Universidad de Matanzas "Camilo Cienfuegos"
Facultad de Ingenierías Química – Mecánica

MONOGRAFÍA

PROGRAMANDO PARA AUTOCAD CON VBA

SEGUNDA PARTE

Ramón Quiza Sardiñas

Departamento de Ingeniería Mecánica

Noviembre, 2006

Programando para AutoCAD con VBA – Segunda Parte.

Ramón Quiza Sardiñas
e-mail: quiza@umcc.cu

Autopista a Varadero, km 3 ½, Matanzas CP 44740, Cuba.
http://www.umcc.cu

TABLA DE CONTENIDO

CAPÍTULO 6
Textos y Cotas

OBJETIVO

Crear y modificar entidades de textos y cotas en los dibujos de AutoCAD utilizando Visual Basic for Applications.

6.1 – Estilos de texto.

Hablar sobre la importancia del trabajo con textos en el dibujo, es realmente redundar. No existe, literalmente hablando, un documento de proyecto donde no existan elementos de texto. AutoCAD proporciona dos herramientas básicas para la creación de textos: los textos de una sola línea (que se crean con el comando `TEXT`) y los textos de múltiples líneas (que se crean con el comando `MTEXT`).

Todo texto en AutoCAD tiene un estilo asociado. Cada vez que se crea una entidad de texto, AutoCAD utiliza el estilo de texto activo, con el cual establece la fuente, el tamaño, el ángulo, la orientación y otras características del texto. Los estilos de texto controlan los siguientes atributos:

Propiedad	Valor Predeterminado	Descripción
Name	`STANDARD`	Nombre del estilo
Font file	`txt.shx`	Archivo de definición de fuentes
Big Font file	`None (ninguno)`	Archivo de definición de formas especiales usado para caracteres que no pertenecen al código ASCII (como los caracteres Kanji)
Height	`0`	Altura de las letras
Width	`1`	Factor de compresión o expansión del ancho de los caracteres
Oblique angle	`0`	Inclinación de la letra
Text generation flag	`No, No`	Texto hacia atrás, o de arriba a abajo

Es posible utilizar o modificar el estilo de texto predeterminado o crear y cargar un nuevo estilo. Una vez que ha sido creado un estilo, es posible modificarlo o eliminarlo, si no se va a utilizar más.

Es posible crear cualquier estilo de texto que se necesite o se desee utilizar, excepto el estilo STANDARD. Un nuevo estilo de texto creado, heredará los atributos del estilo en uso en el momento de su creación. Para crear un nuevo estilo, se emplea el método `Add` de la colección `TextStyles`, el cual requiere, como parámetro, el nombre del nuevo estilo. Una vez creado un estilo de texto, no es posible cambiarle el nombre desde Automatización ActiveX.

Las propiedades de un estilo de texto pueden ser modificadas mediante las propiedades del objeto `AcadTextStyle` que lo contiene. En el siguiente ejemplo, se crea un nuevo estilo llamado SUDP, al cual se le asigna como archivo de definición de fuente "isocp.shx"; como altura del texto, 4 unidades; y como ángulo de inclinación, 16°.

```
Sub NuevoEstiloTexto()
  ' Crea y configura un nuevo estilo de texto
  Dim MiEstilo As AcadTextStyle
  Set MiEstilo = ThisDrawing.TextStyles.Add("SUDP")
  MiEstilo.fontFile = "isocp.shx"
  MiEstilo.Height = 4
  MiEstilo.ObliqueAngle = 3.1416 * 16 / 180
End Sub
```

Un estilo puede ser eliminado con el método `Delete`, del propio objeto que lo contiene.

6.2 – Textos de una línea.

Los textos de una línea se utilizan para representar textos sencillos. Se crean con el método `AddText`, del objeto `ModelSpace` (o `PaperSpace`, si se va a crear en el espacio de papel). Este método requiere tres parámetros: el texto que se va a agregar, el punto de inserción (determinado por una variable de arreglo de tres valores de tipo `Double`) y la altura de la letra.

El texto creado será un objeto de tipo `AcadText`, el cual posee varias propiedades que pueden ser modificadas. En el siguiente ejemplo, se muestra como crear el texto "VBA no es tan difícil como parece", en el punto de inserción (100, 100) y con una altura de 10 unidades. Finalmente, se le asigna el color rojo y el estilo "SUDP" creado en el ejemplo anterior.

```
Sub Texto()
  ' Crea un texto de una línea
  Dim Punto(0 To 2) As Double
  Dim MiTexto As AcadText
  Punto(0) = 100: Punto(1) = 100
  Set MiTexto = ThisDrawing.ModelSpace.AddText( _
            "VBA no es tan difícil como parece", _
```

```
            Punto, 10)
  MiTexto.Color = acRed
  MiTexto.StyleName = "SUDP"
  MiTexto.ObliqueAngle = 3.1416 * 16 / 180
  MiTexto.Update
End Sub
```

6.3 – Textos de múltiples líneas.

Los textos de múltiples líneas se emplean para representar textos de mayor complejidad. Se crean con el método `AddMText`, el cual requiere tres parámetros: el punto de inserción, el ancho del área asignada al texto, y el texto propiamente dicho.

Dentro de los textos de múltiples líneas, se pueden utilizar un conjunto de caracteres de control, que permiten modificar la apariencia del texto. Dentro de ellos tenemos los siguientes.

Carácter	Efecto
`\0...\o`	Activa y desactiva el sobrerayado
`\L...\l`	Activa y desactiva el subrayado
`\~`	Inserta un espacio
`\\`	Inserta un *backslash*
`\File name;`	Cambia a la fuente definida por el archivo que se especifica
`\Hvalue;`	Cambia a la altura de texto especificada en unidades
`\Hvaluex;`	Cambia a la altura de texto especificada en un múltiplo de la actual
`\Qangle;`	Cambia al ángulo de inclinación especificado
`\Wvalue;`	Cambia al factor de ancho especificado
`\A`	Cambia la alineación (0 = abajo, 1 = centro, 2 = arriba)

También el texto puede contener caracteres del código ASCII o caracteres especiales como el símbolo diámetro (%%c), el símbolo de grado (%%d) o el símbolo más o menos (%%p).

En el siguiente ejemplo, se muestra la inserción de texto de múltiples líneas, que incluye algunos caracteres es control.

```
Sub TextoMultiLineas()
  ' Crea un texto multilíneas
  Dim Punto(0 To 2) As Double
  Dim MiTexto As AcadMText
  Dim Ancho As Double, Texto As String
  Punto(0) = 100: Punto(1) = 100
  Ancho = 140
```

```
  Texto = "\Q0;La rotulación ayuda de modo " & _
          "decisivo a la claridad y belleza " & _
          "del dibujo. Un buen dibujo " & _
          "puede, por estar mal rotulado, " & _
          "perder su buen aspecto, su claridad e " & _
          "incluso su utilidad. \Q15;\LW. " & _
          "Schneider\l (Manual Práctico de Dibujo)"
  Set MiTexto = ThisDrawing.ModelSpace.AddMText( _
  Punto, Ancho, Texto)
  MiTexto.Color = acRed
End Sub
```

Al igual que con los textos de una sola línea, los de múltiples líneas, tienen propiedades que permiten controlar su aspecto.

6.4 – Cotas.

Cotas lineales.

Las cotas lineales pueden ser alineadas o rotadas. En las primeras, la línea de dimensión es paralela a la línea definida por los orígenes de la línea de extensión. En las segundas, estas dos líneas forman un ángulo determinado (Fig. 6.1).

Mediante los métodos `AddDimAligned` y `AddDimRotated`, se crean cotas lineales alineadas y rotadas, respectivamente. Al primer método, se le pasan como parámetros tres puntos que localizan los orígenes de las líneas de extensión y posición de la línea de dimensión. Los tres puntos son variables de arreglo de tres valores de tipo `Double`.

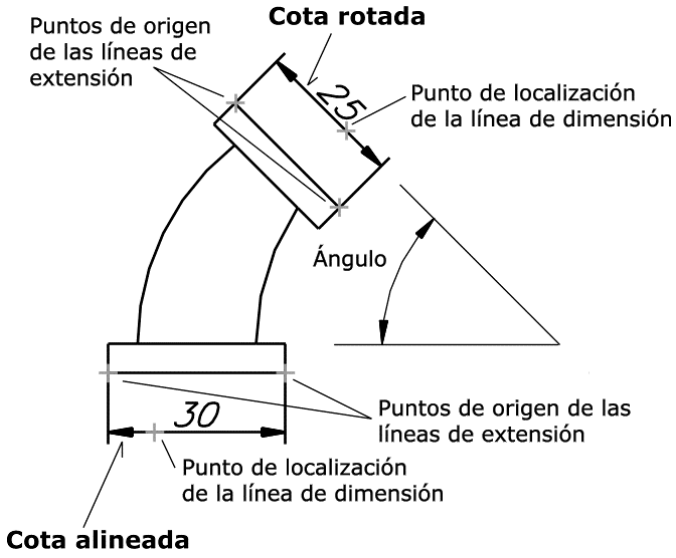


Fig. 6.1 – Cotas alineadas y rotadas.

En el siguiente código, se muestra como crear una cota alineada. En el mismo, los orígenes de las líneas de extensión pasarán por los puntos (100, 100) y (130, 100) (contenidos en las variables `Pto1` y `Pto2`) y la línea de dimensión pasará por el punto (115, 90).

```
' Crea una cota alineada
Dim CotaAlineada As AcadDimAligned
Dim Pto1(0 To 2) As Double, Pto2(0 To 2) As Double, _
Pto3(0 To 2) As Double
Pto1(0) = 100: Pto1(1) = 100
Pto2(0) = 130: Pto2(1) = 100
Pto3(0) = 115: Pto3(1) = 90
Set CotaAlineada = ThisDrawing.ModelSpace. _
AddDimAligned(Pto1, Pto2, PtoTexto)
```

Al método `AddDimRotated`, además de los tres puntos anteriormente citados, hay que pasarle un valor de tipo `Double` que representa el ángulo que forma la línea de dimensión con el eje *x*.

En el siguiente ejemplo se muestra la creación de una cota rotada a 45º.

```
' Crea una cota rotada
Dim CotaRotada As AcadDimRotated
Dim Pto1(0 To 2) As Double, Pto2(0 To 2) As Double, _
Pto3(0 To 2) As Double
Dim Angulo As Double
Pto1(0) = 100: Pto1(1) = 100
Pto2(0) = 130: Pto2(1) = 130
Pto3(0) = 107.03: Pto3(1) = 122.07
Angulo = 3.1416 / 4
Set CotaRotada = ThisDrawing.ModelSpace. _
AddDimRotated(Pto1, Pto2, Pto3, Angulo)
```

Cotas circulares.

Las cotas circulares permiten acotar los radios y diámetros. Se crean con los métodos `AddDimRadial` y `AddDimDiametric`, respectivamente.

El método `AddDimRadial` requiere tres parámetros: el punto central del círculo o arco que se desea acotar, el punto de inicio de la línea de indicación (que está situado sobre la circunferencia o arco) y la longitud de dicha línea de indicación (ver Fig. 6.2).

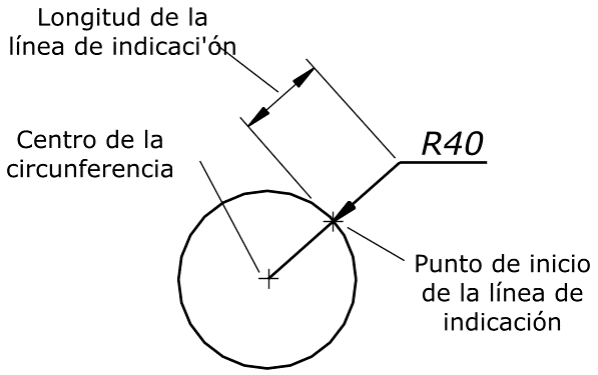


Fig. 6.2 – Cotas radiales.

En el siguiente código se muestra la creación de una cota radial. La variable `Pto1` contiene las coordenadas del centro de la circunferencia; `Pto2`, las coordenadas del punto de origen de la línea de indicación; y `Longitud`, la longitud de dicha línea.

```
' Crea una cota radial
Dim CotaRadial As AcadDimRadial
Dim Pto1(0 To 2) As Double, Pto2(0 To 2) As Double
Dim Longitud As Double
Dim Circulo As AcadCircle
Pto1(0) = 100: Pto1(1) = 100
Pto2(0) = 130: Pto2(1) = 130
Longitud = 20
Set Circulo = ThisDrawing.ModelSpace. _
AddCircle(Pto1, 42.5264)
Set CotaRadial = ThisDrawing.ModelSpace. _
AddDimRadial(Pto1, Pto2, Longitud)
```

La creación de cotas diametrales es muy similar a la de cotas radiales, tal como se puede ver en la figura 6.3.

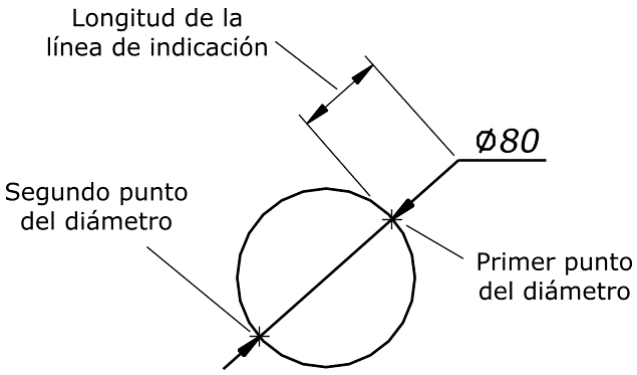


Fig. 6.3 – Cotas diametrales.

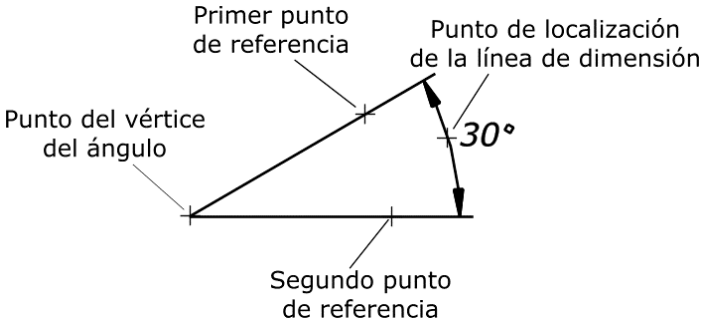
El método `AddDimDiametric` requiere tres parámetros: el primer punto del diámetro que se va a acotar (que coincide con el origen de la línea de indicación); el segundo punto del diámetro y la longitud de la línea de indicación

En el siguiente ejemplo, se crea una cota diametral. Las variables `Pto0`, `Pto1` y `Pto2` contienen las coordenadas del punto central de la circunferencia, del primer punto del diámetro y del segundo punto de diámetro, respectivamente. La variable `Longitud` contiene la longitud de la línea de indicación.

```
' Crea una cota diametral
Dim CotaDiametral As AcadDimDiametric
Dim Pto0(0 To 2) As Double, Pto1(0 To 2) As Double, _
    Pto2(0 To 2) As Double
Dim Longitud As Double
Dim Circulo As AcadCircle
Pto0(0) = 100: Pto0(1) = 100
Pto1(0) = 130: Pto1(1) = 130
Pto2(0) = 70: Pto2(1) = 70
Longitud = 20
Set Circulo = ThisDrawing.ModelSpace. _
               AddCircle(Pto0, 42.5264)
Set CotaDiametral = ThisDrawing.ModelSpace. _
                     AddDimDiametric(Pto1, Pto2, Longitud)
```

Cotas angulares.

