

## Capítulo 10

# Meshes

En los capítulos anteriores se ha utilizado el vertex buffer junto con otros elementos como las texturas o el index buffer para pintar gráficos en pantalla. Sin embargo, estas estructuras no resultan prácticas cuando se dibujan figuras complejas, pues aún si se utiliza el index buffer, siempre resultará tedioso tener que definir varios cientos o talvez varios miles de vértices.

Para solucionar esto, DirectX incorpora una estructura llamada Mesh (malla). Esta estructura encapsula los datos del vertex buffer y del index buffer en un solo objeto, y además ofrece la opción de leer los datos desde un archivo. De esta forma, se puede usar un software de diseño gráfico para crear los modelos en 3D y posteriormente cargarlos con DirectX en un objeto de tipo mesh.

En este capítulo se muestra como utilizar el objeto Mesh para pintar figuras básicas en pantalla. En capítulos posteriores se explicará como crear un objeto Mesh a partir de los datos almacenados en un archivo.

### **Utilizar un Mesh para pintar figuras en pantalla**

Para pintar gráficos en pantalla utilizando un mesh, lo primero que se necesita es una variable para almacenar el objeto Mesh. Agregue la siguiente variable miembro a la clase:

```
Private miMesh As Mesh = Nothing
```

Teniendo esto, ya se puede crear el mesh. Como el mesh no se va a modificar, se puede crear al iniciar el programa. Agregue la siguiente instrucción al final del evento **Load**:

```
' Se crea el objeto Mesh (Un cubo de 2 x 2 x 2)
miMesh = Mesh.Box(device, 2, 2, 2)
```

Como se puede suponer, esta instrucción crea un objeto mesh que contendrá un cubo. La clase “Mesh” tiene métodos para crear figuras básicas en 3 dimensiones. Las figuras básicas que se pueden crear con la clase Mesh son: Cubo, Cilindro, Esfera, Toroide, y Tetera (si, tetera). Por ejemplo, con la instrucción anterior se ha creado un cubo de dimensiones 2 x 2 x 2.

También la forma de pintar el cubo en pantalla resulta más sencilla comparada con la forma de dibujar utilizando el vertex buffer. Para pintar el cubo, agregue la siguiente instrucción en el evento **Paint**, entre las instrucciones **BeginScene** y **EndScene**:

```
' Se dibuja la figura en pantalla
miMesh.DrawSubset(0)
```

Un objeto Mesh puede estar dividido en varias secciones, esto es, puede estar formado por varios subconjuntos de triángulos. La finalidad de dividirlo es que para cada subconjunto de triángulos se pueda definir una textura diferente o un material diferente (los materiales se explicarán en capítulos posteriores).

De esta forma, el objeto tridimensional cuyos datos están contenidos dentro del objeto mesh no está limitado a tener una sola textura o un solo material, sino que puede utilizarse uno diferente para cada sección del objeto.

La instrucción DrawSubset dibuja un subconjunto de triángulos del mesh, y el parámetro que recibe es precisamente el número de subconjunto que se debe dibujar en esa llamada. Como en este caso el mesh solamente tiene un subconjunto de triángulos, es el único que se debe dibujar (el número cero).

Si se ejecuta el programa ahora, se verá de nuevo un cuadrado negro en pantalla, pues de nuevo se está mirando una sola cara del cubo. Para mirar más caras del

cubo, modifique la vista de cámara de forma que se pueda ver la figura desde otro punto de vista:

```
'Vista de Cámara
device.Transform.View = Matrix.LookAtLH(New Vector3(-5, 5, -5),
                                         New Vector3(0, 0, 0), New Vector3(0, 1, 0))
```

Y si se ejecuta el programa ahora, ya se puede ver el cubo realmente como una figura tridimensional, pero como no se ha definido información de color, todo el cubo se dibuja de color negro haciendo que se pierdan las aristas y dando como resultado que aparezca solamente una silueta.

Una forma de corregir esto, es acceder los datos del vertex buffer que se encuentra dentro del mesh, modificar el tipo de estructura que deberá utilizar para agregar información de color, y a partir de estos nuevos datos crear otro mesh. Esto simplemente puede resultar aún más tedioso que haber utilizado un vertex buffer desde un principio. Además se estarían desperdiciando dos elementos muy poderosos de DirectX que hacen interesante la programación de gráficos: las luces y los materiales.

Estos elementos se explicarán a detalle en capítulos posteriores. Por ahora basta con comentar que las luces se utilizan para iluminar una escena y los materiales definen el comportamiento de color de los objetos.

