8 Cómo agregar un efecto en XNA

En este capítulo se muestra cómo cargar un efecto proveniente de un archivo **fx**, en específico serán algunos de los ejemplos vistos anteriormente sobre el lenguaje HLSL, y se espera que el lector pueda cargar cualquier otro shader.

El **ContentManager** de XNA realiza la mayor parte del trabajo, sin embargo, la manera en cómo se le pasarán los datos de XNA a HLSL dependerá del programador, pues si en el shader no se restringieron sus variables **extern uniform**, en C# se pueden limitar; además se debe tener cuidado en la asignación de datos.

La pregunta es ¿cómo reconocer los tipos de datos correspondientes del **.Net Framework Class Library**(CL) y del **Framework de XNA** a HLSL? En sí, la mayoría de los datos comunes o no compuestos como los enteros, flotantes y boleanos tienen nombres similares entre CL y HLSL. La manera de reconocer los tipos de datos compuestos o estructuras como **float4x4**, **float3** es haciendo una comparación "manual" entre sus campos y los que constituyen aquellos que pueden parecerse en el Framework de XNA, por lo tanto hay que conocer los tipos de datos de ambos, lo cual se deja al lector como estudio, y que en ocasiones será intuitivo esta asignación de tipos de datos.

8.1 Efecto ambiental

En este aparatado se muestran dos formas de cargar un efecto proveniente de un archivo **fx**. El primero muestra cómo cambiar las instancias de la clase **Effect** del objeto **Model** por la instancia **Effect** asociada al efecto ambiental; esto permite mandar a llamar el método **Draw** de la clase **ModelMesh**, permitiendo una programación más sencilla, pero a costa de cambiar todas las propiedades de material de la instancia **Model**.

La segunda forma no cambia las propiedades del material asociadas a la instancia **Model**. Para poder dibujar la geometría con el efecto proveniente de un archivo **fx**, se utiliza el método **DrawIndexedPrimitives** de la clase **GraphicsDevice**.

8.1.1 Clonación de efecto

En este primer ejemplo se utiliza el método **Effect.Clone** para copiar los valores de un objeto existente a otro, por lo que este método arroja una excepción cuando se tratan de copiar los valores de un objeto nulo, así que no se recomienda el uso del signo de asignación, =.

Aclarado el uso del método **Clone** y el signo de asignación, creé un nuevo proyecto de XNA en Visual Studio. En la clase **Game1** escriba las siguientes variables de instancia:

Código 8-1

```
1.      private Model modelo;
2.      private Matrix[] transformaciones;
3.      private Texture2D textura;
4.      Single tiempo;
```

La declaración de la variable modelo es para el modelo tridimensional, línea 1, Código 8-1; que en este caso fue **ElefanteC.fbx** y que puede ser cualquiera que tengan a la mano. En seguida se declara un arreglo de matrices que almacenaran las transformaciones del modelo, línea 2.

El efecto **Ambiental.fx** tiene dos tipos de técnicas, **Material** y **Texturizado**, para este último se ha declarado a textura como tipo **Texture2D**, línea 3. La variable tiempo servirá como ángulo de rotación alrededor del eje **y**, línea 4.

Antes de continuar con el código, creé tres carpetas en el **Content** en el **Explorador de soluciones** de Visual Studio, los nombres que se utilizaron para las carpetas fueron: **Modelos**, **Shaders** y **Texturas**.

En cada una de ellas se añadieron el modelo **ElefanteC.fbx**, el shader **Ambiental.fx** y la imagen **Omision.png**, respectivamente.

Una vez que el **ContentManager** les ha asignado el **Asset** a cada elemento, en el método **LoadContent** de la clase **Game1** escriba las siguientes líneas.

Código 8-2

```
1.      protected override void LoadContent()
2.      {
3.          spriteBatch = new SpriteBatch(GraphicsDevice);
4.
5.          modelo = Content.Load<Model>(@"Modelos\ElefanteC");
6.          Effect efecto = Content.Load<Effect>(@"Shaders\Ambiental");
7.          textura = Content.Load<Texture2D>(@"Texturas\Omision");
8.          transformaciones = new Matrix[modelo.Bones.Count];
9.          modelo.CopyAbsoluteBoneTransformsTo(transformaciones);
10.
11.         foreach (ModelMesh mesh in modelo.Meshes)
12.            foreach (ModelMeshPart meshPart in mesh.MeshParts)
13.                 meshPart.Effect = efecto.Clone(GraphicsDevice);
14.     }
```

De la línea 5 a la 7, Código 8-2, el método Load genérico de la clase **ContentManager** crea las instancias respectivas a su tipo de dato de entrada, y se los asigna a modelo, efecto y textura. Luego se copian las matrices de transfomación del modelo al arreglo transformaciones.

En las líneas 11 a 13, se copian los datos de efecto a la propiedad **Effect** del **ModelMeshpart** de cada **ModelMesh** del modelo.

Para rotar el modelo alrededor del eje **y** se ha declarado la variable **tiempo**, que servirá como ángulo. En el método **Update** de la clase **Game1,** se actualizará tiempo cada segundo con el total de tiempo que ha transcurrido la apliación desde que inició. En la línea 6, Código 8-3, se sebe convertir el tipo **double** a **float**.

Código 8-3

```
1.      protected override void Update(GameTime gameTime)
2.      {
3.          if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
4.              this.Exit();
5.
6.          tiempo = (Single)gameTime.TotalGameTime.TotalSeconds;
7.
8.          base.Update(gameTime);
9.      }
```

Por último, en el método **Draw** se dibuja la geometría como se vio en el Cómo cargar modelos 3D desde un archivo X y FBX, con la diferencia en que el **foreach** anidado, líneas 7 a 22, Código 8-4, tiene como elemento de interación una instancia **Effect**.

En la línea 9 del método **Draw**, se establece la técnica del shader con la propiedad **CurrentTechnique** de la clase **Effect** y se obtiene a partir de la colección **Techniques** del objeto **effect**. Se puede utilizar como índice de selección un entero, sin embargo, el **string** con el nombre de la técnica es más útil para el mantenimiento del programa.

Después de seleccionar la técnica del shader, se establecen los valores de las variables **uniform extern** del shader, con la ayuda del método **SetValue**. La instancia de la clase **Effect** tiene una colección de parámetros que representan estas variables del shader. La forma de selección es por medio de un tipo entero o un **string**. Este último se utiliza como preferente, por legibilidad del programa.

El método **SetValue** está sobrecargado 18 veces[38] para aceptar diferentes tipos de datos, sin embargo, no verifica en tiempo de compilación la integridad de éstos.

[38] Para consultar todas las sobrecargas del método consulte la siguiente página: http://msdn.microsoft.com/query/dev10.query?appId=Dev10IDEF1&l=EN-US&k=k(MICROSOFT.XNA.FRAMEWORK.GRAPHICS.EFFECTPARAMETER.SETVALUE);k(DevLang-CSHARP)&rd=true

La línea 10 del método **Draw**, establece la matriz de mundo a partir del elemento del arreglo transformaciones y la matriz de rotación alrededor del eje **y**. En la línea 13 se establece la matriz de vista a partir del método **CreateLookAt**. Siguiendo con la asignación del color del material ambiental, línea 19 – 20, se debe convertir el tipo de dato **Color** por un **Vector4**, que contiene cuatro elementos de tipo flotante. Por último, se asigna la textura al shader en la línea 21.