Las cotas angulares se crean con el método AddDimAngular, al cual se le pasan como parámetros cuatro puntos (variables de arreglo de tres elementos de tipo Double) que indican la posición del vértice del ángulo, del punto de referencia para el primer rayo del ángulo, del punto de referencia para el segundo punto y del punto de localización de la línea de dimensión (ver Fig. 6.4).



Fig, 6.4 – Cota angular.

Para ejemplificar la creación de una cota angular, se muestra el siguiente código.

```
' Crea una cota angular
Dim CotaAngular As AcadDimAngular
Dim Pto0(0 To 2) As Double, Pto1(0 To 2) As Double, _
    Pto2 As Variant, Pto3 As Variant
Pto0(0) = 100: Pto0(1) = 100
Pto1(0) = 130: Pto1(1) = 100
Pto2 = ThisDrawing.Utility.PolarPoint(Pto0, 3.1416/6, 30)
Pto3 = ThisDrawing.Utility.PolarPoint(Pto0, 3.1416/12, 40)
ThisDrawing.ModelSpace.AddLine Pto0, Pto1
ThisDrawing.ModelSpace.AddLine Pto0, Pto2
Set CotaAngular = ThisDrawing.ModelSpace. _
                  AddDimAngular(Pto0, Pto1, Pto2, Pto3)
```

Nótese en uso del método `PolarPoint`, del objeto `Utility`, que devuelve las coordenadas cartesianas de un punto, dados un punto de referencia, el valor de radio vector que va desde el este hasta el punto calculado, y el ángulo que forma dicho vector con el eje *x*.

Cotas ordinales.

Las cotas ordinales permiten indicar una de las coordenadas de un punto determinado. Para agregar una cota ordinal, se utiliza el método `AddDimOrdinate`, el cual requiere tres parámetros: el punto que se desea acotar, el punto final de la línea de indicación y un valor que puede ser verdadero o falso (`True/False`) y que indica si la coordenada a acotar será la *x* (cuando toma valor `True`) o la *y* (cuando es `False`).

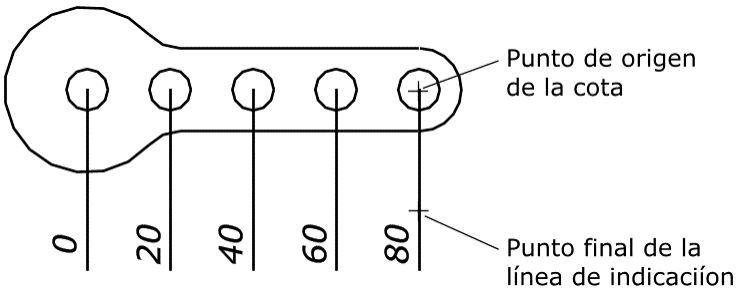


Fig. 6.5 – Cotas ordinales.

Las coordenadas son determinadas a partir del origen del sistema de coordenadas definidas por el usuario (UCS), por lo cual si queremos que la coordenada sea acotada a partir de un determinado punto, primero debemos mover el UCS a éste.

En el siguiente ejemplo, se muestra como crear una cota ordinal, que indica la coordenada *x* del vértice superior de un triángulo.

```
' Crea una cota ordinal
Dim CotaOrdinal As AcadDimOrdinate
Dim PtosTr(0 To 11) As Double
Dim Triangulo As AcadPolyline
Dim Pto0(0 To 2) As Double, Pto1(0 To 2) As Double
Dim EjeX As Integer
PtosTr(0) = 0: PtosTr(1) = 0: PtosTr(2) = 0
PtosTr(3) = 60: PtosTr(4) = 0: PtosTr(5) = 0
PtosTr(6) = 30: PtosTr(7) = 40: PtosTr(8) = 0
PtosTr(9) = 0: PtosTr(10) = 0: PtosTr(11) = 0
Set Triangulo = ThisDrawing.ModelSpace.AddPolylin(PtosTr)
Pto0(0) = 30: Pto0(1) = 40
Pto1(0) = 30: Pto1(1) = 50
EjeX = True
Set CotaOrdinal = ThisDrawing.ModelSpace. _
                    AddDimOrdinate(Pto0, Pto1, EjeX)
```

6.5 – Edición de cotas.

Como, al fin y al cabo, las cotas son también entidades gráficas, es posible editarlas utilizando las propiedades y métodos que proporciona el objeto que las contiene. La mayor parte de los objetos de cotas contienen, entre otras, las siguientes propiedades:

`Rotation`	Define el ángulo de rotación, en radianes, de la línea de dimensión de la cota
`StyleName`	Define el estilo de acotado
`TextPrefix`	Define el prefijo del texto de la cota
`TextSuffix`	Define el sufijo del texto de la cota
`TextOverride`	Define el texto que aparecerá en la cota
`TextPosition`	Define la posición del texto de la cota
`TextRotation`	Define el ángulo de rotación del texto de la cota
`Measurement`	Contiene el valor numérico real de la cota (sólo lectura)

Aparte de los anteriormente citados, los objetos de cotas poseen otros métodos y propiedades de edición comunes al resto de los objetos gráficos, como son: `ArrayPolar`, `ArrayRectangular`, `Copy`, `Erase`, `Mirror`, `Move`, `Rotate` y `ScaleEntity`.

Ejemplo resuelto.

Elaborar una macro que permita, dado el diámetro del escalón de un árbol, donde se encuentra un chavetero, dibujar la sección transversal del mismo.

Solución.

La solución del problema se dividirá en cinco etapas: definición de las variables, toma de datos del usuario, cálculo y dibujo de los contornos y los ejes, dibujo del rayado y cálculo y dibujo de las cotas.

Las variables se definen en dependencia de su uso. Las variables `D`, `B` y `T`, son de tipo `Double`, y representan el diámetro, el ancho y la profundidad del chavetero, respectivamente. Las variables `ang0A` y `ang0B` representan los ángulos de inicio y final del arco del contorno del chavetero, y son de tipo `Double`. La variable p0 representa las coordenadas del punto de inserción (punto central) y `pA`, ... , `pM` el resto de los puntos necesarios para construir el dibujo (ver Fig. 6.6).

Las variables que contienen los puntos son declaradas como arreglos de tres valores de tipo `Double`, si sus coordenadas se especifican explícitamente, o de tipo `Variant`, si son obtenidas mediante algún método auxiliar o a partir de los puntos notables de otra entidad.

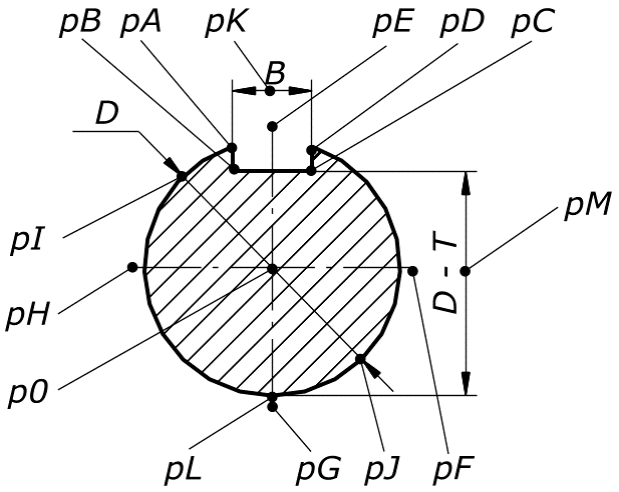


Fig. 6.6 – Esquema del ejercicio propuesto.

Para representar el arco, se define la variable `aAD`, de tipo `AcadArc`, mientras que para las líneas, se utilizan las variables `lAB`, `lBC` ... , de tipo `AcadLine`. Los contornos del rayado se asignan a la variable `Contorno`, que es un arreglo de cuatro valores de tipo `AcadEntity`. El rayado propiamente dicho, en la variable `Rayado` de tipo `AcadHatch`. Finalmente, se definen las variables `cD`, `cB` y `cT`, de tipo `AcadDimDiametric`, `AcadDimAligned` y `AcadDimRotated`, respectivamente,

las cuales contendrán las cotas del diámetro, del ancho y de la distancia desde el fondo del chavetero hasta el cuadrante opuesto.

Los datos necesarios para dibujar la sección del escalón son el punto de inserción (que corresponde al centro de la sección) y el diámetro de la misma. Se obtienen con los métodos `GetPoint` y `GetReal`, del objeto `Utility`. Como paso previo al dibujo, y formando parte de la propia toma de datos, se comprueba que el diámetro suministrado esté en el rango considerado en la solución del problema 17 mm ... 65 mm. En caso contrario, se envía un mensaje a la consola (con el método `Prompt`) y se termina la ejecución de la subrutina (instrucción `Exit Sub`).

Los valores del ancho y la profundidad del chavetero se determinan a partir de la norma ISO correspondiente, obteniéndose los valores que se muestran en la siguiente tabla:

Diámetro (D)	Ancho(B)	Profundidad (T)
17 ... 22	6	3.5
22 ... 30	8	4
30 ... 38	10	5
38 ... 44	12	5
44 ... 50	14	5.5
50 ... 58	16	6
58 ... 65	18	7

Para asignar los valores a las variables `B` y `T`, se implementan un grupo de estructuras condicionales que analizan si `D` está en el rango correspondiente a cada fila.

Para calcular los ángulos de inicio y fin del arco del contorno de la sección, se emplean las expresiones:

$$\sphericalangle OA = \frac{p}{2} + \arctan\left(\frac{B}{\sqrt{D^2 - B^2}}\right); \text{ y} \tag{6.1}$$

$$\sphericalangle OD = \frac{p}{2} - \arctan\left(\frac{B}{\sqrt{D^2 - B^2}}\right); \tag{6.2}$$

las cuales se deducen, a partir del siguiente esquema:

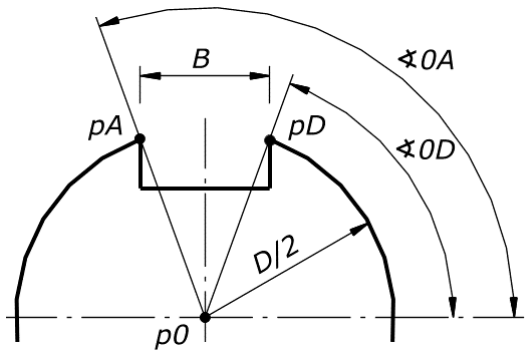


Fig. 6.7 – Esquema de los ángulos inicial y final del arco del contorno.

Con los ángulos anteriores, el diámetro del arco (que es el mismo que el del escalón) y el punto central (punto de inserción) se dibuja el arco con el método `AddArc`.

Para determinar las coordenadas de los puntos `pA` y `pD`, se emplean las propiedades `StartPoint` y `EndPoint`, del arco trazado, ya que, como resulta evidente, estos puntos coinciden con el inicio y el final del arco, respectivamente. Las coordenadas de los puntos `pB` y `pC` se determinan a partir de los anteriores mediante relaciones geométricas sencillas. Las líneas `lAB`, `lBC` y `lCD` se dibujan con el método `AddLine`.

Los valores de los extremos de los ejes (puntos `pE`, `pF`, `pG` y `pH`) se calculan a partir de las coordenadas del punto de inserción (`p0`) y del diámetro `D`. También las líneas de eje (`lEG` y `lFH`) se añaden con el método `AddLine`.

Para crear el contorno, se asigna a cada uno de los valores de la variable arreglo Contorno los objetos que contienen al arco y los tres segmentos de recta `lAB`, `lBC` y `lCD`. Acto seguido, se crea el rayado con el método `AddHatch`, utilizando como parámetros el tipo de rayado `acHatchPatternTypePreDefined` (predefinido por AutoCAD), el nombre del patrón de rayado `"Line"` y la asociatividad del rayado como `False` (no asociar). Después de creado el rayado, se cambia su ángulo (`PatternAngle`) a 45º (3,1416 / 4) y se le asigna, como contorno exterior la variable Contorno, mediante el método `AppendOuterLoop`. Finalmente, se le aplica el método Validate, para crear efectivamente el rayado.

El último paso, consiste en determinar las coordenadas de los puntos necesarios para definir las tres cotas. Se destaca el uso del método `PolarPoint`, tal como se vio anteriormente, para determinar las coordenadas cartesianas a partir de las polares. Las cotas se agregan mediante los métodos `AddDimDiametric`, `AddDimAligned` y `AddDimRotated`, para el diámetro, el ancho y la distancia del fondo del chavetero al cuadrante opuesto, respectivamente. Una vez creadas las cotas, se modifica el texto que muestran, mediante la propiedad `TextOverride`, de cada una de ellas, para evitar que se muestren con imprecisiones debidas a los errores de redondeo en los cálculos de los diversos puntos.