Para definir una luz y un material agregue las siguientes instrucciones al final del evento **Load**:

```
' Se habilita la iluminacion
device.RenderState.Lighting = True

' Se define el material
Dim miMaterial As Material
miMaterial.Diffuse = Color.Salmon
device.Material = miMaterial

' Se define la luz
device.Lights(0).Type = LightType.Directional
device.Lights(0).Direction = New Vector3(0, -0.5, 1)
device.Lights(0).Diffuse = Color.White
```

```
device.Lights(0).Commit()  
device.Lights(0).Enabled = True
```

La primera instrucción le indica a DirectX que se va a utilizar iluminación. Este es el comportamiento predeterminado de DirectX, solo que aquí se ha especificado explícitamente.

Las siguientes líneas de código definen un material de color salmón, que se le aplicará al cubo al momento de dibujarlo en pantalla. Las últimas líneas definen una luz blanca de tipo direccional.

Ahora el programa ya está completo. Si se ejecuta, se verá un cubo como el de la figura 1.

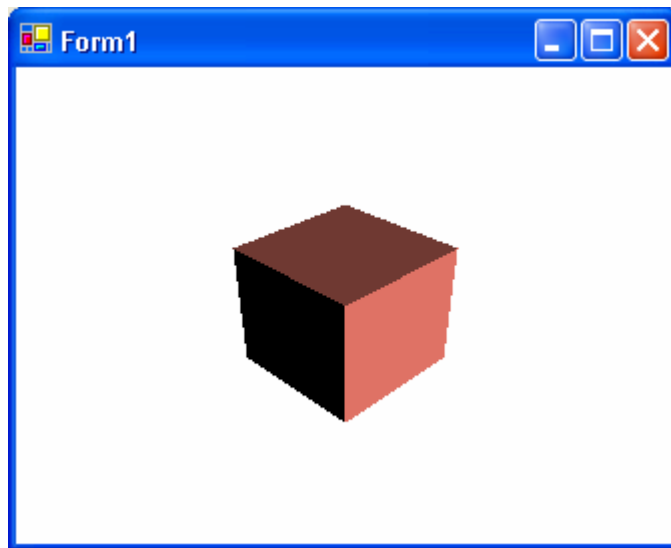


Figura 1

Como se comentó anteriormente, la clase Mesh tiene métodos para crear varias figuras, no solo un cubo. Por ejemplo, para crear un cilindro reemplace la instrucción donde se crea el objeto Mesh por esta otra:

```
' (Un Cilindro de radio 1 y 3 unidades de largo)  
miMesh = Mesh.Cylinder(device, 1, 1, 3, 20, 1)
```

Los parámetros que recibe este método son (a parte del dispositivo) el radio superior e inferior, el largo, y la cantidad de divisiones que tendrá a lo largo y a lo ancho (figura 2).

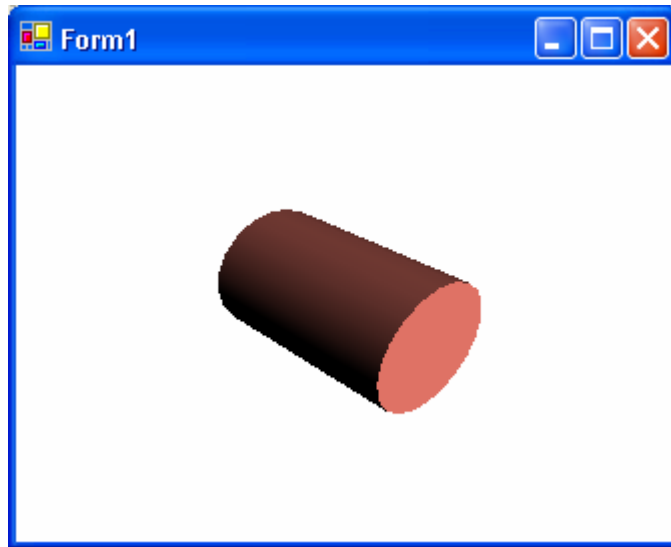


Figura 2

Para crear una esfera, la instrucción es la siguiente:

```
' (Una esfera de radio 2)
miMesh = Mesh.Sphere(device, 2, 50, 50)
```

Los parámetros son el radio y las divisiones que tendrá a lo largo y a lo ancho (figura 3).