Código 8-4

```
1.      protected override void Draw(GameTime gameTime)
2.      {
3.          GraphicsDevice.Clear(Color.White);
4.
5.          foreach (ModelMesh mesh in modelo.Meshes)
6.          {
7.              foreach (Effect effect in mesh.Effects)
8.              {
9.                  effect.CurrentTechnique = effect.Techniques["Material"];
10.                 effect.Parameters["world"].SetValue(
11.                     transformaciones[mesh.ParentBone.Index] *
12.                     Matrix.CreateRotationY(tiempo));
13.                 effect.Parameters["view"].SetValue(Matrix.CreateLookAt(
14.                     new Vector3(0, 0, 300), Vector3.Zero, Vector3.Up));
15.                 effect.Parameters["projection"].SetValue(
16.                     Matrix.CreatePerspectiveFieldOfView(
17.                     MathHelper.PiOver4, GraphicsDevice.Viewport.AspectRatio,
18.                     1.0F, 1000.0F));
19.                 effect.Parameters["colorMaterialAmbiental"].SetValue(
20.                     Color.Gray.ToVector4());
21.                 effect.Parameters["texturaModelo"].SetValue(textura);
22.             }// fin del foreach
23.             mesh.Draw();
24.         }// fin del foreach
25.
26.         base.Draw(gameTime);
27.     }
```

Hasta ahora no ha sido tan complicada la asingación de valores al shader, sin embargo, esta manera de utilizar el efecto no es la más conveniente, porque si quisiéramos pintar este mismo objeto **Model** con otros efectos de iluminación se tendría que clonar una vez más la instancia **Effect** del **ModelMesh**, ocupando cuatro bucles **foreach**. En los siguientes ejemplos de este capítulo se utiliza una manera más eficaz pero no tan sencilla o por lo menos no tan corta.

Corra el programa para ver un elefante, o el modelo tridimensional de su preferencia, pintado de un sólo color, véase Ilustración 8-1. Cambie la técnica **Material** por **Texturizacion**, en la línea 9 del método **Draw**.

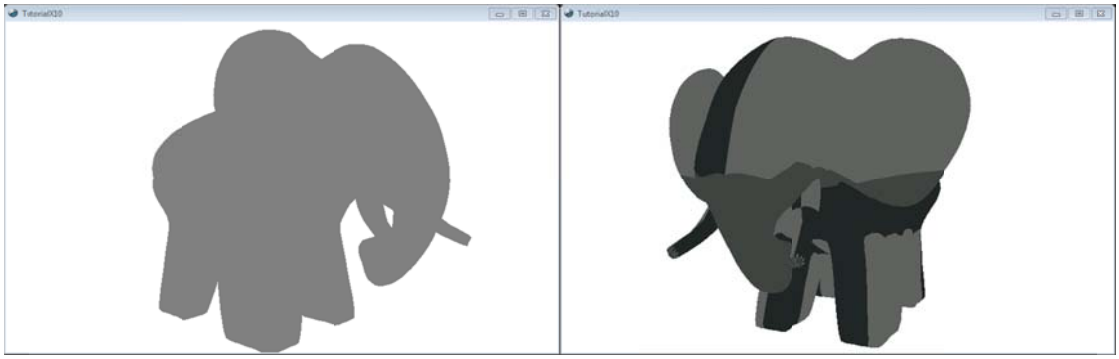

Ilustración 8-1 Integración de shader en XNA

8.1.2 Interfaz para el shader

El subtítulo de este apartado no debe confundirse con la palabra reservada interface de C#, es algo más general. Es la manera de comunicación entre XNA y el shader. Y eso es lo que se programara para dibujar la geometría con el efecto deseado, sin necesidad de cambiar sus propiedades de material.

A partir de este ejemplo y lo queda de este capítulo se reutilizarán los enlistados, con el fin de aprovechar las ventajas de la programación orientada a objetos.

Comience por crear un nuevo proyecto en Visual Studio de XNA Game. Vuelva a elaborar las tres carpetas **Modelos**, **Shaders** y **Texturas** en el **Content**, en la parte del explorador de soluciones. En la carpeta **Modelos**, añada cualquier modelo tridimensional que pueda manejar el **ContentManager**. En la carpeta **Shader** vuelva a añadir el efecto **Ambiental.fx** que se vio en el capítulo previo a éste. Asegúrese que la textura que agregue en la carpeta **Texturas**, sea compatible con los formatos que el **ContentManager** controla.

Con un diagrama de clases se puede visualizar mejor el software que se contruirá, para las interfaces de los efectos. En la ilustración 8 – 2 se muestran las interfaces **IFx**, **IDifuso** e **IEspecular**. Cada una de ellas servirá para el polimorfismo y herencia de las clases que interactúen con el shader.

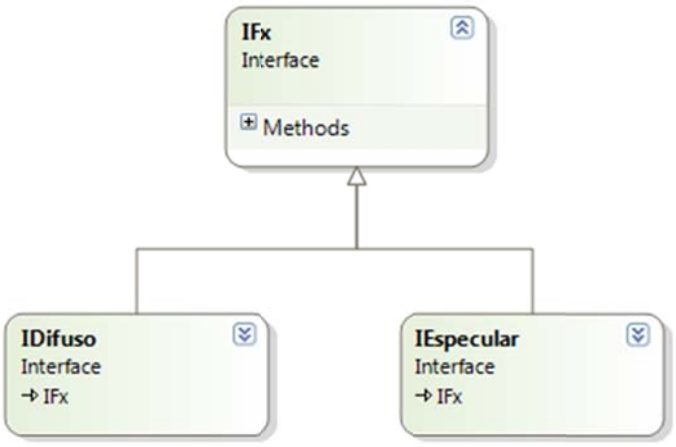

Ilustración 8-2 Diagrama de clases. Interfaces

Para crear una interfaz en Visual Studio seleccione el nombre de la solución en el explorador de soluciones. Haga clic derecho y seleccione **Añadir nuevo elemento**. Inmediatamente se abre una ventana de diálogo para seleccionar una plantilla de cualquiera de las categorías. Seleccione la plantilla Interface de la categoría **Código**. El nombre que le asigne a cualquier interfaz debe comenzar con la letra i mayúscula, seguida del nombre que la describe.

Una vez creado el archivo de la interfaz **IFx** escriba el método **DrawModelMeshPart**, dentro de las llaves de la interfaz **IFx**.

```
public interface IFx
{
    /// <summary>
    /// Método que dibuja un ModelMeshPart.
    /// </summary>
    void DrawModelMeshPart(Microsoft.Xna.Framework.Graphics.ModelMeshPart modelMeshPart,
        Microsoft.Xna.Framework.Graphics.GraphicsDevice graphicsDevice);
}
```

Este método dibujará el **ModelMeshPart** que compone al **ModelMesh** de la clase **Model**. Cabe aclarar que cada **ModelMeshPart** se distingue de otro por sus diferentes materiales.

La interfaz **IDifuso**, hereda de la interfaz IFx, y añade una propiedad que se llama **ColorDifuso**. Creé una nueva interfaz con el mismo nombre **Idifuso** y escriba las siguientes líneas.

```
public interface IDifuso : IFx
{
    /// <summary>
    /// Propiedad que obtiene o establece el color difuso.
    /// </summary>
    Color ColorDifuso { get; set; }
}
```

La creación de esta interfaz es para separar los modelos de iluminación difusa del especular. Por lo que la siguiente interfaz es para el especular.

Agregue una interfaz al proyecto con el nombre **IEspecular** y escriba en ella el siguiente enlistado.

```
{
    /// <summary>
    /// Propiedad que obtiene o establece el coeficiente especular.
    /// </summary>
    Single Ks { get; set; }

    /// <summary>
    /// Propiedad que obtiene o establece la potencia especular.
    /// </summary>
    Single N { get; set; }

    /// <summary>
    /// Propiedad que obtiene o establece el color especular.
    /// </summary>
    Color ColorEspecular { get; set; }

    /// <summary>
    /// Propiedad que habilita o inhabilita el modelo especular.
    /// </summary>
    Boolean HabilitarEspecular { get; set; }

    /// <sumary>
    /// Propiedad que obtiene o establece la posición de la cámara.
    /// </sumary>
    Vector3 PosicionCamara { get; set;}

}
```

Esta interfaz **IEspecular** tiene las propiedades del coeficiente especular, la potencia especular, el color especular, la posición de cámara y la opción de habilitar o inhabilitar el modelo de iluminación especular.