El código del ejemplo resuelto se muestra a continuación:

```
Sub SeccionChavetero()
  ' Dibuja la sección transversal del escalón de
  ' un árbol que contiene un chavetero
  ' Definición de variales
  Dim D As Double, B As Double, T As Double
  Dim ang0A As Double, ang0D As Double
  Dim p0 As Variant, pA As Variant, _
      pB(0 To 2) As Double, pC(0 To 2) As Double, _
      pD As Variant, pE(0 To 2) As Double, _
      pF(0 To 2) As Double, pG(0 To 2) As Double, _
      pH(0 To 2) As Double, pI As Variant, _
      pJ As Variant, pK(0 To 2) As Double, _
      pL(0 To 2) As Double, pM(0 To 2) As Double
  Dim aAD As AcadArc
  Dim lAB As AcadLine, lBC As AcadLine, _
      lCD As AcadLine, lEG As AcadLine, _
      lFH As AcadLine
  Dim Contorno(0 To 3) As AcadEntity
  Dim Rayado As AcadHatch
  Dim cD As AcadDimDiametric, _
      cB As AcadDimAligned, _
      cT As AcadDimRotated
  ' Toma de datos
  p0 = ThisDrawing.Utility.GetPoint(, _
         "Entre el punto de inserción: ")
  D = ThisDrawing.Utility.GetReal( _
        "Entre el diámetro de escalón: ")
  ' Comprobación del diámetro
  If (D < 17) Or (D > 65) Then
    ThisDrawing.Utility.Prompt ( _
      "El diámetro está fuera del rango válido")
    Exit Sub
  End If
  ' Cálculo de los parámetros
  If (D >= 17) And (D < 22) Then B = 6: T = 3.5
  If (D >= 22) And (D < 30) Then B = 8: T = 4
  If (D >= 30) And (D < 38) Then B = 10: T = 5
  If (D >= 38) And (D < 44) Then B = 12: T = 5
  If (D >= 44) And (D < 50) Then B = 14: T = 5.5
  If (D >= 50) And (D < 58) Then B = 16: T = 6
  If (D >= 58) And (D <= 65) Then B = 18: T = 7
  ' Cálculo y dibujo del arco del contorno
  ang0A = 3.1416 / 2 + Atn(B / (D ^ 2 - B ^ 2) ^ 0.5)
  ang0D = 3.1416 / 2 - Atn(B / (D ^ 2 - B ^ 2) ^ 0.5)
  Set aAD = ThisDrawing.ModelSpace.AddArc(p0, D / 2, _
            ang0A, ang0D)
```

```
  ' Cálculo y dibujo de las líneas del contorno
  pA = aAD.StartPoint
  pD = aAD.EndPoint
  pB(0) = pA(0): pB(1) = p0(1) + D / 2 - T
  pC(0) = pD(0): pC(1) = pB(1)
  Set lAB = ThisDrawing.ModelSpace.AddLine(pA, pB)
  Set lBC = ThisDrawing.ModelSpace.AddLine(pB, pC)
  Set lCD = ThisDrawing.ModelSpace.AddLine(pC, pD)
  ' Cálculo y dibujo de los ejes
  pE(0) = p0(0): pE(1) = p0(1) + D / 2 + 2
  pF(0) = p0(0) + D / 2 + 2: pF(1) = p0(1)
  pG(0) = p0(0): pG(1) = p0(1) - D / 2 - 2
  pH(0) = p0(0) - D / 2 - 2: pH(1) = p0(1)
  Set lEG = ThisDrawing.ModelSpace.AddLine(pE, pG)
  Set lFH = ThisDrawing.ModelSpace.AddLine(pF, pH)
  ' Dibujo del rayado
  Set Contorno(0) = lAB
  Set Contorno(1) = lBC
  Set Contorno(2) = lCD
  Set Contorno(3) = aAD
  Set Rayado = ThisDrawing.ModelSpace.AddHatch( _
                 acHatchPatternTypePreDefined, _
                 "Line", False)
  Rayado.PatternAngle = 3.1416 / 4
  Rayado.AppendOuterLoop Contorno
  Rayado.Evaluate
  ' Cálculo y dibujo de las cotas
  pI = ThisDrawing.Utility.PolarPoint(p0, _
         5 * 3.1416 / 6, D / 2)
  pJ = ThisDrawing.Utility.PolarPoint(p0, _
         -3.1416 / 5, D / 2)
  Set cD = ThisDrawing.ModelSpace.AddDimDiametric( _
           pI, pJ, 10)
  cD.TextOverride = "%%c" & D
  pK(0) = p0(0): pK(1) = p0(1) + D / 2 + 10
  Set cB = ThisDrawing.ModelSpace.AddDimAligned( _
           pA, pD, pK)
  cB.TextOverride = B
  pL(0) = p0(0): pL(1) = p0(1) - D / 2
  pM(0) = p0(0) + D / 2 + 10: pM(1) = p0(1)
  Set cT = ThisDrawing.ModelSpace.AddDimRotated( _
           pL, pC, pM, -3.1416 / 2)
  cT.TextOverride = D - T
End Sub
```

Preguntas y ejercicios propuestos.

1. ¿Qué utilidad tienen los estilos de texto? ¿Cómo pueden crearse desde VBA?

2. ¿Qué método permite crear textos de una línea? ¿Y de múltiples líneas?

3. ¿Para qué sirven los caracteres de control en los textos de múltiples líneas?

4. ¿Qué diferencia existe entre las cotas alineadas y las rotadas?

5. ¿Qué tipos de cotas se emplean en los elementos circulares? ¿Con qué métodos se crean?

6. Escriba un programa para dibujar un símbolo de tolerancia de cilindricidad, dados los puntos de inserción y el valor de la tolerancia.

7. Escriba un programa para dibujar el croquis de la sección de una viga doble T, con sus cotas, dados por el usuario el punto de inserción y sus dimensiones.

CAPÍTULO 7
Entidades Tridimensionales

OBJETIVO

Crear y modificar entidades de AutoCAD en el espacio tridimensional. utilizando Visual Basic for Applications.

7.1 – Sistemas de coordenadas.

Una parte importante del trabajo en el espacio tridimensional es la manipulación de los sistemas de coordenadas. Los sistemas de coordenadas se controlan, desde VBA, con la colección `UserCoordinateSystems`, del objeto `ThisDrawing`. Con el método `Add` de la misma, es posible crear nuevos sistemas de coordenadas definidos por el usuario, indicando el origen de coordenadas, un punto situado sobre el eje de las *x* y uno situado sobre el eje de las *y*. Naturalmente, una vez especificados estos dos ejes, el eje *z* queda automáticamente definido, según la regla de la mano derecha (ver Fig, 7.1).

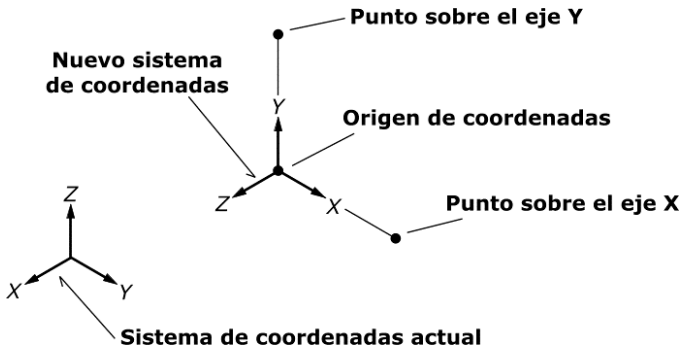


Fig. 7.1 – Creación de un nuevo sistema de coordenadas.

Los tres datos anteriores son pasados, como parámetros al método `Add`, en forma de arreglos de tres valores de tipo `Double` (tal como se manipulan los puntos en la mayor parte de las aplicaciones de VBA). Además, requiere un cuarto parámetro que es el nombre que se le asignará al sistema de coordenadas.

En el siguiente ejemplo, se muestra como crear un sistema de coordenadas que corresponde al mostrado en la figura anterior. Para ello, el origen de coordenadas del nuevo sistema se ubicará en el punto (100, 100, 0); el punto que define el segmento positivo del eje *x*, en punto (100, 101, 0), o sea, desplazado una unidad en el sentido de las *y* con respecto al anterior; y el punto que define a la parte positiva del eje de las *y*, está situado en el punto (100, 100, 1), o sea, desplazado una unidad en dirección de las *z*, con

respecto al nuevo origen. Al nuevo sistema de coordenadas se le asignará el nombre "NuevoUCS" y se contendrá en la variable MiUCS, de tipo AcadUCS.

Una vez creado el nuevo sistema de coordenadas, se modifica la propiedad ActiveUCS, del objeto ThisDrawing, asignándole como valor la variable MiUCS, para hacer que el sistema recién creado sea el activo en el dibujo. A través del ícono del sistema de coordenadas se pueden ver los efectos de la modificación.

```
Sub CrearUCS()
  ' Crea un nuevo sistema de coordenadas
  Dim MiUCS As AcadUCS
  Dim Punto0(0 To 2) As Double, _
      PuntoX(0 To 2) As Double, _
      PuntoY(0 To 2) As Double
  Punto0(0) = 100: Punto0(1) = 100: Punto0(2) = 0
  PuntoX(0) = 100: PuntoX(1) = 101: PuntoX(2) = 0
  PuntoY(0) = 100: PuntoY(1) = 100: PuntoY(2) = 1
  Set MiUCS = ThisDrawing.UserCoordinateSystems. _
        Add(Punto0, PuntoX, PuntoY, "NuevoUCS")
  ThisDrawing.ActiveUCS = MiUCS
End Sub
```

7.2 – Visualización de los dibujos.

La manipulación de los puntos de vistas, desde VBA, se realiza con los objetos de tipo AcadView y AcadViewport. El objeto ThisDrawing proporciona dos colecciones llamadas Views y Viewports, que contienen los puntos de vista y las vistas.

La modificación del punto de vista de la vista actual se puede realizar modificando directamente la propiedad Direction, del objeto ActiveViewport, o creando una nueva vista y asignándosela a la propiedad View, de este objeto.

```
Sub CambiarPuntoDeVista()
  ' Cambia el punto de vista
  Dim MiVista As AcadView
  Dim Direccion(0 To 2) As Double
  Set MiVista = ThisDrawing.Views.Add("NuevoPV")
  Direccion(0) = 1: Direccion(1) = 2: Direccion(2) = 1
  MiVista.Direction = Direccion
  ThisDrawing.ActiveViewport.SetView MiVista
  ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport
  ZoomAll
End Sub
```

En el código anterior se ejemplifica la segunda variante: se crea una vista nueva, a la cual se le establece como dirección de visualización el punto (1, 2, 1). Luego esta vista es asignada a la propiedad `View` de `ActiveViewPort`.

Nótese la asignación final de la propiedad `ActiveViewport` a sí misma. Esta sintaxis es necesaria para que se actualicen las propiedades de la vista. También debe señalarse el uso del procedimiento `ZoomAll`, que, como se deduce de su nombre, cambia la magnitud de la visualización de forma tal que se muestre todo el contenido del dibujo en pantalla.

7.3 – Creación de modelos de alambre.

Los modelos de alambres, son aquellos en los que los cuerpos se representan sólo por los elementos que definen sus aristas (normalmente, líneas rectas y arcos). La creación de estos modelos no implica ningún conocimiento especial, salvo la consideración de la tercera coordenada en todos los puntos.

Para ejemplificar el trabajo con modelos de alambre, se muestra el siguiente código, donde se crea un tetraedro regular de arista igual a 100 unidades. Las coordenadas de los cuatro vértices (ver Fig. 7.2) se calculan previamente basándose en consideraciones geométricas.

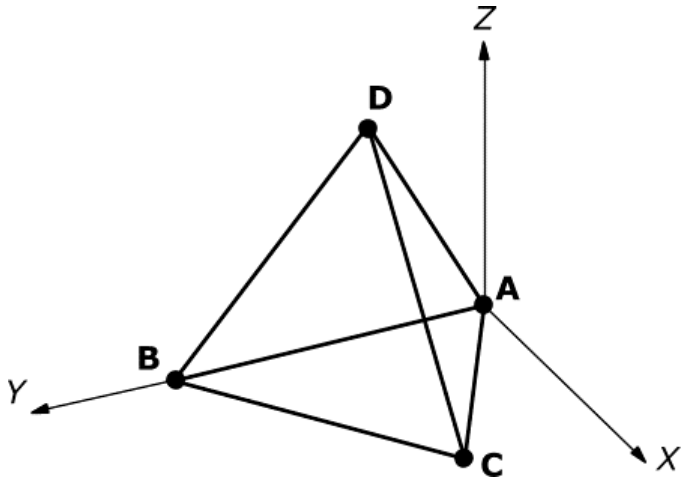


Fig. 7.2 – Modelo de alambre de un tetraedro regular.

```
Sub AlambreTetraedro()
  ' Crea el modelo de alambre de un
  ' tetraedro regular
  Dim PtoA(0 To 2) As Double, PtoB(0 To 2) As Double, _
      PtoC(0 To 2) As Double,  PtoD(0 To 2) As Double
  PtoA(0) = 0: PtoA(1) = 0: PtoA(2) = 0
  PtoB(0) = 100: PtoB(1) = 0: PtoB(2) = 0
```

```
  PtoC(0) = 50: PtoC(1) = 86.6025: PtoC(2) = 0
  PtoD(0) = 50: PtoD(1) = 28.8675: PtoD(2) = 81.6497
  ThisDrawing.ModelSpace.AddLine PtoA, PtoB
  ThisDrawing.ModelSpace.AddLine PtoB, PtoC
  ThisDrawing.ModelSpace.AddLine PtoC, PtoA
  ThisDrawing.ModelSpace.AddLine PtoA, PtoD
  ThisDrawing.ModelSpace.AddLine PtoB, PtoD
  ThisDrawing.ModelSpace.AddLine PtoC, PtoD
End Sub
```

Por supuesto, para visualizar mejor el modelo obtenido se debe cambiar a un punto de vista tridimensional. La propia vista creada en el epígrafe anterior, resulta adecuada.

7.4 – Creación de modelos de superficies.

Los modelos de superficies son una mejor aproximación a la realidad que los de alambres. A diferencia de estos, representan los objetos no sólo por sus aristas, sino por las superficies o caras de los mismos.

Dentro de los componentes más comunes de los modelos de superficies, tenemos las caras y las mallas tridimensionales, las cuales se analizarán a continuación.

Caras tridimensionales.

Las caras tridimensionales son sectores planos limitados por tres o cuatro puntos, tal como lo muestra la Fig. 7.3. Se crean con el método `Add3DFace`, que requiere como parámetros cuatro puntos, dados por arreglos de tres valores de tipo Double, cada uno. Si las coordenadas del cuarto punto se hacen coincidir con las del tercero, se obtiene una cara de tres vértices, lo cual es lo más apropiado para aproximar superficies curvas.

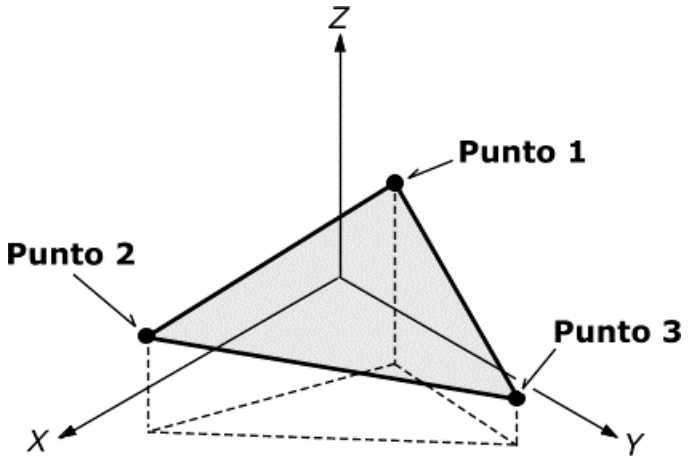


Fig. 7.3 – Cara tridimensional.

En el ejemplo que se muestra a continuación, se crea un modelo compuesto por cuatro caras tridimensionales, que representa al tetraedro modelado mediante alambres en el epígrafe anterior.

```
Sub MallaTetraedro()
  ' Crea el modelo de superficies de
  ' un tetraedro regular
  Dim PtoA(0 To 2) As Double, PtoB(0 To 2) As Double, _
      PtoC(0 To 2) As Double, PtoD(0 To 2) As Double
  PtoA(0) = 0: PtoA(1) = 0: PtoA(2) = 0
  PtoB(0) = 100: PtoB(1) = 0: PtoB(2) = 0
  PtoC(0) = 50: PtoC(1) = 86.6025: PtoC(2) = 0
  PtoD(0) = 50: PtoD(1) = 28.8675: PtoD(2) = 81.6497
  ThisDrawing.ModelSpace.Add3DFace PtoA, PtoB, PtoC, PtoC
  ThisDrawing.ModelSpace.Add3DFace PtoA, PtoB, PtoD, PtoD
  ThisDrawing.ModelSpace.Add3DFace PtoB, PtoC, PtoD, PtoD
  ThisDrawing.ModelSpace.Add3DFace PtoA, PtoC, PtoD, PtoD
End Sub
```

Mallas tridimensionales.