La siguiente instrucción creará un Toroide de radio interior 1, radio exterior 2, con 20 divisiones a lo ancho y con 15 lados formando el círculo (figura 4):

```
' (Un toroide de radios 1 y 2)
miMesh = Mesh.Torus(device, 1, 2, 20, 15)
```

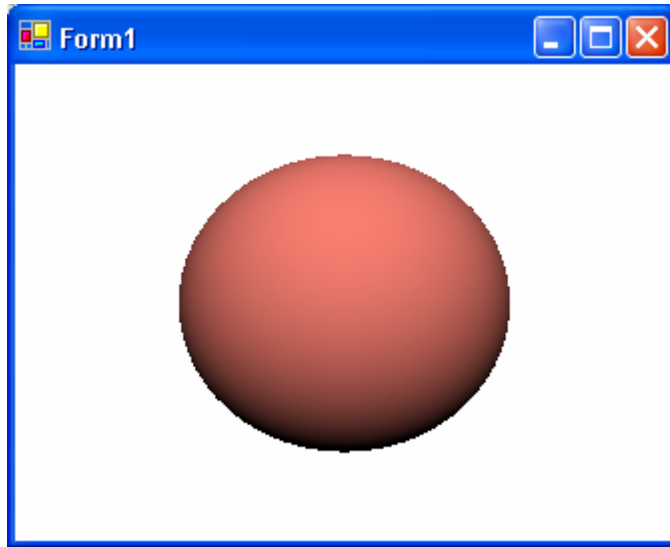


Figura 3

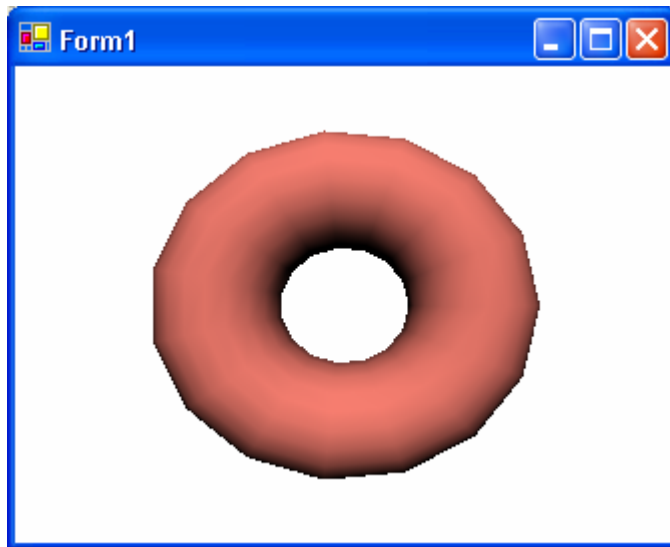


Figura 4

Finalmente, la siguiente instrucción creará una tetera (figura 5):

```
' (Una tetera)
miMesh = Mesh.Teapot(device)
```

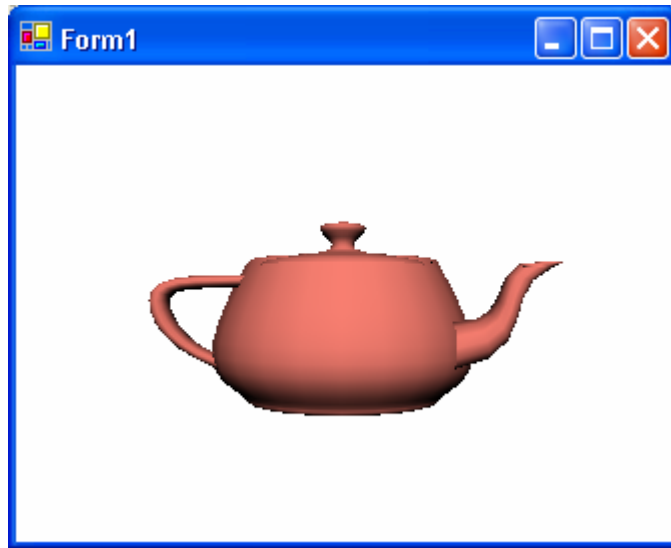


Figura 5

---

**Nota:** En cada ejemplo se ha modificado la vista de cámara para mirar con mayor claridad a cada objeto.

---

**Nota:** Para comprender mejor los diferentes parámetros de estas rutinas, se puede utilizar el modo de relleno “WireFrame”, tal como se utilizó en el capítulo 7.

---

### **En resumen...**

En este capítulo se ha explicado como utilizar la clase Mesh para pintar figuras básicas en pantalla:

- Como crear una figura básica utilizando la clase Mesh
- Como dibujar en pantalla el objeto contenido dentro de un Mesh