8.1.2.1 Clase FxAmbiental.

La clase **FxAmbiental**, que a continuación se describe, hereda de la interfaz **IFx**, y define el método **DrawModelMeshPart**. Además se definen los campos y propiedades que ayudan a la comunicación entre XNA y el efecto **Ambiental.fx**.

Creé una nueva clase en la solución, con el nombre **FxAmbiental**, y en la parte de directivas escriba las siguientes.

```
using System;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework;
```

Dentro del cuerpo de la clase escriba las variables instancia, líneas 3 – 9, Código 8-5. La única variable cuyo modificador de acceso es diferente a private es **effect**, línea 9. Esto es porque de esta clase se heredan

las siguientes clases para el shader **Direccional.fx** y **MulDirec.fx**, este último es una versión reducida del shader **Multilliminación** visto en el capítulo anterior. Véase que cada una de estas variables, excepto **habilitarTextura** y **effect**, corresponden a las variables **uniform** del shader **Ambiental.fx**.

En el constructor, líneas 11 – 16, se obtiene una instancia **Effect** como parámetro y el cuerpo inicializa el color ambiental e inhabilita la técnica de **Texturizacion**.

Cada una de las propiedades obtiene o establece el valor correspondiente a la variable de instancia de la clase, y para ser legible las propiedades tienen el mismo nombre que la variables, pero diferenciándolas por tener la primera letra en mayúsculas.

El descriptor de acceso **set** de cada propiedad establece el valor a la variable de instancia de la clase, antes de pasar el dato al método **SetValue**.

En casi todas las propiedades la asignación de los datos hacia el efecto es igual al ejemplo anterior, exceptuando en **HabilitarTextura**, línea35 – 44. El valor de ésta es de tipo boleano, sirve como selector entre las técnicas. Para ello se utiliza el operador ternario **? :**. Si el valor de la variable **habilitarTextura** es verdadero, se selecciona el **string "Texturizacion"**, en caso contrario será **"Material"**.

El método **DrawModelMeshPart** utiliza el modificador virtual para que otras clases derivadas de **FxAmbiental** puedan cambiar el cuerpo, pero no así la firma. La manera en que se manda a dibujar la geometría es las misma que se vio en el Vertex Buffer, en donde se envuelve el método **DrawIndexedPrimitives** entre los métodos **Begin** y **End** del objeto **effect**, líneas 103 – 110. En este caso el shader Ambiental contiene una pasada y se toma el primer elemento de la colección **Passes**. Todos los parámetros que necesita este método se obtienen del **ModelMeshPart**.

Código 8-5

```
1.      public class FxAmbiental : IFx
2.      {
3.          private Matrix view;
4.          private Matrix projection;
5.          private Matrix world;
6.          private Color colorAmbiental;
7.          private Texture2D textura2D;
8.          private Boolean habilitarTextura;
9.          protected Effect effect;
10.
11.         public FxAmbiental(Effect effect)
12.         {
13.             this.effect = effect;
14.             ColorAmbiental = Color.Black;
15.             HabilitarTextura = false;
16.         }
17.
18.         /// <summary>
19.         /// Propiedad que obtiene o establece el color ambiental del material.
20.         /// </summary>
21.         public virtual Color ColorAmbiental
22.         {
23.             get { return colorAmbiental; }
24.             set
25.             {
26.                 colorAmbiental = value;
27.                 effect.Parameters["colorMaterialAmbiental"].SetValue(
28.                 colorAmbiental.ToVector4());
29.             }
30.         }// fin de la propiedad ColorAmbiental
31.
32.         /// <summary>
33.         /// Propiedad que habilita o inhabilita la textura.
34.         /// </summary>
35.         public virtual Boolean HabilitarTextura
36.         {
37.             get { return habilitarTextura; }
38.             set
```

```
39.             {
40.                 habilitarTextura = value;
41.                 effect.CurrentTechnique = effect.Techniques[
42.                 (value) ? "Texturizado" : "Material"];
43.             }
44.         }// fin de la propiedad HabilitarTextura
45.
46.         /// <summary>
47.         /// Propiedad que obtiene o establece la matriz projection.
48.         /// </summary>
49.         public virtual Matrix Projection
50.         {
51.             get { return projection; }
52.             set
53.             {
54.                 projection = value;
55.                 effect.Parameters["projection"].SetValue(projection);
56.             }
57.         }// fin de la propiedad Projection
58.
59.         /// <summary>
60.         /// Propiedad que obtiene o establece la textura.
61.         /// </summary>
62.         public virtual Texture2D Textura2D
63.         {
64.             get { return textura2D; }
65.             set
66.             {
67.                 textura2D = value;
68.                 effect.Parameters["texturaModelo"].SetValue(textura2D);
69.             }
70.         }// fin de la propiedad Textura2D
71.
72.
73.         /// <summary>
74.         /// Propiedad que obtiene o establece la matriz view.
75.         /// </summary>
76.         public virtual Matrix View
77.         {
78.             get { return view; }
79.             set
80.             {
81.                 view = value;
82.                 effect.Parameters["view"].SetValue(view);
83.             }
84.         }// fin de la propiedad View
85.
86.         /// <summary>
87.         /// Propiedad que obtiene o establece la matriz world.
88.         /// </summary>
89.         public virtual Matrix World
90.         {
91.             get { return world; }
92.             set
93.             {
94.                 world = value;
95.                 effect.Parameters["world"].SetValue(world);
96.             }
97.         }// fin de la propiedad World
98.         #region IFx Members
99.
100.        public virtual void DrawModelMeshPart(ModelMeshPart modelMeshPart,
101.            GraphicsDevice graphicsDevice)
102.        {
103.            effect.Begin();
104.            effect.CurrentTechnique.Passes[0].Begin();
105.            graphicsDevice.DrawIndexedPrimitives(PrimitiveType.TriangleList,
106.                modelMeshPart.BaseVertex, 0,
107.                modelMeshPart.NumVertices, modelMeshPart.StartIndex,
108.                modelMeshPart.PrimitiveCount);
109.            effect.CurrentTechnique.Passes[0].End();
```

```
110.            effect.End();
111.        }// fin del método DrawModelMeshPart
112.
113.        #endregion
114.    }// fin de la clase FxAmbiental
```

Queda claro que la nueva forma de asignar un efecto a la geometría es más complicada, o más grande que la anterior, sin embargo, se obtiene un mejor control de los valores de éste.

La implementación del modelo de iluminación ambiental por shader, se hará en la clase **Game1**. Como primer paso escriba las siguientes variables de instancia de la clase **Game1**.

```
GraphicsDeviceManager graphics;
Model elefante;
Matrix[] transformaciones;
FxAmbiental fxAmbiental;
Texture2D textura;
Matrix view, projection;
```

Luego hay que cambiar el tamaño del back – buffer, esto con la finalidad de seguir algunas de las recomendaciones que se hace en la documentación acerca de tamaños de pantalla para ser representados en cualquier dispositivo visual en el que se conecte el Xbox 360. El tamaño recomendado es de 1280 para el ancho y 720 para el alto.