Otro constituyente de gran relevancia dentro de los modelos de superficies son las mallas tridimensionales. Estas se crean con el método `Add3DMesh`, de los objetos `ModelSpace` o `PaperSpace`. Lleva como argumento la cantidad de nodos en las dos direcciones de la malla, y las coordenadas de todos los nodos. Las coordenadas se establecen a través de un arreglo de (N*M*3 – 1) valores de tipo `Double`, donde M y N son la cantidad de nodo en las dos direcciones (ver. Fig. 7.4).

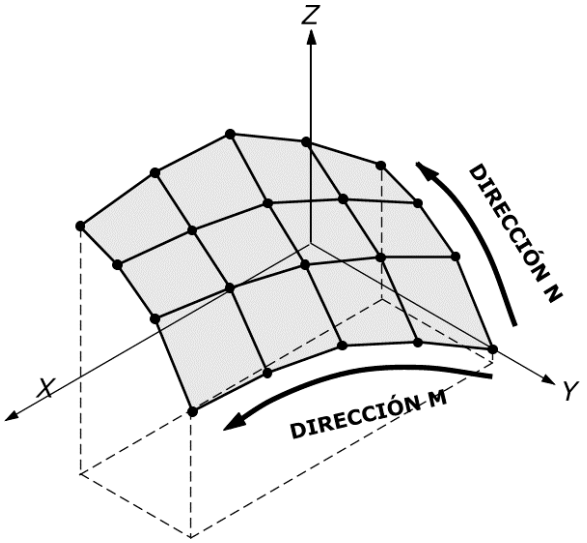


Fig. 7.4 – Malla tridimensional.

En el siguiente código se muestra la creación de la malla correspondiente a un paraboloide hiperbólico, dado por la función $z = (x/5)^2 - (y/5)^2$, en los rangos $-10 = x = = 10$; $-10 = y = 10$.

```
Sub MallaParabHiperb()
  ' Crea una malla tridimensional que
  ' representa un paraboloide hiperbólico
  Dim Puntos(0 To 1322) As Double
  Dim X As Integer, Y As Integer, C As Integer
  Dim Z As Double
  C = 0
  For X = -10 To 10
    For Y = -10 To 10
      Puntos(C) = X
      Puntos(C + 1) = Y
      Puntos(C + 2) = (X / 5) ^ 2 - (Y / 5) ^ 2
      C = C + 3
    Next Y
  Next X
  ThisDrawing.ModelSpace.Add3DMesh 21, 21, Puntos
End Sub
```

En el ejemplo, se utiliza la variable Puntos para almacenar los 1323 (de 0 a 1322) valores de coordenadas de los puntos del arreglo (3 coordenadas por cada uno de los 21x21 = 441puntos de la malla). Los intervalos de x y de y, se dividen en 21 valores (-10, -9, ... , 9, 10). Se emplean dos ciclos repetitivos anidados para establecer las coordenadas de cada punto.

7.5 – Creación de modelos sólidos.

Los modelos sólidos no sólo son los que mejor representan la realidad y los que se crean con mayor facilidad en AutoCAD, sino que también son los que cuentan con mayor número de métodos para su creación dentro de la Automatización ActiveX.

Al igual que en el trabajo desde la consola, hay tres formas básicas de crear sólidos: a través de formas estereométricas básicas: esferas, paralelepípedos, conos, cilindros, etc.; a través de la extrusión de un objeto bidimensional a lo largo de una ruta; y a través de la revolución de un objeto bidimensional alrededor de un eje.

Sólidos elementales.

Para la creación de sólidos mediante figuras elementales, AutoCAD proporciona los siguientes métodos:

`AddBox`	Crea un paralelepípedo. Requiere cuatro parámetros: las coordenadas del punto central del paralelepípedo, la longitud (dimensión en la dirección del eje *x*), el ancho (eje *y*) y la altura (eje *z*).
`AddWedge`	Crea una cuña. Requiere cuatro parámetros: las coordenadas del punto central, la longitud, el ancho y la altura.
`AddSphere`	Crea una esfera. Requiere dos parámetros: las coordenadas del centro de la esfera y su radio.
`AddCylinder`	Crea un cilindro de base circular. Requiere tres parámetros: las coordenadas del centro, el radio y la altura.
`AddEllipticalCylinder`	Crea un cilindro de base elíptica. Requiere cuatro parámetros: las coordenadas del centro, el radio mayor, el radio menor y la altura.
`AddCone`	Crea un cono de base circular. Requiere tres parámetros: las coordenadas del centro, el radio de la base y la altura.
`AddEllipticalCone`	Crea un cono de base elíptica. Requiere cuatro parámetros: las coordenadas del centro, el radio mayor de la base, el radio menor de la base y la altura.
`AddTorus`	Crea un toroide. Requiere tres parámetros: las coordenadas del centro, el radio del toroide, y el radio del tubo del toroide.

En los métodos anteriores, todas las coordenadas se representan por variables de arreglo de tres valores de tipo `Double`, y las dimensiones (longitudes, radios, etc.), mediante variables de tipo `Double`. Los sólidos, independientemente de su tipo, siempre se contienen en objetos de tipo Acad3DSolid.

Para ejemplificar lo anterior, se muestra el siguiente código, donde se crea un cubo, sobre el cual se encuentra un cilindro de base circular. La arista del cubo es igual a 20, el radio del cilindro a 10, y su altura a 30 unidades.

Debe destacarse que los puntos de referencia, usados por los métodos empleados, a diferencia de lo que ocurre en el trabajo con la consola, no coinciden con un vértice o una cara, sino que están situados en el centro del volumen de la figura (ver Fig. 7.5).

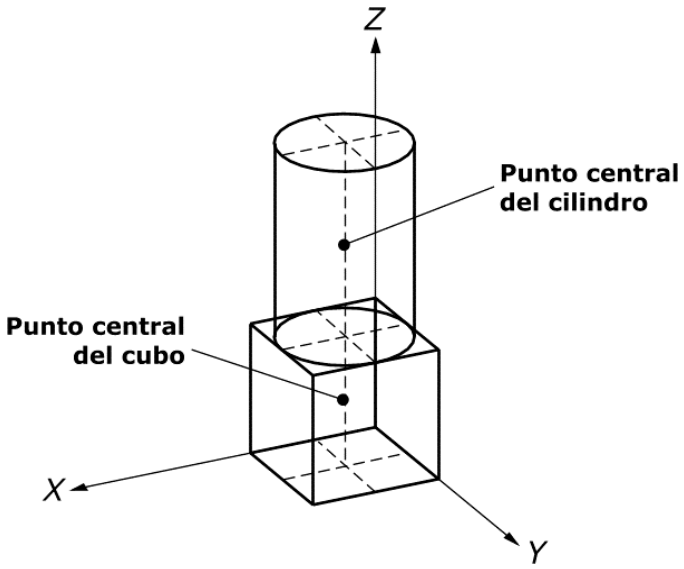


Fig. 7.5 – Modelo de sólidos elementales.

```
Sub SolidosBasicos()
  ' Crea un modelo sólido compuesto por un cubo
  ' y un cilindro
  Dim Pto1(0 To 2) As Double, Pto2(0 To 2) As Double
  Pto1(0) = 10: Pto1(1) = 10: Pto1(2) = 10
  ThisDrawing.ModelSpace.AddBox Pto1, 20, 20, 20
  Pto2(0) = 10: Pto2(1) = 10: Pto2(2) = 35
  ThisDrawing.ModelSpace.AddCylinder Pto2, 10, 30
End Sub
```

Sólidos creados por extrusión.

Los sólidos por extrusión se crean con los métodos `AddExtrudedSolid` y `AddExtrudedSolidAlongPath`. El primero permite obtener el sólido mediante la extrusión a lo largo de una línea recta, cuya distancia se especifica. Requiere tres parámetros: la región que define el área que se va a someter a la extrusión (debe ser un objeto de tipo `AcadRegion` o `Variant` creado con el método `AddRegion`), la altura de la extrusión y el ángulo de inclinación de las generatrices.

A continuación se muestra el código de creación de un cono truncado, por extrusión, a partir de una circunferencia. Véase que antes de proceder a la extrusión, es necesario convertir el circulo que servirá como perfil, en región.

```
Sub SolidoExtrusion()
  ' Crea un cono truncado por extrusión
  ' de una circunferencia
  Dim Circulo(0 To 0) As AcadEntity
  Dim Region As Variant
  Dim Pto1(0 To 2) As Double
```

```
  Pto1(0) = 10: Pto1(1) = 10: Pto1(2) = 0
  Set Circulo(0) = ThisDrawing.ModelSpace. _
                     AddCircle(Pto1, 10)
  Region = ThisDrawing.ModelSpace.AddRegion(Circulo)
  ThisDrawing.ModelSpace.AddExtrudedSolid _
                       Region(0), 50, 5 * 3.1416 / 180
End Sub
```

El segundo método es similar al anterior, con la diferencia de que en lugar de la altura y el ángulo, se le especifica un objeto que servirá como ruta de extrusión. Para tal fin puede servir solamente objetos de tipo polilínea, círculo, elipse, *spline* o arco.

Sólidos por revolución.

La creación de sólidos por revolución, se lleva a cabo con el método `AddRevolvedSolid`. A éste se le pasan como parámetros el perfil que se va a revolucionar (debe ser una región), el punto de origen del eje de revolución, un punto que sirve como vector de dirección del eje y, por último, el ángulo que barrerá la revolución.

En el siguiente ejemplo, se muestra como crear un cilindro hueco a partir de un perfil de forma rectangular, mediante su revolución alrededor de un eje.

```
Sub SolidoRevolucion()
  ' Crea un cilindro hueco mediante la revolución
  ' de un rectángulo alrededor de un eje
  Dim Rect(0 To 0) As AcadEntity
  Dim Region As Variant
  Dim Puntos(0 To 9) As Double
  Dim Eje1(0 To 2) As Double, Eje2(0 To 2) As Double
  Puntos(0) = 10: Puntos(1) = 10
  Puntos(2) = 50: Puntos(3) = 10
  Puntos(4) = 50: Puntos(5) = 30
  Puntos(6) = 10: Puntos(7) = 30
  Puntos(8) = 10: Puntos(9) = 10
  Set Rect(0) = ThisDrawing.ModelSpace. _
                 AddLightWeightPolyline(Puntos)
  Region = ThisDrawing.ModelSpace.AddRegion(Rect)
  Eje1(0) = 0: Eje1(1) = 50: Eje1(2) = 0
  Eje2(0) = 1: Eje2(1) = 0: Eje2(2) = 0
  ThisDrawing.ModelSpace.AddRevolvedSolid _
               Region(0), Eje1, Eje2, 3.1416 * 2
End Sub
```

Nótese que para crear una región debe utilizarse polilíneas ligeras (*lightweight polylines*) en lugar de polilíneas comunes. También es importante que se preste atención a que la variable `Eje2` no es en realidad un punto, sino un vector que indica la dirección del eje, a partir del punto definido por `Eje1`.

7.6 – Edición de sólidos.

La edición de sólidos comprende un grupo de operaciones que permiten obtener, a partir de sólidos simples, otros más complejos. Dentro de estas tenemos las operaciones booleanas, la interferencia y el corte, las cuales serán examinadas a continuación.

Operaciones booleanas.

Permiten llevar a cabo la unión, la intersección y la sustracción de los sólidos. Se llevan a cabo con el método `Boolean`, el cual requiere dos parámetros: una constante que define el tipo de operación (debe tomar uno de los valores siguientes: `acUnion`, `acSubtraction`, `acIntersection`) y el sólido con el que se realizará la operación. El método `Boolean` pertenece siempre a uno de los sólidos que participará en la operación y, en el caso de la sustracción, debe ser aquel al que se le sustraerá una parte.

En el siguiente ejemplo, se muestra la creación de dos sólidos: un cubo y una esfera. Posteriormente, se le sustrae el segundo al primero.

```
Sub Sustraccion()
  ' Aplica la sustracción de una esfera a un cubo
  Dim Pto1(0 To 2) As Double, Pto2(0 To 2) As Double
  Dim Cubo As Acad3DSolid, Esfera As Acad3DSolid
  Pto1(0) = 10: Pto1(1) = 10: Pto1(2) = 10
  Pto2(0) = 20: Pto2(1) = 20: Pto2(2) = 20
  Set Cubo = ThisDrawing.ModelSpace.AddBox( _
              Pto1, 20, 20, 20)
  Set Esfera = ThisDrawing.ModelSpace.AddSpher( _
                Pto2, 10)
  Cubo.Boolean acSubtraction, Esfera
End Sub
```

Interferencia.

La interferencia permite obtener un nuevo sólido que corresponde a la parte común de otros dos. Es similar a la operación booleana intersección, pero no modifica los sólidos originales. Se realiza con el método `CheckInterference`, el cual necesita dos parámetros: el sólido con el cual interferirá aquel al cual le estamos aplicando el método, y un valor booleano que indica si se creará el sólido que surge como resultado de la interferencia.

A continuación se muestra un ejemplo donde se obtiene la interferencia entre el cubo y la esfera vistos en el ejemplo anterior. Al sólido resultante se le asigna color rojo.

```
Sub Interferencia()
  ' Obtiene la interferencia de una esfera a un cubo
  Dim Pto1(0 To 2) As Double, Pto2(0 To 2) As Double
  Dim Cubo As Acad3DSolid, Esfera As Acad3DSolid, _
      Interf As Acad3DSolid
  Pto1(0) = 10: Pto1(1) = 10: Pto1(2) = 10
  Pto2(0) = 20: Pto2(1) = 20: Pto2(2) = 20
  Set Cubo = ThisDrawing.ModelSpace.AddBox( _
              Pto1, 20, 20, 20)
  Set Esfera = ThisDrawing.ModelSpace.AddSphere( _
                Pto2, 10)
  Set Interf = Cubo.CheckInterference(Esfera, True)
  Interf.Color = acRed
End Sub
```

Corte.

El corte permite separar un sólido por un plano dado. Se lleva a cabo con el método `SliceSolid`, que requiere cuatro parámetros, los tres primeros son los puntos que definen el plano de corte. El cuarto, es un valor booleano que, si toma valor `False`, elimina la parte del sólido que está en la parte positiva del plano de corte; si toma valor `True`, no la elimina, sino que la asigna como resultado del método.

