1;
       Inc(contador);
     end;
 end; //Función
```

El siguiente procedimiento limpia la imagen que se le envía como parámetro.

```
Procedure TForm1.LimpiaImagen(var Imagen:TImage);
begin
   Imagen.Canvas.Brush.Color:=clWhite;
   Imagen.Canvas.Pen.Color :=clWhite;
   Imagen.Canvas.Rectangle(0,0,Imagen.Width,Imagen.Height);
end;
```

CAPÍTULO 5. RESULTADOS Y CONCLUSIONES

Con el trabajo realizado obtenemos un archivo *obj* que al ser graficado nos permite observar un modelo de esquemas o alámbrico del objeto escaneado en nuestra pantalla. (Figuras 5.1 y 5.2)

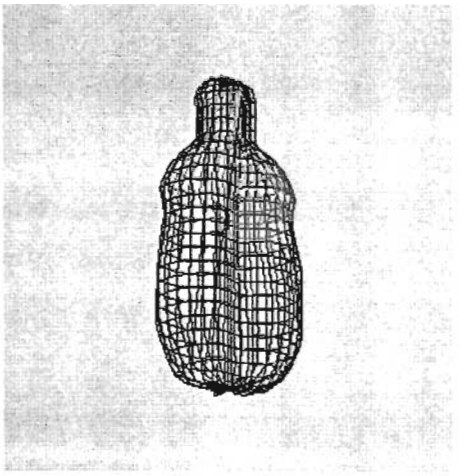


Figura 5.1 Botella escaneada vista de frente

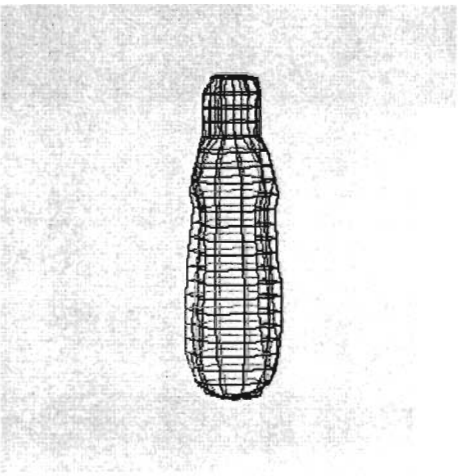


Figura 5.2 Botella escaneada vista de perfil

En cuanto a hardware se puede concluir que:

- Para mover de manera precisa el objeto a escanear lo más recomendable es el uso de un motor de pasos, controlado por un circuito integrado.
- El puerto paralelo permite la comunicación entre la computadora y el periférico.
- El soporte probablemente deba hacerse de otro material, o bien diseñarse de otra manera, pues el plato por sí solo resultaba muy pesado para el motor. El orificio del plato en el que ajusta el rotor del motor a presión fue haciéndose más

grande con el uso, y al ir girando se desplazaba hacia abajo imposibilitando el libre movimiento del motor, deteniéndolo, girándolo hacia el lado opuesto, o brincando sin seguir un orden. Al quitar el plato el motor tomaba su paso normalmente. Tal vez pudiera construirse de aluminio y agregarse engranes al equipo para que tuviera mayor capacidad de movimiento, y por lo tanto, aumentar el conjunto de objetos que pudieran ser escaneados con este dispositivo.

- En el momento de sacar fotos, las luces y el fondo juegan un papel muy importante. Al principio se usaba un fondo verde para el objeto, sin embargo, cada doblez del papel o cada reflejo de la luz que variara aunque fuera de forma mínima generaban mucho ruido. Por ello se optó por utilizar un fondo negro, menos susceptible a lo anteriormente descrito.

En cuanto a software podemos concluir que:

- A través del software el usuario puede controlar los periféricos como la cámara y el digitalizador, teniendo a su elección parámetros de resolución de la imagen original, y número de fotos. En el futuro podría agregarse que el usuario determine la resolución de la digitalización de la imagen, ya que actualmente es un parámetro fijo, entre otros.
- Para hacer más sencilla la labor del programador se usa para comunicarse con el puerto paralelo, la función externa inpout32.dll, que se encuentra disponible en Internet.

En cuanto a modelado podemos concluir que:

- Las aplicación práctica es básicamente poder obtener de manera rápida un modelo de algo real, sin tener que crearlo desde el principio como sucede cuando modelamos con programas como 3D Studio
- Este digitalizador funciona para objetos simétricos y que no tengan concavidades, pues el algoritmo que une los puntos horizontales no contempla ese caso. Figuras 5.3 y 5.4.

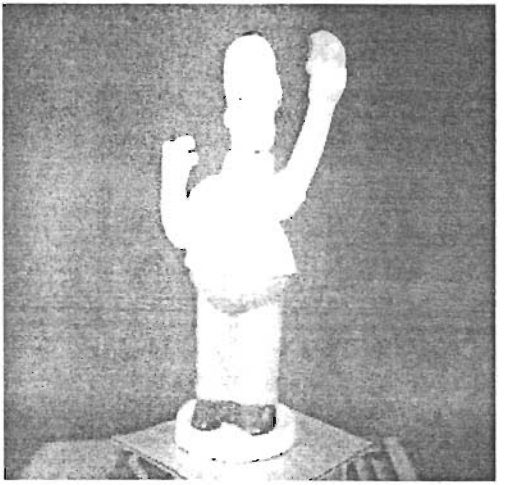


Figura 5.3 Figura original, asimétrica y con concavidades

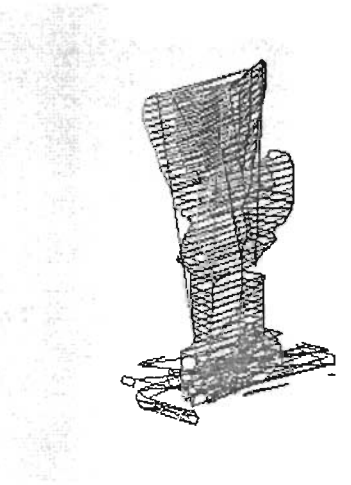


Figura 5.4 Modelo generado de la figura 5.3

BIBLIOGRAFÍA

[1] Lehmann, Charles H.
1965 *Geometría Analítica.* Ed. Hispanoamericana.
(Reimpresión de 1967)

[2] Rider, Paul R.
1940. *Álgebra.* Ed. Herrero
(Primera edición en español, 19 de octubre de 1964)

[3] Hearn M. Donald, Baker P.
1995. *Graficación por computadora.*
Ed. Prentice Hall.
(Traducción de computer graphics, segunda edición)

[4] Pérez Medel, Marcelo.
2002. *Análisis y diseño de un sistema de modelado y animación 3D GPL*
México
(Tesis maestría)

[5] Escribano Cauqui, Manuel.
1995. Programación de gráficos en 3D.
Ed. Adisson-Wesley Iberoamericana.

[6] Aguilar, Cárdenas, Martínez, Martínez.
Procesamiento Digital de Imágenes
1994

[7] Kuo, Benjamín C.
Sistemas de control digital
Ed. CECSA