En el constructor de la clase **Game1**, después de la asignación de la ruta del **Content**, escriba las siguientes líneas.

```
public Game1()
{
    graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = "Content";
    graphics.PreferredBackBufferHeight = 720;
    graphics.PreferredBackBufferWidth = 1280;
}
```

La propiedad **PreferredBackBufferHeight** obtiene o establece el alto del búfer, y **PreferredBackBufferWidth** hace lo mismo, pero para el alto del búfer.

En el método **Initialize**, inicalice las matrices **view** y projection, como se muestra en el siguiente enlistado, y que se deja como opción los valores propuestos aquí, pues es para visualizar toda la geometría que se dibujará en este ejemplo y los dos más que le siguen.

```
protected override void Initialize()
{
    view = Matrix.CreateLookAt(new Vector3(120.0F, 96.0F, 473.0F),
            new Vector3(75.0F, 104.0F, 7.0F), Vector3.Up);
    projection = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,
            GraphicsDevice.Viewport.AspectRatio,
            1.0F, 1000.0F);

    base.Initialize();
}
```

En el método **LoadContent** de la clase **Game1**, se cargan los datos de textura, el modelo trdimensional y el shader, los cuales no deben ser desconcidos para el lector, pues en anteriores ejemplos se utilizaron. Luego de leer los datos del archivo **Ambiental.fx** y asignarlos al objeto **fxAmbiental**, se modifican las propiedades **ColorAmbiental** y **Textura2D**.

```
protected override void LoadContent()
```

```
{
    elefante = Content.Load<Model>(@"Modelos\Elefante");
    transformaciones = new Matrix[elefante.Bones.Count];
    elefante.CopyAbsoluteBoneTransformsTo(transformaciones);

    fxAmbiental = new FxAmbiental(
            Content.Load<Effect>(@"Shaders\Ambiental"));
    fxAmbiental.ColorAmbiental = Color.Brown;
    fxAmbiental.Textura2D = textura;

    textura = Content.Load<Texture2D>(@"Texturas\Omision");
}
```

Para concluir con este ejemplo, el método **Draw** de la clase **Game1** retoma parte del primer ejemplo del Cómo cargar modelos 3D desde un archivo X y FBX, en dónde se dibuja un **ModelMeshPart** de la colección **ModelMesh** de la instancia de la clase **Model**. Sin embargo, en esta ocasión se mandarán a dibujar todos los **ModelMeshPart**.

En las líneas 5 y 6, Código 8-6, se establecen las matrices de vista y proyección del objeto **fxAmbiental**. Sin duda, estas propiedades pueden no estar aquí, pero se ha dejado así porque son algunas de las variables que pueden cambiar en una aplicación, por no decir las comunes.

El primer **foreach**, líneas 8 – 22, mapea cada elemento de la colección **Meshes** del objeto **elefante**. Enseguida se establece el valor de la propiedad **World** del objeto **fxAmbiental** con la matriz de transformación obtenida del objeto **elefante**.

Se debe establecer el búfer de índices en el dispositivo gráfico, línea 12, a partir de la instancia **mesh** que el primer **foreach** utiliza como elemento de mapeo sobre la colección **Meshes** del objeto **elefante**. La propiedad **IndexBuffer** de la clase **ModelMesh** solo obtiene dicho búfer.

En el **foreach** anidado, líneas 14 – 21, se utiliza el objeto meshPart para hacer referencia a cada elemento de la colección **MeshPart** del objeto mesh que utiliza el primer **foreach**. Éste es menester utilizarlo para establecer el búfer de vértices y la declaración de estos, líneas 16 – 18, las propiedades **VertexBuffer**, **StreamOffset**, **VertexStride** y **VertexDeclaration** son la clave para dicho éxito.

Antes de la llave que cierra el **foreach** anidado se invoca el método **DrawModelMeshPart** del objeto **fxAmbiental**, y cuyos parámetros son el **mehsPart** y el **GraphicsDevice**, línea 20.

Código 8-6

```
1.      protected override void Draw(GameTime gameTime)
2.      {
3.          GraphicsDevice.Clear(Color.White);
4.
5.          fxAmbiental.View = view;
6.          fxAmbiental.Projection = projection;
7.
8.          foreach (ModelMesh mesh in elefante.Meshes)
9.          {
10.             fxAmbiental.World = transformaciones[mesh.ParentBone.Index];
11.
12.             GraphicsDevice.Indices = mesh.IndexBuffer;
13.
14.             foreach (ModelMeshPart meshPart in mesh.MeshParts)
15.             {
16.                 GraphicsDevice.Vertices[0].SetSource(mesh.VertexBuffer,
17.                         meshPart.StreamOffset, meshPart.VertexStride);
18.                 GraphicsDevice.VertexDeclaration = meshPart.VertexDeclaration;
19.
20.                 fxAmbiental.DrawModelMeshPart(meshPart, GraphicsDevice);
21.             }
22.         }
23.
24.         base.Draw(gameTime);
```

```
25.     }
```

Ejecute el programa con la tecla **F5** y verá algo parecido a la ilustración 8 -3; corriga cualquier error de compilación si es el caso y vuleva a intentar.

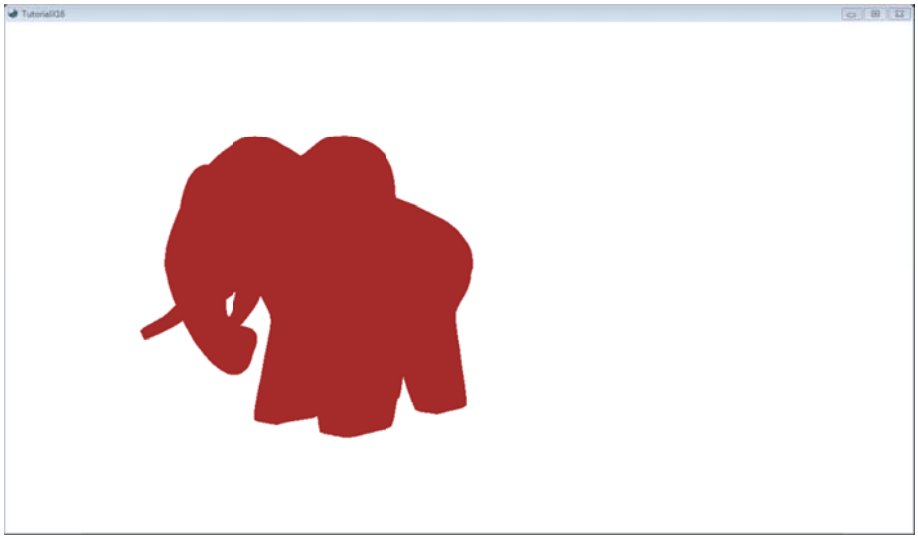

Ilustración 8-3 Integración de shader en XNA

8.2 Luz direccional

Para este ejemplo se usa el shader que implementa una luz de tipo direccional que se estudió en el capítulo anterior, empero se le agregó todas las variables **uniform** necesarias para el modelo de iluminación especular, como no es el punto de este capítulo explicar el siguiente enlistado, se deja como ejercicio entender su funcionamiento.

Escriba el siguiente código en Fx Composer y luego agréguelo en la carpeta **Shaders** de la solución anterior, en la que se creó la clase **FxAmbiental**.