En el siguiente código se ejemplifica como cortar una esfera por la mitad, utilizando un plano horizontal.

```
Sub Corte()
  ' Corta una esfera por la mitad
  Dim Pto1(0 To 2) As Double, Pto2(0 To 2) As Double, _
      Pto3(0 To 2) As Double
  Dim Esfera As Acad3DSolid
  Pto1(0) = 10: Pto1(1) = 10: Pto1(2) = 10
  Pto2(0) = 20: Pto2(1) = 10: Pto2(2) = 10
  Pto3(0) = 10: Pto3(1) = 20: Pto3(2) = 10
  Set Esfera = ThisDrawing.ModelSpace.AddSphere( _
                Pto1, 10)
  Esfera.SliceSolid Pto1, Pto2, Pto3, False
End Sub
```

7.7 – Edición en el espacio tridimensional.

Dentro de la edición de objetos en el espacio tridimensional tenemos un conjunto de herramientas, algunas de las cuales son comunes a los objetos bidimensionales, mientras otras son específicas. En el primer caso tenemos el copiado, el movimiento y la rotación alrededor del eje z, entre otros. En el segundo caso están la rotación, la reflexión y los arreglos tridimensionales.

Rotación tridimensional.

Se realiza con el método Rotate3D, el cual requiere dos puntos que definan el eje de rotación, y el ángulo a rotar. Es importante recordar que el sentido positivo del ángulo lo define el orden en que se especifiquen los dos puntos del eje, y puede determinarse mediante la regla del sacacorchos.

En el siguiente ejemplo, se muestra la creación de un cilindro y su rotación alrededor de un eje definido por dos puntos.

```
Sub Rotacion3D()
  ' Rota un cilindro 90 grados alrededor de un eje
  ' definido por los puntos Pto1 y Pto2
  Dim Pto1(0 To 2) As Double, Pto2(0 To 2) As Double
  Dim Cilindro As Acad3DSolid
  Pto1(0) = 10: Pto1(1) = 10: Pto1(2) = 10
  Pto2(0) = 20: Pto2(1) = 10: Pto2(2) = 10
  Set Cilindro = ThisDrawing.ModelSpace.AddCylinder( _
                   Pto1, 10, 50)
  Cilindro.Rotate3D Pto1, Pto2, 3.1416 / 2
End Sub
```

Reflexión tridimensional.

Permite reflejar un objeto a través de un plano arbitrario definido por tres puntos. Se realiza con el método Mirror3D, que requiere, como parámetros, los tres puntos que definen al plano de reflexión. A continuación se muestra un ejemplo de reflexión de una cuña a través de un plano.

```
Sub Reflexion3D()
  ' Aplica una reflexión a una cuña, mediante el
  ' plano definido por los puntos Pto2, Pto3 y Pto4
  Dim Pto1(0 To 2) As Double, Pto2(0 To 2) As Double, _
      Pto3(0 To 2) As Double, Pto4(0 To 2) As Double
  Dim Cunya As Acad3DSolid
  Pto1(0) = 20: Pto1(1) = 10: Pto1(2) = 15
  Set Cunya = ThisDrawing.ModelSpace.AddWedge( _
                Pto1, 40, 20, 30)
  Pto2(0) = 0: Pto2(1) = 0: Pto2(2) = 0
  Pto3(0) = 0: Pto3(1) = 10: Pto3(2) = 0
  Pto4(0) = 0: Pto4(1) = 0: Pto4(2) = 10
  Cunya.Mirror3D Pto2, Pto3, Pto4
End Sub
```

Arreglos tridimensionales.

Tal como se vio previamente, el método `ArrayRectangular` permite crear arreglos tridimensionales. Para ello sólo es necesario especificarle una cantidad de capas superior a uno, y una distancia entre capas mayor que cero, tal como muestra el siguiente ejemplo.

```
Sub Arreglo3D()
  ' Crea un arreglo tridimensional a partir
  ' de una caja
  Dim Pto1(0 To 2) As Double
  Dim Caja As Acad3DSolid
  Dim CantFilas As Long, DistFilas As Double, _
      CantCols As Long, DistCols As Double, _
      CantCapas As Long, DistCapas As Double
  Pto1(0) = 10: Pto1(1) = 5: Pto1(2) = 3
  Set Caja = ThisDrawing.ModelSpace.AddBox( _
                Pto1, 20, 10, 6)
  CantFilas = 3: DistFilas = 30
  CantCols = 4: DistCols = 40
  CantCapas = 5: DistCapas = 26
  Caja.ArrayRectangular CantFilas, CantCols, _
         CantCapas, DistFilas, DistCols, DistCapas
End Sub
```

Ejemplo resuelto.

Elaborar una macro que permita, crear una T sin bridas, para tubería, dados el diámetro exterior (D), el interior (d) y la longitud (L).

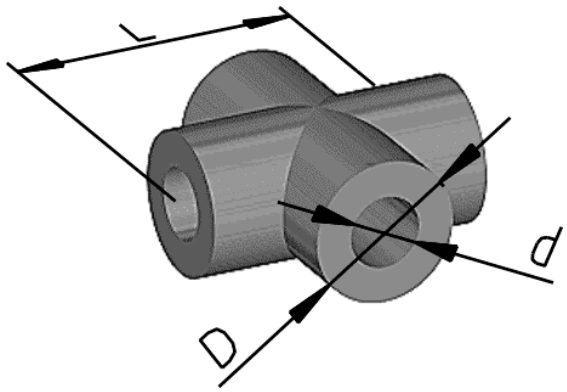


Fig. 7.6 - Dimensiones de una T, sin bridas, para tubería.

Solución.

El primer paso, en la solución del problema, es tomar los datos necesarios, para lo cual se utilizan los métodos `GetPoint` y `GetReal`, del objeto `Utility`. Nótese que las coordenadas del punto de inserción se almacenan inicialmente en una variable de tipo `Variant`, que es el valor que devuelve el método `GetPoint`. Posteriormente, se pasan a una variable de tipo arreglo de tres dimensiones de `Double` (la forma estándar de almacenar un punto), para usarlo en la creación y edición de entidades.

A continuación se crean dos cilindros idénticos, que corresponden a los contornos exteriores de la T. Uno de ellos se rota 90° alrededor de dos puntos que indican un eje en dirección de x. Finalmente, ambos se unen con la opción `acUnion` del método `Boolean`.

Un proceso idéntico se lleva a cabo con los dos cilindros correspondientes al contorno interior. Luego, se sustraen utilizando, también, el método `Boolean`, pero ahora con la opción `acSubtraction`.

Finalmente, se rota el sólido obtenido, sobre el eje y, para que la T resultante quede paralela al plano xy. A continuación se muestra el código de la macro.

```
Sub TeSinBridas()
  ' Crea una T sin bridas, para tubería
  ' Definición de variables
  Dim Centro As Variant
  Dim Punto0(0 To 2) As Double, Punto1(0 To 2) As Double
  Dim DiamInt As Double, DiamExt As Double, _
      Longitud As Double
  Dim CilExt1 As Acad3DSolid, _
      CilExt2 As Acad3DSolid, _
      CilInt1 As Acad3DSolid, _
      CilInt2 As Acad3DSolid
  ' Toma de datos del usuario
  Centro = ThisDrawing.Utility.GetPoint(, _
           "Establezca el punto central del resorte: ")
  Punto0(0) = Centro(0)
  Punto0(1) = Centro(1)
  Punto0(2) = Centro(2)
  DiamExt = ThisDrawing.Utility.GetReal( _
            "Establezca el diámetro exterior: ")
  DiamInt = ThisDrawing.Utility.GetReal( _
            "Establezca el diámetro interior: ")
  Longitud = ThisDrawing.Utility.GetReal( _
             "Establezca la longitud de la T: ")
  ' Creación de los cilindros exteriores
  Set CilExt1 = ThisDrawing.ModelSpace.AddCylinder( _
```

```
                Punto0, DiamExt / 2, Longitud)
  Set CilExt2 = ThisDrawing.ModelSpace.AddCylinder( _
                Punto0, DiamExt / 2, Longitud)
  Punto1(0) = Punto0(0) + 1
  Punto1(1) = Punto0(1)
  Punto1(2) = Punto0(2)
  CilExt2.Rotate3D Punto0, Punto1, 3.1416 / 2
  CilExt1.Boolean acUnion, CilExt2
  ' Creación de los cilindros interiores
  Set CilInt1 = ThisDrawing.ModelSpace.AddCylinder( _
                Punto0, DiamInt / 2, 1.5 * Longitud)
  Set CilInt2 = ThisDrawing.ModelSpace.AddCylinder( _
                Punto0, DiamInt / 2, 1.5 * Longitud)
  CilInt2.Rotate3D Punto0, Punto1, 3.1416 / 2
  CilInt1.Boolean acUnion, CilInt2
  ' Subtracción de ambos cuerpos
  CilExt1.Boolean acSubtraction, CilInt1
  ' Rotación de la T
  Punto1(0) = Punto0(0)
  Punto1(1) = Punto0(1) + 1
  Punto1(2) = Punto0(2)
  CilExt1.Rotate3D Punto0, Punto1, 3.1416 / 2
End Sub
```

Preguntas y ejercicios propuestos.

1. ¿Cómo se manipulan los sistemas de coordenadas definidos por el usuario?

2. ¿Cómo se manipulan los puntos de vista?

3. ¿Qué semejanzas y diferencias tiene el trabajo con modelos de malla de alambre y con entidades bidimensionales?

4. ¿Qué métodos se vieron para crear mallas superficiales?

5. ¿Qué métodos se vieron para la creación y edición de modelos sólidos?

6. Escriba un programa para crear la malla de la superficie de un paraboloide elíptico, dado por la ecuación $z = 2x^2 + 4y^2$.

7. Escriba un programa para crear el modelo sólido de un remache de cabeza semiesférica, dadas sus dimensiones.

CAPÍTULO 8
Manipulación de Eventos

OBJETIVO

Crear subrutinas para manipular los eventos generados por AutoCAD, mediante el uso de Visual Basic for Applications.

8.1 – Elementos básicos.

Los eventos son mensajes que envía AutoCAD para notificar que algo a ocurrido. Este algo puede ser de naturaleza muy variada. Puede ser, por ejemplo, una acción generada por el usuario, como hacer clic sobre un control o presionar una tecla. También puede ser una acción realizada por la propia aplicación como la apertura de un documento.

Es posible escribir subrutinas y asociarlas con los eventos, de forma que se ejecuten cada vez que ocurra una llamada a uno de ellos. A estas subrutinas se les llama manipuladores de eventos. Los eventos pueden pasar determinada información a sus manipuladores, en forma de parámetros.

Los eventos de AutoCAD están organizados en tres niveles:

- Eventos a nivel de aplicación: Responden a los cambios o las acciones del propio programa, tal como la apertura, el guardado, la impresión y el cierre de documentos, la creación de nuevos documentos, el lanzamiento de comandos, la carga y descarga de aplicaciones ARX o de AutoLISP, y los cambios en las variables del sistema o en la ventana principal del programa.

- Eventos a nivel de documento: Responden a los cambios realizados en el documento o en su contenido. Incluyen acciones tales como la adición, borrado o cambio de objetos, la activación de los menús, los cambios en los conjuntos de selección y los cambios en la ventana del dibujo.

- Eventos a nivel de objeto: Responden a acciones realizadas sobre los objetos. Actualmente sólo hay un evento de este tipo, que se lanza cuando se modifica el objeto.

También Visual Basic for Applications es capaz de generar eventos para un grupo de acciones realizadas por el usuario sobre los controles, en los cuadros de diálogos, tales como hacer clic o doble clic, mover el ratón, presionar o liberar una tecla, etc. Esos eventos serán vistos en el próximo capítulo.

Es importante tener siempre en cuanta que los eventos sólo proporcionan información sobre el estado de las actividades que están teniendo lugar en AutoCAD. Los manipuladores de eventos muchas veces se ejecutan durante el procesamiento de acciones o comandos, por lo cual hay ciertas precauciones que deben tenerse en cuenta para que la manipulación de eventos tenga lugar con seguridad. A saber:

- No confiar en la secuencia de los eventos. Los eventos no siempre ocurren en la secuencia que uno espera. Por ejemplo, al lanzar el comando `OPEN` se generan los eventos `BeginCommand`, `BeginOpen`, `EndOpen` y `EndCommand`; pero no hay ninguna seguridad de que sean generados en esa misma secuencia. Lo único que se puede asegurar que un evento `Begin` siempre es anterior a su correspondiente evento `End`.

- No confiar en la secuencia de las operaciones. Por ejemplo, si se borra el objeto1 y luego el objeto2, se lanzará un evento `ObjectErased` para cada uno de ellos, pero no hay garantía de que se reciban en ese orden.

- No intentar ninguna función interactiva desde un manipulador de eventos. Como AutoCAD está en medio de la ejecución de un comando, el uso de métodos como `GetPoint`, `GetEntity` y `GetDistance`, o del método `SendCommand`, puede causar serios problemas. Lo anterior se aplica también para los cuadros de diálogos, aunque se exceptúan los cuadros de mensaje y alerta.

- No modificar propiedades del objeto que está lanzando el evento. No obstante otros objetos sí pueden ser modificados. La lectura de propiedades sí es segura en todos los casos.

- No realizar ninguna acción desde el manipulador de un evento, que genere el mismo evento. Si esto ocurre, se puede caer en un lazo infinito.

- Tener presente que mientras se está mostrando un cuadro de diálogo modal, no se genera ningún evento, salvo los generados por los propios controles del diálogo.

8.2 – Eventos de nivel de aplicación.

Los eventos de nivel de aplicación son:

- `AppActivate`: Es activado justo antes de que se active la ventana principal de la aplicación.
- `AppDeactivate`: Es activado justo antes de que se desactive la ventana principal de la aplicación.
- `ARXLoaded`: Es activado cuando una aplicación ARX ha sido cargada.
- `ARXUnloaded`: Es activado cuando una aplicación ARX ha sido descargada.

- `BeginCommand`: Es activado inmediatamente después de que es lanzado un comando, pero antes de que se complete su ejecución.
- `BeginFileDrop`: Es activado cuando un archivo es colocado en la ventana principal de la aplicación.
- `BeginLISP`: Es activado inmediatamente después de que AutoCAD recibe una petición para evaluar una expresión de LISP.
- `BeginModal`: Es activado justo antes de que se muestre un cuadro de diálogo modal.
- `BeginOpen`: Es activado inmediatamente después de que AutoCAD recibe la petición de abrir un documento existente.
- `BeginPlot`: Es activado inmediatamente después de que AutoCAD recibe una petición de imprimir un documento.
- `BeginQuit`: Es activado justo antes de que termine una sesión de AutoCAD.
- `BeginSave`: Es activado inmediatamente después de que AutoCAD recibe una petición de guardar un documento.
- `EndCommand`: Es activado inmediatamente después de que se completa la ejecución de un comando.
- `EndLISP`: Es activado luego de completarse la evaluación de una expresión de LISP.
- `EndModal`: Es activado luego de que es cerrado un cuadro de diálogo modal.
- `EndOpen`: Es activado inmediatamente después de que AutoCAD termina de abrir un dibujo existente.
- `EndPlot`: Es activado luego de que un documento es enviado a la impresora.
- `EndSave`: Es activado cuando AutoCAD ha terminado de guardar un dibujo.
- `LISPCancelled`: Es activado cuando se cancela la evaluación de una expresión de LISP.
- `NewDrawing`: Es activado exactamente antes de que se crea un nuevo dibujo.
- SysVarChanged: Es activado cuando se cambia el valor de una variable de sistema.
- WindowChanged: Es activado cuando hay algún cambio en la ventana principal de la aplicación.
- WindowMovedOrResized: Es activado después de que la ventana principal de la aplicación es movida o cambiada de tamaño.

Los eventos de nivel de aplicación no son persistentes en VBA para AutoCAD. Esto significa que ellos no se habilitan automáticamente al cargar un proyecto. Por lo tanto, estos eventos deben ser habilitados para VBA (y para otros controladores de Automatización ActiveX).

Antes de utilizar eventos de nivel de aplicación debe crearse un nuevo módulo de clase y declarar, en él, un objeto de tipo `AcadApplication,` con eventos. Por ejemplo para crear un manipulador para el evento `BeginOpen`, que muestre un mensaje con el nombre del archivo que será abierto, se procede de la siguiente forma:

1. Se inserta un nuevo módulo de clase en el proyecto (utilizando la opción *Class Module*, del menú *Insert*, en el IDE de VBA), y, en la ventana de propiedades, se le cambia el nombre a `clsAppEve`.

2. En el módulo de clase recién creada se escribe el siguiente código:

```
Public WithEvents App As AcadApplication

Private Sub App_BeginOpen(FileName As String)
  MsgBox FileName
End Sub
```

   La primera línea indica la definición de una instancia pública, de tipo `AcadApplication`, con eventos, a la cual se le ha llamado `App`. El procedimiento que le sigue es el manipulador del evento `BeginOpen`, para el objeto `App` definido anteriormente. Como se puede ver, a este procedimiento se le pasa el parámetro `FileName`, que contiene el nombre del archivo abierto. La única acción que realiza el manipulador del evento es mostrar un mensaje con dicho nombre.

3. En el código correspondiente al objeto ThisDrawing, se escribe el código siguiente:

```
Dim MiAppEve As New clsAppEVe

Public Sub InitializeEvents()
  Set MiAppEve.App = GetObject(, "AutoCAD.Application")
End Sub
```

   Mediante la primera línea se declara la variable `MiAppEve`, que es una instancia de la clase `clsAppEve`. Luego, se añade el procedimiento público (macro) `InitializeEvents`, donde se le asigna, a la variable `App` del objeto `MiAppEve`, la propia aplicación de AutoCAD. Esto pudiera hacerse con la propiedad `ThisDrawing.Application`, en lugar de con la función `GetObject`, pero entonces la instancia de perdería al cerrar el dibujo activo. La macro anterior debe ser ejecutada desde AutoCAD para que comience a hacer efecto la manipulación del evento.

8.3 – Eventos de nivel de documento.

Los eventos de nivel de documento son persistentes, o sea, que se habilitan automáticamente al cargar el proyecto de VBA. No obstante, para otros controladores como Visual Basic estándar, no ocurre así, por lo cual deben ser previamente habilitados.

Existe un amplio número de eventos de nivel de documento, dentro de los que se tienen:

- `Activate`: Es activado cuando la ventana de documento se activa.
- `BeginClose`: Es activado justo antes de que un documento sea cerrado.
- `BeginCommand`: Es activado inmediatamente después de que un comando es lanzado.
- `BeginDoubleClick`: Es activado cuando el usuario realiza un clic doble sobre un objeto del dibujo.
- `BeginLISP`: Es activado inmediatamente después de que AutoCAD recibe una petición de evaluar una expresión de LISP.
- `BeginPlot`: Es activado inmediatamente después de que AutoCAD recibe una petición de imprimir un dibujo.
- `BeginRightClick`: Es activado después de que el usuario hace clic derecho sobre la ventana del dibujo.
- `BeginSave`: Es activado inmediatamente después de que AutoCAD recibe una petición de guardar el dibujo.
- `BeginShortcutMenuCommand`: Es activado luego de que el usuario hace clic derecho sobre la ventana del dibujo y antes de que aparezca el menú contextual, durante la ejecución de un comando.
- `BeginShortcutMenuDefault`: Es activado luego de que el usuario hace clic derecho sobre la ventana del dibujo y antes de que aparezca el menú contextual, en modo predeterminado.
- `BeginShortcutMenuEdit`: Es activado luego de que el usuario hace clic derecho sobre la ventana del dibujo y antes de que aparezca el menú contextual, durante la edición de una entidad.
- `BeginShortcutMenuGrip`: Es activado luego de que el usuario hace clic derecho sobre la ventana del dibujo y antes de que aparezca el menú contextual, durante la edición por agarre de una entidad.
- `BeginShortcutMenuOsnap`: Es activado luego de que el usuario hace clic derecho sobre la ventana del dibujo y antes de que aparezca el menú contextual, en el modo Osnap.
- `Deactivate`: Es activado cuando la ventana del dibujo es desactivada.
- `EndCommand`: Es activado inmediatamente después que se completa la ejecución de un comando.
- `EndLISP`: Es activado luego de que se completa la evaluación de una expresión de LISP.
- `EndPlot`: Es activado luego de que un documento ha sido enviado a la impresora.
- `EndSave`: Es activado cuando AutoCAD ha terminado de guardar el dibujo.
- `EndShortcutMenu`: Es activada luego de que aparece el menú contextual.
- `LayoutSwitched`: Es activado cuando el usuario cambia a un nuevo espacio de papel (*layout*).

- `LISPCancelled`: Es activado cuando se cancela la evaluación de una expresión de LISP.
- `ObjectAdded`: Es activado cuando se adiciona un nuevo objeto al dibujo.
- `ObjectErased`: Es activado cuando se elimina un objeto del dibujo.
- `ObjectModified`: Es activado cuando un objeto del dibujo está siendo modificado.
- `SelectionChanged`: Es activado cuando la selección actual experimenta algún cambio.
- `WindowChanged`: Es activado cuando hay algún cambio en la ventana del documento.
- `WindowMovedOrResized`: Es activado justo después de que la ventana del documento ha sido movida o cambiada de tamaño.

Para escribir manipuladores para los eventos de nivel de documento, se selecciona el objeto `AcadDocument` de la lista desplegable de objetos, en la hoja de código. En la lista desplegable de procedimientos, aparecerán todos los eventos disponibles.

Debe tenerse siempre presente que los manipuladores de eventos, creados así, pertenecen al documento activo.

En el siguiente ejemplo, se muestra el código para manipular los eventos `BeginShortcutMenuDefault` para que se agregue un nuevo elemento de menú contextual y `EndShortcutMenu` para que sea eliminado al terminarse su ejecución. El nuevo menú ejecutará una macro llamada `AcercaDe` la que, a su vez, mostrará un mensaje con los créditos de este libro.

```
Private Sub AcadDocument_BeginShortcutMenuDefault( _
            ShortcutMenu As AutoCAD.IAcadPopupMenu)
  Dim NombreMacro As String
  On Error Resume Next
  sMacro = "-vbarun AcercaDe" + Chr(32)
  ShortcutMenu.AddMenuItem 0, "Acerca de...", NombreMacro
End Sub

Private Sub AcadDocument_EndShortcutMenu( _
            ShortcutMenu As AutoCAD.IAcadPopupMenu)
  On Error Resume Next
  ShortcutMenu.Item("Acerca de...").Delete
End Sub

Public Sub AcercaDe()
  MsgBox "Programando para AutoCAD con VBA" + Chr(13) + _
         "Ramón Quiza Sardiñas" + Chr(13) + _
         "(c) 2006, Universidad de Matanzas", _
         vbInformation, "Acerca de.."
End Sub
```

Nótese el uso de la sentencia `On Error Resume Next`, en ambos manipuladores de eventos, la cual permite que, en caso de un error, se continúe la ejecución del código de forma transparente para el usuario.

8.3 – Eventos de nivel de objeto.

Existe, de momento, un solo evento a nivel de aplicación, llamado `Modified`, que se activa cuando se modifica el objeto al cual pertenece. Este evento, al igual que los de nivel de aplicación, no son persistentes, por lo cual deben habilitarse antes de poder ser utilizados.

Para poder utilizar el evento `Modified`, debe crearse un módulo de clase y declarar un objeto de tipo `AcadObject` con eventos. En el siguiente ejemplo se crea y habilita un manipulador del evento `Modified` de una región, para que muestre, tras cada modificación, las coordenadas del nuevo centroide. Para ello se deben seguir los siguientes pasos:

1. Insertar un nuevo módulo de clase y cambiarle el nombre a `clsObjEve`.

2. Agregar al módulo de clase anterior, el código siguiente:

```
Public WithEvents Region As AcadRegion

Private Sub Region_Modified _
               (ByVal pObject As AutoCAD.IAcadObject)
  On Error GoTo MIERROR
  Dim C As Variant
  C = pObject.Centroid
  MsgBox "Las coordenadas del centroide son: (" & _
         Format(C(0), "###0.00") & "; " & _
         Format(C(1), "###0.00") & ")."
  Exit Sub
MIERROR:
  MsgBox Err.Description
End Sub
```

En la primera línea del código anterior se declara la propiedad `Region`, de tipo `AcadRegion`. La subrutina siguiente es el manipulador del evento `Modified` para la variable `Region`. En ella se le asignas las coordenadas del centroide (obtenidas con el método `Centroid`) a la variable `C`. Luego se muestran en pantalla mediante el procedimiento `MsgBox`. Véase el uso de la sintaxis `On Error GoTo`, para manipular posibles errores.

3. Agregar, en la hoja de código de `ThisDrawing`, lo siguiente:

```
Dim MiObjEve As New clsObjEve

Public Sub InitializeObjectEvents()
  Dim MisPtos(0 To 11) As Double
  Dim MiPLinea(0 To 0) As AcadEntity
  Dim MiRegion As Variant
  MisPtos(0) = 0:  MisPtos(1) = 0:  MisPtos(2) = 0
  MisPtos(3) = 40: MisPtos(4) = 0:  MisPtos(5) = 0
  MisPtos(6) = 40: MisPtos(7) = 30: MisPtos(8) = 0
  MisPtos(9) = 0:  MisPtos(10) = 0: MisPtos(11) = 0
  Set MiPLinea(0) =  _
             ThisDrawing.ModelSpace.AddPolyline(MisPtos)
  MiRegion = ThisDrawing.ModelSpace.AddRegion(MiPLinea)
  MiPLinea(0).Delete
  Set MiObjEve.Region = MiRegion(0)
End Sub
```

La primera línea corresponde a la definición de un objeto llamado `MiObjEve` de tipo `clsObjEve`. En la macro siguiente se crea una polilínea triangular y luego, se genera una región a partir de ella. Finalmente, se asigna la región creada a la propiedad `Region` del objeto `MiObjEve`. A partir de la ejecución de esta macro, queda habilitado el manipulador del evento `Modified` de la región creada.

Ejemplo resuelto.

Crear un manipulador de evento que permita escribir un archivo de registro con el nombre de cada documento impreso, indicando la fecha y la hora de impresión.

Solución.

En primer lugar es necesario determinar que evento se gestionará para realizar la acción perseguida. Hay dos posibles opciones: los eventos `BeginPlot` y `EndPlot`. Aunque ambos pueden considerarse, el segundo es el más conveniente ya que se activa después que el proceso de impresión ha terminado.

Para escribir los datos al archivo se crea un manipulador de evento, donde se utilizan las instrucciones `Open`, `Write` y `Close`, para abrir, agregar datos y cerrar, respectivamente, el archivo de registro. Para determinar la fecha y hora se utiliza la función `Now` y, mediante la función `Format`, se le da el formato deseado.

Finalmente, el código del manipulador del evento queda de la forma:

```
Private Sub AcadDocument_EndPlot( _
                            ByVal DrawingName As String)
  Open "prints.log" For Append As #1
  Write #1, DrawingName & " " & _
            Format(Now, "dd-mmm-yyyy HH:mm:ss")
  Close #1
End Sub
```

Preguntas y ejercicios propuestos.

1. ¿Qué niveles de eventos manipula AutoCAD? Ponga ejemplos de cada uno.

2. ¿A qué se le llama eventos persistentes y no persistentes? ¿Qué eventos son de cada tipo en AutoCAD?

3. ¿Qué es un manipulador de eventos?

4. Escriba un manipulador de evento para mantener un archivo de registro con los nombres de todos los archivos abiertos, por AutoCAD.

CAPÍTULO 9
Cuadros de Diálogo

OBJETIVO

Crear cuadros de diálogo, mediante Visual Basic for Applications, para utilizarlos en el intercambio de datos con el usuario.

9.1 – Introducción.

Como se dijo ya, formularios es el nombre que se le da a las ventanas de una aplicación durante el proceso de su desarrollo. En VBA, las ventanas son siempre modales, lo que significa que mientras permanezcan abiertas mantienen capturado el foco sobre sí, sin permitir realizar ninguna acción sobre otra. Este tipo de ventana se emplea para intercambiar datos con el usuario, por lo que se les llamas cuadros de diálogo.

En un proyecto de VBA, un formulario se agrega con la opción *UserForm*, del menú *Insert*. VBA dispone de herramientas para el desarrollo rápido de aplicaciones (RAD), que permiten crear los formularios, con suma facilidad, desde un entorno visual. Para editar un formulario, se utiliza el botón de objeto del Explorador de Proyecto.

Todo formulario está compuesto por controles, que son objetos, generalmente visuales, que realizan funciones de interacción con el usuario. Dentro de los controles más comunes tenemos los botones, las etiquetas, las cajas de textos, los cuadros de lista y los cuadros desplegables.

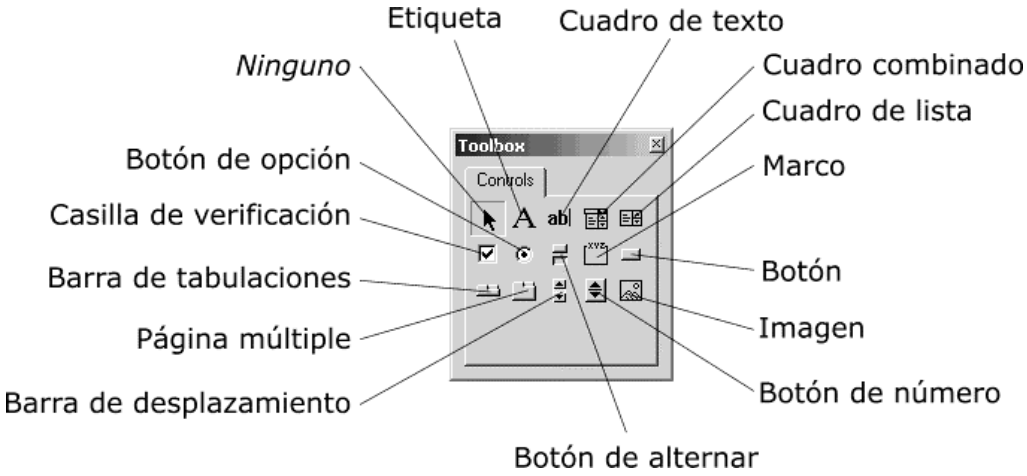


Fig. 9.1 - Controles en el cuadro de herramientas.