Código 8-7

```
1.      /*
2.      Email: xintalalai@live.com.mx
3.      Autor: Carlos Osnaya Medrano
4.
5.      Modelo de iluminación: Especular o Difusa.
6.      Tipo de fuente: Direccional.
7.      Técnica empleada: PixelShader.
8.      Material clásico.
9.      */
10.
11.     float4x4 world : World;
12.     float4x4 view : View;
13.     float4x4 projection : Projection;
14.
15.     float3 direccionLuz
16.     <
17.         string UIName = "Dirección de Luz";
18.     >;
19.     float3 posicionCamara
20.     <
21.         string UIName = "Posición cámara";
22.     >;
23.     float ks
24.     <
25.         string UIWidget = "slider";
26.         float UIMin = 0.0F;
27.         float UIMax = 1.0F;
```

```
28.         float UIStep = 0.05F;
29.         string UIName = "Specular";
30.     > = {0.05F};
31.
32.     float n
33.     <
34.         string UIWidget = "slider";
35.         float UIMin = 0.0F;
36.         float UIMax = 128.0F;
37.         float UIStep = 1.0F;
38.         string UIName = "Specular pow";
39.     > = 5.0F;
40.
41.     float4 colorMaterialAmbiental
42.     <
43.         string UIName = "Material ambiental";
44.         string UIWidget = "Color";
45.     > = {0.05F, 0.05F, 0.05F, 1.0F};
46.     float4 colorMaterialDifuso
47.     <
48.         string UIName = "Material difuso";
49.         string UIWidget = "Color";
50.     > = {0.24F, 0.34F, 0.39F, 1.0F};
51.     float4 colorMaterialEspecular
52.     <
53.         string UIName = "Material especular";
54.         string UIWidget = "Color";
55.     > = {1.0F, 1.0F, 1.0F, 1.0F};
56.
57.     texture texturaModelo
58.     <
59.         string UIName = "Textura";
60.     >;
61.
62.     sampler modeloTexturaMuestra = sampler_state
63.     {
64.         Texture = <texturaModelo>;
65.         MinFilter = Linear;
66.         MagFilter = Linear;
67.         MipFilter = Linear;
68.         AddressU = Wrap;
69.         AddressV = Wrap;
70.     };
71.
72.     struct VertexShaderEntrada
73.     {
74.         float4 Posicion : POSITION;
75.         float3 Normal : NORMAL;
76.         float2 CoordenadaTextura : TEXCOORD0;
77.     };
78.
79.     struct VertexShaderSalida
80.     {
81.         float4 Posicion : POSITION;
82.         float2 CoordenadaTextura : TEXCOORD0;
83.         float3 WorldNormal : TEXCOORD1;
84.         float3 WorldPosition : TEXCOORD2;
85.     };
86.
87.     struct PixelShaderEntrada
88.     {
89.         float2 CoordenadaTextura : TEXCOORD0;
90.         float3 WorldNormal : TEXCOORD1;
91.         float3 WorldPosition : TEXCOORD2;
92.     };
93.     bool modIluEsp
94.     <
95.         string UIName = "Modelo Iluminación Especular";
96.     > = false;
97.
98.     VertexShaderSalida MainVS(VertexShaderEntrada entrada)
```

```
99.     {
100.        VertexShaderSalida salida;
101.        float4x4 wvp = mul(world, mul(view, projection));
102.        salida.Posicion = mul(entrada.Posicion, wvp);
103.        salida.WorldNormal = mul(entrada.Normal, world);
104.        float4 worldPosition = mul(entrada.Posicion, world);
105.        salida.WorldPosition = worldPosition / worldPosition.w;
106.        salida.CoordenadaTextura = entrada.CoordenadaTextura;
107.
108.        return salida;
109.    }// fin del vertex shader MainVS
110.
111.    float4 MainPS(PixelShaderEntrada entrada,
112.    uniform bool habilitarTextura) : COLOR
113.    {
114.        // modelo de iluminación difusa
115.        float3 dirLuz = normalize(direccionLuz);
116.        float intensidadDifusa = max(dot(-dirLuz, entrada.WorldNormal), 0.0F);
117.        float4 difuso = colorMaterialDifuso * intensidadDifusa;
118.        float4 color;
119.        // color final
120.        color = colorMaterialAmbiental + difuso;
121.
122.        // modelo de iluminación especular
123.        if(modIluEsp)
124.        {
125.            // iluminación especular
126.            float3 reflexion = normalize(2 * entrada.WorldNormal *
127.            max(dot(-dirLuz, entrada.WorldNormal), 0.0F) + dirLuz);
128.            float3 direccionCamara = normalize(posicionCamara -
129.            entrada.WorldPosition.xyz);
130.            float intensidadEspecular = pow(max(dot(reflexion,
131.                direccionCamara), 0.0F), n);
132.            float4 especular = ks * intensidadEspecular * colorMaterialEspecular;
133.
134.            color += especular * intensidadDifusa;
135.        }// fin del if
136.
137.        if(habilitarTextura)
138.        {
139.            float4 colorTextura = tex2D(modeloTexturaMuestra,
140.                entrada.CoordenadaTextura);
141.            color *= colorTextura;
142.        }
143.        // color final
144.        color.a = 1.0F;
145.        return color;
146.    }// fin pixel shader MainPS
147.
148.    technique Texturizado
149.    {
150.        pass P0
151.         {
152.            VertexShader = compile vs_3_0 MainVS();
153.            PixelShader = compile ps_3_0 MainPS(true);
154.        }
155.    }
156.
157.    technique Material
158.    {
159.        pass P0
160.        {
161.            VertexShader = compile vs_3_0 MainVS();
162.            PixelShader = compile ps_3_0 MainPS(false);
163.        }
164.    }
```

La clase **FxDireccional** es la clase que servirá como “interfaz” entre el shader anterior y XNA. Esta clase necesita las propiedades de las interfaces **IDifuso**, **IEspecular** y las hereda de la clase **FxAmbiental**; en la

Ilustración 8-4 se muestra el diagrama de clase. Creé una nueva clase en la solución anterior, llamada **FxDireccional** y escriba el Código 8-8.

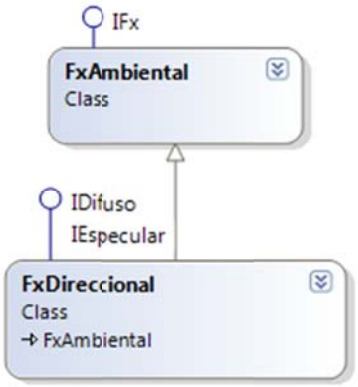

Ilustración 8-4 Diagrama de clases. FxDireccional.

Entre las líneas 3 – 9, se encuentran las variables de instancia que sirven de intermediarios entre los datos que recibe del usuario de la clase **FxDireccional** y el shader.

Vea la equivalencia entre estas variables y aquellas a las que se quiere asociar al shader, excepto en aquellas que se han declarado como estructuras **Color**, líneas 5 y 6.

En el constructor de la clase, líneas 11 – 19, se inicializan las variables de instancia a partir de sus propiedades, esto con la finalidad de darle los valores a cada elemento de la colección de **Parameters** del efecto. Cada propiedad se encargará de establecer dicho valor a cada variable **uniform extern**.

Para la propiedad **Ks**, líneas 26 – 34, coeficiente especular, no se tiene ninguna restricción en el momento de establecer el valor de la variable de instancia **ks**, que varía entre cero y uno. Esto se ha dejado así para que se pueda visualizar qué pasa con valores mayores a uno y para valores negativos. Recuerde que primero se debe asignar el dato a la variable de instancia, y luego ésta se le establece al elemento de la colección correspondiente.

La propiedad potencia especular **N**, líneas 39 – 47, tampoco restringe el valor que generalmente es entre 1 y 128. El objetivo, al igual que la propiedad **Ks**, es que se pueda visualizar con valores fuera de lo común.

Las propiedades **ColorEspecular**, líneas 52 – 60, y **ColorDifuso**, líneas 96 – 105, tienen que convertir la estructura **Color** a una estructura **Vector4** con el fin de garantizar la integración de los datos.