Para agregar controles a los formularios, se hace clic sobre el ícono correspondiente, en el cuadro de herramientas (*ToolBox*) (ver Fig. 9.1). Luego, arrastrando el ratón, se indica la posición del control en el formulario, y su tamaño.

Cada control tiene su propio nombre, que debe ser establecido por el programador. Al igual que para los módulos, es recomendable utilizar la notación húngara, para identificar, mediante las tres primeras letras, que tipo de objeto es. En la tabla siguiente se muestran los prefijos recomendados para los controles estándar de VBA.

Control	Prefijo
Formulario	`frm`
Etiqueta (*label*)	`lbl`
Cuadro de texto	`txt`
Botón (*command button*)	`btn`
Casilla de verificación (*checkbox*)	`chk`
Botón de opción	`opt`
Cuadro de lista	`lst`
Cuadro combinado	`cbo`
Marco (*frame*)	`fra`
Barra de tabulaciones	`tab`
Página múltiple	`mpg`
Barra de desplazamiento (*scroll bar*)	`scr`
Botón de número (*spin button*)	`spi`
Botón de alternar (*toggle button*)	`tgl`
Imagen	`img`

Tanto los formularios como los controles, tienen un grupo de propiedades que regulan su aspecto y su comportamiento. Éstas pueden modificarse a través de la ventana de propiedades (ver Fig. 9.2).

Fig. 9.2 - Ventana de propiedades.

También los controles activan eventos, para notificar la realización de acciones sobre ellos. Una gran parte de la programación asociada a los formularios está dirigida, precisamente, a la creación de los manipuladores de estos eventos.

Para crear el manipulador de un evento, se selecciona el nombre del control que activará el evento, en el cuadro de objetos de la hoja de código asociada al formulario. Luego, en el cuadro de eventos de la propia hoja, se selecciona el evento que se desea manipular. Inmediatamente, se agregará a la hoja, el código del manipulador del evento, tal como muestra el siguiente ejemplo:

```
Private Sub btnOK_Click()

End Sub
```

El ejemplo anterior corresponde al manipulador de un evento `Click` del control llamado `btnOK`. Naturalmente, el código que se agrega contiene sólo la declaración y el cierre del procedimiento. Corresponde al programador agregar el código para realizar cualquier acción específica que desee que ejecute el manipulador.

9.2 – Formularios.

Como es lógico, los formularios son el espinazo de los cuadros de diálogo, ya que sirven de contenedor para el resto de los controles. Los formularios poseen un gran número de propiedades, entre las cuales deben destacarse:

- `BackColor` (color de fondo): Establece el color de fondo. Puede tomar cualquier valor defino por una combinación de rojo, verde y azul. Dispone de un grupo de valores estándares que incluyen los colores más comunes (blanco, negro, rojo, etc.) y los utilizados por el sistema operativo (color de ventana, color de la cara de los botones, etc.).

- `BordeStyle` (estilo de borde): Puede tomar dos valores: sin borde (`fmBorderStyleNone`) o con borde simple (`fmBorderStyleSingle`).

- `Caption` (título): Establece el texto que aparecerá en la barra de título del formulario.

- `Font` (fuente): Establece las propiedades de la fuente utilizada en el texto del formulario.

- `ForeColore` (Color de primer plano): Establece el color del texto y demás elementos visuales dibujados sobre el formulario.

- `Height` (altura) y `Width` (ancho): Establece la altura y el ancho del formulario, en puntos (1/72 de pulgada).

- `Left` (izquierda) y `Top` (arriba): Establece la posición del formulario desde la izquierda y desde el borde superior de la pantalla, respectivamente.

- `StartUpPosition` (posición inicial): Establece la posición inicial del formulario. Puede tomar valores de `Manual` (la posición inicial viene dada por las propiedades `Left` y `Top`), `CenterOwner` (el formulario aparece en el centro de la ventana que lo contiene), `CenterScreen` (el formulario aparece en el centro de la pantalla) y `Windows Default` (el formulario aparece en la esquina superior izquierda de la pantalla).

Los principales métodos que se aplican a los formularios están dirigidos a su carga y su visualización. Éstos son:

- `Show`: Muestra el formulario. Tal como se dijo, en VBA los formularios son siempre modales, o sea, que después de ser mostrados, no se ejecuta ningún código siguiente hasta haberlo cerrado. Si al llamar el método `Show`, el formulario no está cargado, el proceso de carga ocurre automáticamente.

- `Hide`: Oculta el formulario, pero no lo descarga de la memoria.

- `Load`: No es en realidad un método, sino una instrucción, cuya sintaxis es: `Load <Objeto>`. Permite cargar un formulario en memoria, pero no lo visualiza.

- `Unload`: Descarga el formulario de la memoria.

Los formularios también poseen eventos los cuales se activan bajo determinadas circunstancias y pueden ser manipulados. Los más notables son:

- `Activate`: Se lanza cuando el formulario pasa a ser la ventana activa. Un formulario puede pasar a activo utilizando el método `Show`, desde el código. El evento `Activate` se puede producir sólo cuando un objeto es visible.

- `Deactivate`: Se lanza cuando un formulario deja de ser la ventana activa. No se produce cuando se descarga el formulario.

- `Click`: Se produce cuando el usuario hace clic sobre el formulario, con el ratón.

- `DblClick`: Se produce cuando el usuario hace doble clic sobre el formulario, con el ratón.

- `KeyPress`: Se genera cuando el usuario presiona una tecla. Pasa un parámetro que es el valor del código ANSI de la tecla presionada.

- `MouseMove`: Se lanza cuando el usuario mueve el ratón. Pasa como parámetros el botón del ratón que fue apretado, las teclas que se encontraban presionadas (*Alt*, *Shift* o *Control*) y las coordenadas del lugar donde está el puntero del ratón.

- `MouseDown` y `MouseUp`: Se activan al presionar y liberar, respectivamente, cualquiera de los botones del ratón. Pasa los mismos parámetros que el evento `MouseMove`.

Para ilustrar lo anterior, en el siguiente ejemplo se crea una macro que muestra un formulario. Se crea un manipulador para el evento `MouseDown`, donde se muestra un mensaje con las coordenadas del punto donde se encuentre el ratón.

Primeramente, se insertan en el proyecto activo un formulario, con la opción *UserForm* del menú *Insert* del IDE de VBA. Al formulario se le establecen las siguientes propiedades:

```
Name = frmEjemplo1
Caption = Ejemplo 1
Height = 150
StartUpPosition = 2 - CenterScreen
Width = 250
```

En la hoja de código del formulario se escribe el siguiente manipulador de evento:

```
Private Sub UserForm_MouseDown(ByVal Button As Integer, _
              ByVal Shift As Integer, ByVal X As Single, _
              ByVal Y As Single)
  MsgBox "Posición del ratón: (" & X & "; " & Y & ")",  _
         vbInformation, "Evento MouseDown"
End Sub
```

Como se puede ver, en el mismo, se muestra un mensaje con las coordenadas de la posición del ratón, dada por las variables `X` y `Y`.

Finalmente, en la hoja de código del objeto `ThisDrawing`, se añade la macro siguiente, donde se muestra el formulario creado mediante el método `Show`.

```
Public Sub FormularioVacio()
  frmEjemplo1.Show
End Sub
```

9.3 – Botones de comando.

Los botones de comando (*command buttons*) permiten ejecutar determinada acción luego de que el usuario haga clic sobre ellos. Entre sus propiedades se destacan:

- `Accelerator`: Establece la llamada tecla caliente. Al apretar simultáneamente *Alt* y dicha tecla, se lanza el evento `Click` del botón, o sea, produce el mismo efecto que hacer clic sobre el botón. El uso de teclas calientes permite trabajar con los botones prescindiendo del ratón.

- `Cancel`: Es una propiedad booleana. Si toma valor verdadero (`True`), al presionar la tecla Escape, se activa el evento `Click` del botón.

- `Caption`: Establece el texto del botón.

- `Default`: Es una propiedad booleana. Si toma valor verdadero (`True`), al presionar la tecla Retorno, se activa el evento `Click` del botón.

- `Font`: Definen la fuente que se utilizará para el texto del botón.

- `Height` y `Width`: Establecen la altura y el ancho del botón.

- `Left` y `Top`: Establece la posición desde la izquierda y la parte superior del contenedor del botón.

El evento principal que se manipula para los botones es el `Click`, aunque dispone de la mayoría de los descritos anteriormente para los formularios.

9.4 - Etiquetas.

Las etiquetas (*labels*) son controles que tienen como función básica mostrar un texto en determinada posición dentro de un formulario. Las principales propiedades de estos controles son:

- `Acelerator`: Establece la tecla caliente, tal como se explicó para los botones. Como las etiquetas no toman el foco, al presionar *Alt* y la tecla caliente, éste se traslada al control que le sigue en orden.

- `AutoSize`: Es una propiedad booleana. Si toma valor verdadero (`True`) el tamaño de la etiqueta se ajusta automáticamente al texto que contiene.

- `BorderStyle`: Establece el estilo de borde. Puede tomar dos valores: `0 - fmBorderStyleNone`, para el cual la etiqueta no tendrá borde y `1 - fmBorderStyleSimgle`, que corresponde a un borde simple.

- `Caption`: Establece el texto de la etiqueta.

- `Height` y `Width`: Establecen la altura y el ancho del botón.

- `Left` y `Top`: Establece la posición desde la izquierda y la parte superior del contenedor del botón.

- `WordWrap`: Es una propiedad booleana. Si toma valor verdadero (`True`), el texto contenido se ajustará de forma automática en múltiples líneas, en dependencia del ancho de la etiqueta.

Las etiquetas son controles informativos, por lo que la interacción de los usuarios con ellas es muy limitada. Aunque posee la mayoría de los eventos, usualmente, éstos no se manipulan.

9.5 – Cuadros de texto.

Los cuadros de texto (*textboxes*) cumplen la función de almacenar texto escrito por el usuario. Posee las propiedades anteriormente descritas para las etiquetas, excepto `Accelerator` y `Caption`. Además, tiene la propiedad:

- `Text`: Establece y contiene el texto del control. A ella se accede desde programación para obtener y manipular el texto establecido por el usuario.

- `SelStart`: Indica la posición del cursor dentro del texto.

- SelLength: Indica el número de caracteres o longitud de la selección.

- `SelText`: Devuelve la cadena de caracteres que representa el texto seleccionado.

Posee diversos eventos, entre los que se destacan:

- `Change`: Se lanza al realizar cualquier acción que provoque un cambio en el contenido de la caja de texto.

- `KeyPress`: Se activa al presionar cualquier tecla.

9.6 – Cuadros de lista y cuadros combinados.

Los cuadros de lista (*listboxes*) permiten mostrar varios registros o líneas, teniendo uno o varios de ellos seleccionados. Para añadir o eliminar registros de la lista en modo de ejecución se utilizan los métodos `AddItem` y `RemoveItem`. Las listas se suelen inicializar desde el evento `Initialize` del formulario. Con el método `Clear` se borran todos los elementos de la lista.

La propiedad `List` es un arreglo que permite acceder al contenido de la lista, utilizando el índice del elemento deseado. La propiedad `Text` devuelve el valor textual del elemento seleccionado.

Las propiedades `ListCount` y `ListIndex` juegan un papel fundamental en el trabajo con las listas. El primero devuelve la cantidad de elementos que contiene la lista; el segundo, la posición del elemento seleccionado dentro de la lista.

El principal evento que se manipula para los cuadros de listas es el `Click`, que se activa no sólo al hacer clic con el ratón sobre el control, sino también cada vez que se modifica la selección, aun cuando se realice con el teclado.

Los cuadros combinados (*ComboBoxes*) son, como su nombre lo indica, una combinación de cuadro de texto y lista. Las propiedades y métodos son muy similares a los de los cuadros de lista. Al igual que ellos también activan el evento `Click` ante cada cambio en la selección. De modo similar a los cuadros de texto, activan el evento `Change` cuando se modifica el texto.

9.7 – Imágenes.

Los controles de imagen (*images*) son controles cuya única función es mostrar una imagen dentro del formulario. Se destaca en ellos la propiedad `Picture`, donde se establece el archivo que contiene la imagen a mostrar. Aunque pueden activar diversos eventos, generalmente éstos no se manipulan.

Ejemplo resuelto.

Elaborar una macro que permita, a través de un formulario, agregar un símbolo de rugosidad superficial a un dibujo.

Solución.

La primera parte de la solución del problema consiste en crear el formulario que conformará la interfaz mediante la cual el usuario establecerá los datos del problema.

En la figura 9.3 se muestra el formulario a crear, al cual se le establecerán las siguientes propiedades:

```
Name: frmRugosidad
Caption: Rugosidad superficial
Height: 180.75
StartUpPosition: 2 - CenterScreen
Width: 249
```

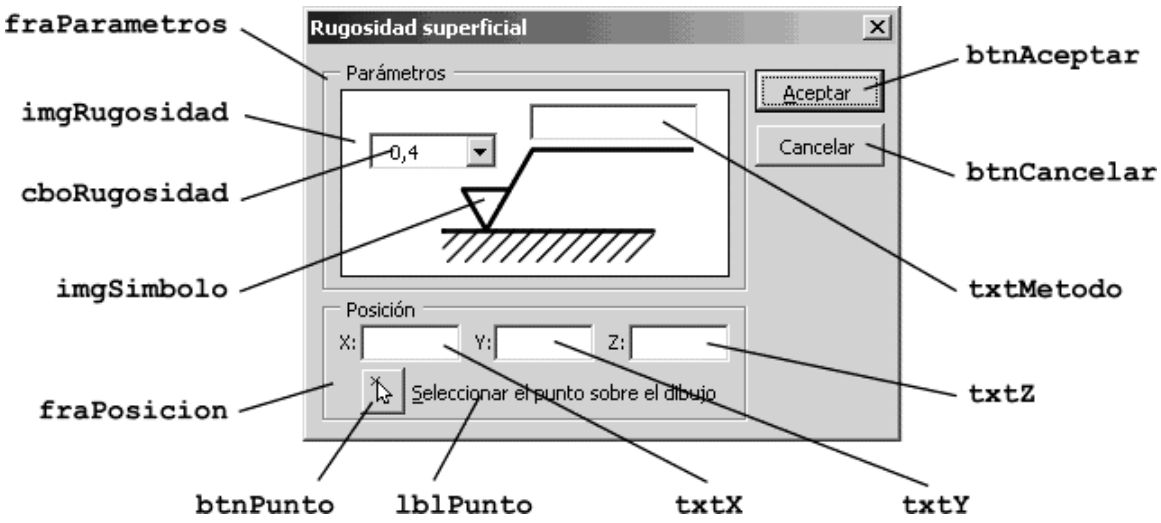


Fig. 9.3 - Formulario del ejemplo resuelto.