HabilitarEspecular, líneas 66 – 74, y **PosicionCamara**, líneas 79 – 87, solo pasan el valor correspondiente al elemento de la colección **Parameters**.

La propiedad **DireccionLuz**, líneas 112 – 121, evalúa el valor antes de asignar el dato a la variable **direccionLuz**, línea 117; si el valor es diferente de **Vector3.Zero** se le asigna a la variable **direccionLuz**, en caso contrario se le da un valor por default. Esto garantiza que siempre tendrá una dirección.

Como esta clase hereda de la clase **FxAmbiental**, las propiedades **ColorAmbiental**, **HabilitarTextura**, **Projection**, **Textura2D**, **View** y **World** deberían de sobrescribirse, debido a que el índice que se utiliza para seleccionar el elemento de la colección **Parameters** del objeto **Effect** podría cambiar. Esto depende del nombre de las variables **uniform extern** de los shaders, en caso que no coincida, el error no se detectará en tiempo de compilación sino en ejecución. Sin embargo, en todos los ejemplos de shaders se les dio un mismo nombre a cada una de las variables para permitir una programación más sencilla. La mejor práctica es agregar el modificador virtual a las propiedades base para poder cambiarlas en las clases que las hereden.

Código 8-8

```
1.      public class FxDireccional : FxAmbiental, IEspecular, IDifuso
2.      {
```

```
3.          private Single ks;
4.          private Single n;
5.          private Color colorDifuso;
6.          private Color colorEspecular;
7.          private Vector3 direccionLuz;
8.          private Boolean habilitarEspecular;
9.          private Vector3 posicionCamara;
10.
11.         public FxDireccional(Effect effect)
12.             : base(effect)
13.         {
14.             ColorDifuso = Color.WhiteSmoke;
15.             ColorEspecular = Color.WhiteSmoke;
16.             DireccionLuz = new Vector3(-1.0F, -1.0F, -1.0F);
17.             Ks = 20.0F;
18.             N = 128.0F;
19.         }// fin del constructor
20.
21.         #region IEspecular Members
22.
23.         /// <summary>
24.        /// Propiedad que obtiene o establece el coeficiente especular.
25.        /// </summary>
26.        public float Ks
27.        {
28.            get { return ks; }
29.            set
30.            {
31.                 ks = value;
32.                 base.effect.Parameters["ks"].SetValue(ks);
33.             }
34.         }// fin de la propiedad Ks
35.
36.         /// <summary>
37.         /// Propiedad que obtiene o establece la potencia
38.         /// </summary>
39.         public float N
40.         {
41.             get { return n; }
42.             set
43.             {
44.                 n = value;
45.                 base.effect.Parameters["n"].SetValue(n);
46.             }
47.         }// fin de la propiedad N
48.
49.         /// <summary>
50.         /// Propiedad que obtiene o establece el color especular del material.
51.         /// </summary>
52.         public Color ColorEspecular
53.         {
54.             get { return colorEspecular; }
55.             set
56.             {
57.                 colorEspecular = value;
58.                 base.effect.Parameters["colorMaterialEspecular"].SetValue(
59.                     colorEspecular.ToVector4());
60.             }
61.         }// fin de la propiedad ColorEspecular
62.
63.         /// <summary>
64.         /// Propiedad que habilita o inhabilita el modelo iliminación especular.
65.         /// </summary>
66.         public Boolean HabilitarEspecular
67.         {
68.             get { return habilitarEspecular; }
69.             set
70.             {
71.                 habilitarEspecular = value;
72.                 base.effect.Parameters["modIluEsp"].SetValue(habilitarEspecular);
73.             }
```

```
74.         }// fin de la propiedad DireccionLuz
75.
76.         /// <summary>
77.         /// Propiedad que obtiene o establece la posición de la cámara.
78.         /// </summary>
79.         public Vector3 PosicionCamara
80.         {
81.             get { return posicionCamara; }
82.             set
83.             {
84.                 posicionCamara = value;
85.                 effect.Parameters["posicionCamara"].SetValue(posicionCamara);
86.             }
87.         }// fin de la propiedad PosicionCamara
88.
89.         #endregion
90.
91.         #region IDifuso Members
92.
93.         /// <summary>
94.         /// Propiedad que obtiene o establece el color difuso del material.
95.         /// </summary>
96.         public Color ColorDifuso
97.         {
98.             get { return colorDifuso; }
99.             set
100.            {
101.                colorDifuso = value;
102.                base.effect.Parameters["colorMaterialDifuso"].SetValue(
103.                colorDifuso.ToVector4());
104.            }
105.        }// fin de la propiedad ColorDifuso
106.
107.        #endregion
108.
109.        /// <summary>
110.        /// Propiedad que obtiene o establece la dirección de luz.
111.        /// </summary>
112.        public Vector3 DireccionLuz
113.        {
114.            get { return direccionLuz; }
115.            set
116.            {
117.                direccionLuz = (value != Vector3.Zero) ? value :
118.                    new Vector3(-1.0F, -1.0F, -1.0F);
119.                base.effect.Parameters["direccionLuz"].SetValue(direccionLuz);
120.            }
121.        }// fin de la propiedad DireccionLuz
122.    }// fin de la clase FxDireccional
```

La implementación de la clase **FxAmbiental** se realiza en la clase **Game1**. Abra el archivo **Game1.cs** y escriba dos variables de instancia nuevas.

```
FxDireccional fxDireccional;
Vector3 posicion;
```

La variable **posicion** sirve para establecer el lugar de la cámara en la propiedad **PosicionCamara** del objeto **fxDireccional**.

En el método **Initialize** se inicializa la estructura posicion para luego dársela al método **CreateLookAt** como uno de sus parámetros.

```
protected override void Initialize()
{
    posicion = new Vector3(-120.0F, 96.0F, 473.0F);
    view = Matrix.CreateLookAt(posicion,
```

```
            new Vector3(75.0F, 104.0F, 7.0F), Vector3.Up);
        projection = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,
            GraphicsDevice.Viewport.AspectRatio,
            1.0F, 1000.0F);

        base.Initialize();
    }
```

En el método **LoadContent** se crea la instancia **fxAmbiental** y se inicializan algunas de sus propiedades, véase el siguiente código.

```
protected override void LoadContent()
{
    elefante = Content.Load<Model>(@"Modelos\Elefante");
    transformaciones = new Matrix[elefante.Bones.Count];
    elefante.CopyAbsoluteBoneTransformsTo(transformaciones);

    fxAmbiental = new FxAmbiental(
            Content.Load<Effect>(@"Shaders\Ambiental"));
    fxAmbiental.ColorAmbiental = Color.Brown;
    fxAmbiental.Textura2D = textura;

    textura = Content.Load<Texture2D>(@"Texturas\Omision");

    fxDireccional = new FxDireccional(Content.Load<Effect>(@"Shaders\Direccional"));
    fxDireccional.Textura2D = textura;
    fxDireccional.ColorAmbiental = Color.Black;
    fxDireccional.ColorDifuso = new Color(5, 2, 2);
    fxDireccional.ColorEspecular = Color.Brown;
    fxDireccional.DireccionLuz = new Vector3(0, -1, -1);
    fxDireccional.HabilitarEspecular = true;
    fxDireccional.HabilitarTextura = true;
    fxDireccional.Ks = 10;
    fxDireccional.N = 120;
    fxDireccional.PosicionCamara = posicion;
}
```

En el método **Draw**,Código 8-9, se actualizan las propiedades **View**, línea 8; **Projection**, línea 9; y **World**, líneas 13 y 24; del objeto **fxDireccional**.

Antes de llamar al método **DrawModelMeshPart** de la instancia **FxDireccional**, línea 25, se traslada la geometría menos cien unidades sobre el eje **x** y menos doscientas unidades sobre el eje **z**. Corra el ejemplo con la tecla **F5** y verá algo similar a la Ilustración 8-5.

Código 8-9

```
1.      protected override void Draw(GameTime gameTime)
2.      {
3.          GraphicsDevice.Clear(Color.White);
4.
5.          fxAmbiental.View = view;
6.          fxAmbiental.Projection = projection;
7.
8.          fxDireccional.View = fxAmbiental.View;
9.          fxDireccional.Projection = fxAmbiental.Projection;
10.         foreach (ModelMesh mesh in elefante.Meshes)
11.         {
12.             fxAmbiental.World = transformaciones[mesh.ParentBone.Index];
13.             fxDireccional.World = fxAmbiental.World;
14.
15.             GraphicsDevice.Indices = mesh.IndexBuffer;
16.
17.             foreach (ModelMeshPart meshPart in mesh.MeshParts)
18.             {
19.                 GraphicsDevice.Vertices[0].SetSource(mesh.VertexBuffer,
20.                         meshPart.StreamOffset, meshPart.VertexStride);
```

```
21.                 GraphicsDevice.VertexDeclaration = meshPart.VertexDeclaration;
22.
23.                 fxAmbiental.DrawModelMeshPart(meshPart, GraphicsDevice);
24.                 fxDireccional.World *= Matrix.CreateTranslation(-100, 0, -200);
25.                 fxDireccional.DrawModelMeshPart(meshPart, GraphicsDevice);
26.             }
27.         }
28.
29.         base.Draw(gameTime);
30.     }
```

Si ocurre un error de compilación o de ejecución revisé de nueva cuenta el código, y vuelva intentar.

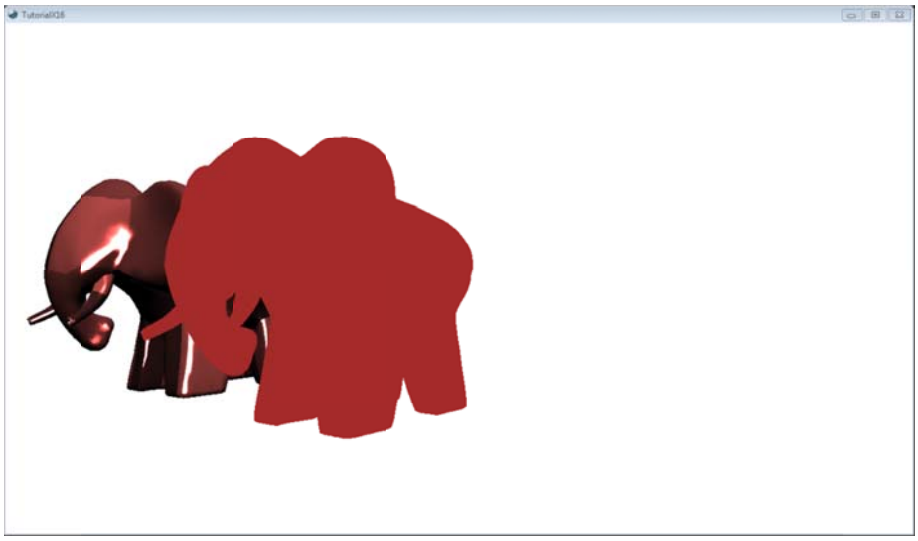

Ilustración 8-5 Integración de los shaders Ambiental y Direccional en XNA

8.3 Efecto multiluces

Para este último ejemplo se utiliza una versión reducida del shader **MultiLuces** que se vio en el capítulo anterior. Este nuevo shader solo contiene un tipo de fuente: **la direccional**. Esto es con el fin que se entienda cómo se crea la clase que comunica los datos de XNA hacia el efecto, y se deja como ejercicio al lector crear la clase correspondiente para la versión completa de **MultiLuces**.

En el listado siguiente se muestra la versión reducida de **MultiLuces**, y se espera que el lector pueda interpretar con facilidad cada línea, pues la explicación ya fue comentada en el capítulo anterior.

Abra Fx Composer y creé un nuevo proyecto para escribir las siguientes líneas en un archivo **fx**.