El formulario, tal como se muestra en la figura anterior, estará formado por los siguientes controles:

- Marco `fraRugosidad`. Contiene los elementos que permiten establecer los parámetros de la rugosidad. Propiedad `Caption = Parámetros de la rugosidad`

- Imagen `imgRugosidad`: Es una imagen vacía, de color blanco, que sirve de fondo a la imagen del símbolo y el resto de los controles de este marco. Propiedad `BackColor = &H00FFFFFF&`

- Imagen `imgSimbolo`: Contiene la imagen del símbolo de la rugosidad superficial. Propiedad `Picture = (ej09-simbolo.bmp)`

- Cuadro combinado `cboRugosidad`: Permite seleccionar el valor de la cota de rugosidad superficial. Propiedades `Style = 2 - fmStyleDropDownList` y `Text = <vacío>`.

- Cuadro de texto `txtMetodo`: Permite establecer, si tiene lugar, el método de elaboración de la superficie acotada.

- Marco fraPosicion. Contiene los controles que permiten establecer la posición del punto de inserción del símbolo de rugosidad. Propiedad `Caption = Posición.`

- Cuadros de texto `txtX`, `txtY` y `txtZ`: Permiten establecer los valores de las coordenadas (*x*, *y*, *z*) del punto de inserción del símbolo.

- Botón de comando `btnPunto`: Permite ocultar momentáneamente el formulario, para seleccionar, directamente sobre el dibujo, las coordenadas del punto de inserción. Propiedades Picture = (ej09-flecha.bmp).

- Botón de comando `btnAceptar`: Permite cerrar el cuadro de diálogo y pasar a la ejecución del dibujo del símbolo. Propiedades `Accelerator = A`, `Caption = Aceptar`, `Default = True`.

- Botón de comando `btnCancelar`: Permite cerrar el cuadro de cancelando la ejecución de la macro. Propiedades `Accelerator = C`, `Caption = Cancelar`, `Cancel = True`.

Para llenar los valores de la lista del cuadro combinado `cboRugosidad`, se implementa el manipulador del evento `Initialice` del formulario, creando el siguiente código:

```
Private Sub UserForm_Initialize()
  cboRugosidad.Clear
  cboRugosidad.AddItem "0,012"
  cboRugosidad.AddItem "0,025"
  cboRugosidad.AddItem "0,05"
  cboRugosidad.AddItem "0,1"
  cboRugosidad.AddItem "0,2"
  cboRugosidad.AddItem "0,4"
  cboRugosidad.AddItem "0,8"
  cboRugosidad.AddItem "1,6"
  cboRugosidad.AddItem "3,2"
  cboRugosidad.AddItem "6,3"
  cboRugosidad.AddItem "12,5"
  cboRugosidad.AddItem "25"
  cboRugosidad.AddItem "50"
  cboRugosidad.AddItem "100"
End Sub
```

Los valores anteriores provienen de la serie de rugosidades preferidas, medidas como Ra, establecidas por la norma ISO.

Para el evento `Click` del botón `btnPunto`, se crea el manipulador que se muestra a continuación. Nótese como se oculta el formulario con el método `Hide` para, una vez tomadas las coordenadas, volver a mostrarlo con el método `Show`.

```
Private Sub btnPunto_Click()
  Dim Pto0 As Variant
  frmRugosidad.Hide
  Pto0 = ThisDrawing.Utility.GetPoint
  txtX.Text = Pto0(0)
  txtY.Text = Pto0(1)
  txtZ.Text = Pto0(2)
  frmRugosidad.Show
End Sub
```

Para el evento `Click` del botón `btnCancel`, se programa el manipulador siguiente, cuya única función es cerrar el formulario.

```
Private Sub btnCancelar_Click()
  Hide
End Sub
```

En cambio, el evento `Click` del botón `btnAceptar` es algo más complicado, ya que tiene que verificar que las cajas de textos de las coordenadas contengan valores numéricos, lo cual se lleva a cabo mediante la función `IsNumeric`. Si para cualquiera de los textos, el resultado de esta función no es verdadero, se muestra un mensaje de error, se le pasa el foco (mediante el método `SetFocus`) al texto correspondiente y se interrumpe el procedimiento, con la instrucción `Exit Sub`.

```
Private Sub btnAceptar_Click()
  If Not IsNumeric(txtX.Text) Then
    MsgBox "El valor de la coordenada X no es válido.", _
           vbExclamation, "Error"
    txtX.SetFocus
    Exit Sub
  End If
  If Not IsNumeric(txtY.Text) Then
    MsgBox "El valor de la coordenada Y no es válido.", _
           vbExclamation, "Error"
    txtY.SetFocus
    Exit Sub
  End If
  If Not IsNumeric(txtZ.Text) Then
    MsgBox "El valor de la coordenada Z no es válido.", _
           vbExclamation, "Error"
    txtZ.SetFocus
    Exit Sub
  End If
  Tag = 1
  Hide
End Sub
```

Finalmente, si los textos son numéricos, se le asigna a la propiedad `Tag` (que sólo es un espacio para almacenar un valor que se utilizará según convenga) del formulario el valor `1`, para notificar que el formulario fue aceptado y, luego, se cierra.

Por último, se implementa la macro, tal como se muestra en el código siguiente:

```
Public Sub Rugosidad()
  ' Definición de las variables
  Dim Pto0(0 To 2) As Double, PtoA(0 To 2) As Double, _
      PtoB(0 To 2) As Double, PtoC(0 To 2) As Double, _
      PtoD(0 To 2) As Double, PtoE(0 To 2) As Double, _
      PtoF As Variant, PtoG As Variant, _
      PtoH(0 To 2) As Double, PtoI(0 To 2) As Double
  Dim TextoCota As AcadText, TextoLong As AcadText
  ' Vaciado de los controles
  frmRugosidad.cboRugosidad.Text = _
                        frmRugosidad.cboRugosidad.List(5)
  frmRugosidad.txtMetodo.Text = ""
  frmRugosidad.txtX.Text = ""
  frmRugosidad.txtY.Text = ""
  frmRugosidad.txtZ.Text = ""
  ' Lanzamiento del formulario
  frmRugosidad.Tag = 0
  frmRugosidad.Show
  ' Verificación de que se aceptó el formulario
  If frmRugosidad.Tag = 1 Then
    ' Símbolo de rugosidad
    Pto0(0) = Val(frmRugosidad.txtX.Text)
    Pto0(1) = Val(frmRugosidad.txtY.Text)
    Pto0(2) = Val(frmRugosidad.txtZ.Text)
    PtoA(0) = Pto0(0) + 8 * 0.5774
    PtoA(1) = Pto0(1) + 8
    PtoA(2) = Pto0(2)
    PtoB(0) = Pto0(0) - 4 * 0.5774
    PtoB(1) = Pto0(1) + 4
    PtoB(2) = Pto0(2)
    PtoC(0) = Pto0(0) + 4 * 0.5774
    PtoC(1) = Pto0(1) + 4
    PtoC(2) = Pto0(2)
    ThisDrawing.ModelSpace.AddLine Pto0, PtoA
    ThisDrawing.ModelSpace.AddLine Pto0, PtoB
    ThisDrawing.ModelSpace.AddLine PtoB, PtoC
    ' Cota de rugosidad
    PtoD(0) = Pto0(0)
    PtoD(1) = Pto0(1) + 5
    PtoD(2) = Pto0(2)
    Set TextoCota = ThisDrawing.ModelSpace.AddText( _
                  frmRugosidad.cboRugosidad.Text, PtoD, 4)
    TextoCota.Alignment = acAlignmentRight
```

```
    TextoCota.TextAlignmentPoint = PtoD
    If (Trim(frmRugosidad.txtMetodo.Text) <> "") Then
      ' Método de elaboración
      PtoE(0) = PtoA(0) + 1
      PtoE(1) = PtoA(1) + 1
      PtoE(2) = PtoA(2)
      Set TextoLong = ThisDrawing.ModelSpace.AddText( _
                          frmRugosidad.txtMetodo.Text, _
                          PtoE, 4)
      TextoLong.GetBoundingBox PtoF, PtoG
      PtoH(0) = PtoG(0) + 1
      PtoH(1) = PtoA(1)
      PtoH(2) = PtoA(2)
      ThisDrawing.ModelSpace.AddLine PtoA, PtoH
    End If
  End If
End Sub
```

En la misma se definen las variables de utilizar. Éstas son los puntos *A* ... *H*, los cuales son arreglos de tres dimensiones de tipo `Double`, excepto *F* y *G*, que son de tipo `Variant` ya que se obtienen como resultado del método `GetBoundingBox`. También se definen las variables `TextoCota` y `TextoLong` de tipo `AcadText`, para contener los textos de la cota de rugosidad y del método de elaboración.

Antes de mostrar el formulario, los valores de sus controles se vacían. Este paso no es en absoluto obligatorio, pero ayuda a darle una buena apariencia a la aplicación. Luego, el valor de la propiedad `Tag` se establece en 0, y se muestra el formulario con `Show`.

Luego de que establecer los valores en los controles del formulario, y de cerrarlo, se verifica que se haya cerrado mediante el botón `btnAceptar`, comprobando que el valor de la propiedad `Tag` sea igual a 1. En caso contrario, se pone fin a la ejecución de la macro.

Los valores de los puntos se calculan según el esquema siguiente:

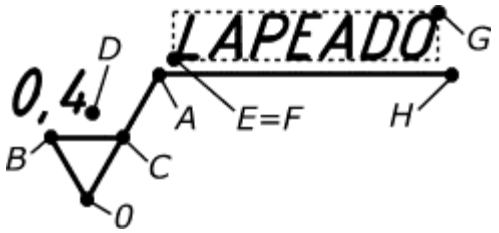


Fig. 9.4 - Puntos utilizados en la generación del símbolo.

Con los puntos calculados, se crean las líneas y textos que forman el símbolo. Nótese que el texto del método de elaboración, con la línea horizontal, sólo se dibujan si se introdujo algún texto en la caja de texto correspondiente, ya que es un parámetro opcional

Preguntas y ejercicios propuestos.

1. ¿Qué es un formulario y para qué se emplea?

2. ¿Qué es un control? ¿Qué tipos de controles existen en VBA?

3. Mencione alguno de los eventos estudiados y diga para que controles se manipulan usualmente.

4. Elabore una macro, que utilice un formulario, para dibujar una cota con tolerancia según el sistema ISO de ajustes.

APÉNDICE
Vinculación con otras aplicaciones

Una de las características que hace poderosa a la automatización *ActiveX*, es la posibilidad de vincular dos aplicaciones entre sí, desde el entorno de programación. Aunque, esta materia se va fuera del alcance de este material, se ha incluido el ejemplo siguiente como un apéndice, con el fin de que sirva de introducción y motivación a los lectores, para adentrarse en esta temática.

En el ejemplo, se toma un archivo de Microsoft Excel desde una macro de VBA para AutoCAD, se leen los valores de un rango de datos de una de sus hojas de cálculo y se crea, a partir de ellos, una malla tridimensional.

Antes de emplear cualquier elemento del modelo de objetos de Excel, es necesario añadirlo como referencia. Esto se puede lograr a través del cuadro de diálogo "Referencias del Proyecto" ("*References ACADProject*"), al cual se accede a través de la opción *References*... del menú *Tools* del menú estándar del IDE de VBA. Debe chequearse la casilla correspondiente a *Microsoft Excel 9.0 Object Library* (la versión puede variar). Naturalmente, esto sólo es posible si Excel se encuentra instalado en la computadora.

```
Public Sub Superficie()
  ' Definición de las variables
  Dim Archivo As String, NombreArchivo As String, _
      Hoja As String, RangoTexto As String
  Dim I As Integer, J As Integer, C As Integer
  Dim LibroExcel As Workbook
  Dim Rango As Range
  Dim Puntos() As Double
  Dim Filas As Integer, Columnas As Integer
  Dim Pto0 As Variant
  Dim XPaso As Double, YPaso As Double
  ' Toma de datos
  Archivo = ThisDrawing.Utility.GetString( _
      1, "Entre la ruta y el nombre del archivo: ")
  Hoja = ThisDrawing.Utility.GetString( _
      1, "Entre el nombre de la hoja: ")
  RangoTexto = ThisDrawing.Utility.GetString( _
      1, "Entre el rango de las celdas (Ejemplo A1:H6): ")
  Pto0 = ThisDrawing.Utility.GetPoint(, _
      "Entre el punto de origen: ")
  XPaso = ThisDrawing.Utility.GetReal( _
      "Entre el paso en dirección de X: ")
  YPaso = ThisDrawing.Utility.GetReal( _
      "Entre el paso en dirección de Y: ")
```

```
  ' Extracción del nombre del archivo
  NombreArchivo = ""
  For I = Len(Archivo) To 1 Step -1
    If Mid(Archivo, I, 1) <> "\" Then
      NombreArchivo = Mid(Archivo, I, 1) & NombreArchivo
    Else
      Exit For
    End If
  Next I
  ' Apertura del archivo
  Workbooks.Open FileName:=Archivo, ReadOnly:=True
  ' Toma de valores
  Set Rango = Workbooks(NombreArchivo).Worksheets(Hoja). _
              Range(RangoTexto)
  ReDim Puntos(Rango.Count * 3 - 1)
  Filas = Rango.Range(RangoTexto).Columns.Count
  Columnas = Rango.Range(RangoTexto).Rows.Count
  C = 0
  For I = 1 To Rango.Columns.Count
    For J = 1 To Rango.Rows.Count
      Puntos(C) = Pto0(0) + (I - 1) * XPaso
      Puntos(C + 1) = Pto0(1) + (J - 1) * YPaso
      Puntos(C + 2) = Rango.Cells(I, J).Value
      C = C + 3
    Next J
  Next I
  ' Creación de la malla
  ThisDrawing.ModelSpace.Add3DMesh _
    Rango.Columns.Count, Rango.Rows.Count, Puntos
  ' Cierre del archivo
  Workbooks(NombreArchivo).Close False
End Sub
```

En el listado anterior se muestra el código de la macro. Si se utiliza el archivo *app1.xls*, que se suministra junto a los ejemplos de este material, tomando el rango de celdas *A1:U21*, de la hoja *Valores*, se obtiene un gráfico como el siguiente.

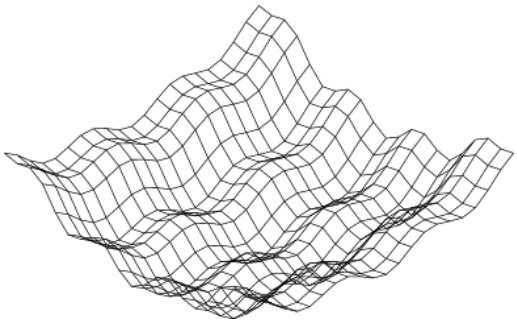


Fig. A.1 – Superficie generada por la macro desde un archivo de Excel