Código 8-10

```
1.      /*
2.      Email: xintalalai@live.com.mx
3.      Autor: Carlos Osnaya Medrano
4.      Multiples luces.
5.      Modelo de iluminación: Especular o Difusa.
6.      Tipo de fuente: Direccional.
7.      Técnica empleada: PixelShader.
8.      Material clásico.
9.      */
10.
11.     float4x4 world: World;
12.     float4x4 view : View;
13.     float4x4 projection : Projection;
14.
15.     float3 posicionCamara
16.     <
17.         string UIName = "Posición cámara";
```

```
18.     >;
19.
20.     float ks
21.     <
22.         string UIWidget = "slider";
23.         float UIMin = 0.0F;
24.         float UIMax = 10.0F;
25.         float UIStep = 0.05F;
26.         string UIName = "Coeficiente de reflexión especular";
27.     > = {0.05F};
28.
29.     float n
30.     <
31.         string UIWidget = "slider";
32.         float UIMin = 0.0F;
33.         float UIMax = 128.0F;
34.         float UIStep = 1.0F;
35.         string UIName = "Exponente de reflexión especular";
36.     > = 5.0F;
37.
38.     float4 colorMaterialAmbiental
39.     <
40.         string UIName = "Material ambiental";
41.         string UIWidget = "Color";
42.     >  = {0.07F, 0.07F, 0.07F, 1.0F};
43.
44.     float4 colorMaterialDifuso
45.     <
46.         string UIName = "Color Material";
47.         string UIWidget = "Color";
48.     > = {0.24F ,0.34F, 0.39F, 1.0F};
49.
50.     float4 colorMaterialEspecular
51.     <
52.         string UIName = "Material especular";
53.         string UIWidget = "Color";
54.     > = {1.0F, 1.0F, 1.0F, 1.0F};
55.
56.     bool habiEspec
57.     <
58.         string UIName = "Modelo Iluminación Especular";
59.     > = false;
60.
61.     texture2D texturaModelo;
62.     sampler modeloTexturaMuestra = sampler_state
63.     {
64.         Texture = <texturaModelo>;
65.         MinFilter = Linear;
66.         MagFilter = Linear;
67.         MipFilter = Linear;
68.         AddressU = Wrap;
69.         AddressV = Wrap;
70.     };
71.
72.     struct VertexShaderSalida
73.     {
74.         float4 Posicion : POSITION;
75.         float2 CoordenadaTextura : TEXCOORD0;
76.         float3 WorldNormal : TEXCOORD1;
77.         float3 WorldPosition : TEXCOORD2;
78.     };
79.
80.     struct PixelShaderEntrada
81.     {
82.         float2 CoordenadaTextura : TEXCOORD0;
83.         float3 WorldNormal : TEXCOORD1;
84.         float3 WorldPosition : TEXCOORD2;
85.     };
86.
87.     struct LuzDireccional
88.     {
```

```
89.         float3 Direccion;
90.         float4 Color;
91.         int Encender;
92.     };
93.
94.     int numLuces = 4;
95.
96.     VertexShaderSalida MainVS(float3 posicion : POSITION,
97.         float3 normal : NORMAL, float2 coordenadaTextura : TEXCOORD0)
98.     {
99.         VertexShaderSalida salida;
100.        float4x4 mvp = mul(world, mul(view, projection));
101.        salida.Posicion = mul(float4(posicion, 1.0F), mvp);
102.        salida.WorldNormal = mul(normal, world);
103.        salida.WorldPosition = mul(float4(posicion, 1.0F), world);
104.        salida.CoordenadaTextura = coordenadaTextura;
105.
106.        return salida;
107.    }// fin del vertexshader MainVS
108.
109.    float4 MainPSAmbiental(PixelShaderEntrada entrada,
110.        uniform bool habiTextura) : COLOR
111.    {
112.        float4 color = colorMaterialAmbiental;
113.        if(habiTextura)
114.        {
115.            color *=  tex2D(modeloTexturaMuestra, entrada.CoordenadaTextura);
116.        }
117.        return color;
118.    }// fin del pixelshader MainPSAmbiental
119.
120.    // Función que calcula la reflexión especular.
121.    float4 ReflexionEspecular(float3 direccionLuz, float3 worldPosicion,
122.        float3 worldNormal, float4 colorLuz)
123.    {
124.        float3 reflexion = normalize(2 * worldNormal *
125.        max(dot(direccionLuz, worldNormal), 0.0F) - direccionLuz);
126.
127.        float3 direccionCamara = normalize(posicionCamara - worldPosicion);
128.        float reflexionEspecular = pow(max(dot(reflexion, direccionCamara), 0.0F), n);
129.
130.        return (ks * reflexionEspecular) * (colorLuz * colorMaterialEspecular);
131.    }// fin de la función ReflexionEspecular
132.
133.    float4 Phong(float3 direccionLuz, float3 worldPosicion,
134.        float3 worldNormal, float4 colorLuz, float4 colorMater)
135.    {
136.        float reflexionDifusa = max(dot(direccionLuz, worldNormal), 0.0F);
137.        float4 phong = reflexionDifusa * colorLuz * colorMater;
138.        // activación de la reflexión especular
139.        if(habiEspec)
140.        {
141.            phong += ReflexionEspecular(direccionLuz, worldPosicion,
142.                worldNormal, colorLuz) * colorMater * reflexionDifusa;
143.        }// fin del if
144.
145.        return phong;
146.    }// fin de la función Phong
147.
148.    // Función que calcula la luz tipo direccional
149.    float4 FuenteDireccional(LuzDireccional luz, float3 worldPosicion,
150.        float3 worldNormal, float4 colorMater)
151.    {
152.        return Phong(-normalize(luz.Direccion), // dirección de luz
153.            worldPosicion,
154.            worldNormal, luz.Color,
155.            colorMater);
156.    }// fin de la función FuenteDireccional
157.
158.    LuzDireccional direccionales[20];
159.
```

```
160.    float4 MainPS(PixelShaderEntrada entrada, uniform bool texturizado) : COLOR
161.    {
162.        float4 color = 0;
163.        float4 colorDifuso = colorMaterialDifuso;
164.
165.        if(texturizado)
166.        {
167.            colorDifuso *= tex2D(modeloTexturaMuestra, entrada.CoordenadaTextura);
168.        }// fin del if
169.
170.        for(int i = 0; i < numLuces; i++)
171.        {
172.            if(direccionales[i].Encender != 0)
173.            {
174.                color += FuenteDireccional(direccionales[i], entrada.WorldPosition,
175.                    entrada.WorldNormal, colorDifuso);
176.            }
177.        }// fin del for
178.        color.a = 1.0F;
179.        return color;
180.    }// fin del pixelShader mainPS
181.
182.    technique Texturizado
183.    {
184.        pass Ambiental
185.        {
186.            VertexShader = compile vs_3_0 MainVS();
187.            PixelShader = compile ps_3_0 MainPSAmbiental(true);
188.        }
189.        pass MultiLuces
190.        {
191.            VertexShader = compile vs_3_0 MainVS();
192.            PixelShader = compile ps_3_0 MainPS(true);
193.        }
194.    }// fin de la técnica Texturizado
195.
196.    technique Material
197.    {
198.        pass Ambiental
199.        {
200.            VertexShader = compile vs_3_0 MainVS();
201.            PixelShader = compile ps_3_0 MainPSAmbiental(false);
202.        }
203.
204.        pass MultiLuces
205.        {
206.            VertexShader = compile vs_3_0 MainVS();
207.            PixelShader = compile ps_3_0 MainPS(false);
208.        }
209.    }// fin de la técnica Texturizado
```

Continuando con el proyecto de Visual Studio se deben agregar dos clases, llamadas **FxMultiluces** y **LuzDireccional**. La primera representa la clase que comunicará los datos desde XNA hacia el shader, y la segunda representará los campos de la estructura **LuzDireccional** del shader.

En la Ilustración 8-6, se muestra el diagrama de clase para **FxMultiluces**. Ésta hereda de la clase **FxAmbiental** y de las interfaces **IDifuso** e **IEspecular**. Lo que hace pensar que dicha herencia de clase pudo haberse dado de **FxDireccional** hacia **FxMultiluces**. Pero la propiedad **DireccionLuz** de **FxDireccional** no se encuentra como miembro de la clase **FxMultiluces**, y por eso se consideran como clases diferentes.

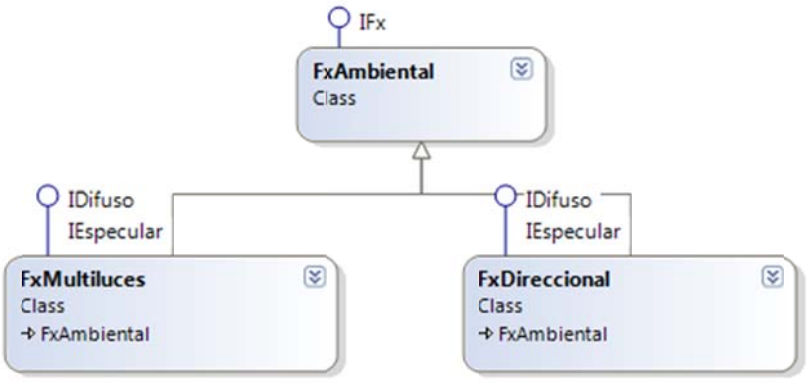

Ilustración 8-6 Diagrama de clases. FxMultiluces

Antes de escribir la clase **FxMultiluces** hay que definir los campos de la clase **LuzDireccional** que utiliza **FxMultiluces** como dependencia.

En las líneas 3 a 6, Código 8-11, se declaran las variables de instancia **color**, **dirección**, **encender** y **estrucutraLuz**. Las tres primeras corresponden a la estructura que **LuzDireccional** del shader **Multluces.fx**. Pero si uno observa el campo **Encender** de dicha estructura en el shader no es del tipo booleano, sino entera. Para conservar la integridad de los datos, en las propiedades, la asignación de dichos valores se hace adecuadamente y dependiendo de la información.

La variable **estructuraLuz**, línea 6, representa un elemento del parámetro direccionales del shader, éste es un arreglo de veinte estrucuturas **LuzDireccional**, así que cada una tiene tres elementos a los cuales se les puede asignar un valor.

El constructor de la clase **LuzDireccional**, líneas 8 – 14, establece la dirección de la luz, el color e inhabilita la luz. Por default todas las luces estarán apagadas.

Para la propiedad **Color**, líneas 19 – 28, se transforma la estructura **Color** por **Vector4**. En la propiedad **DireccionLuz** se evita que se establezca un valor tipo **Vector3.Zero**.

En el descriptor de acceso **set**, líneas 50 – 55 de la propiedad **Encender**, se establece el valor a uno si el valor de la variable encender es **true** y en caso contrario sería cero.

Código 8-11

```
1.      public class LuzDireccional
2.      {
3.          private Color color;
4.          private Vector3 direccion;
5.          private Boolean encender;
6.          private EffectParameter estructuraLuz;
7.
8.          public LuzDireccional(EffectParameter effectParameter, Vector3 direccion)
9.          {
10.             estructuraLuz = effectParameter;
11.             Direccion = direccion;
12.             Color = Color.WhiteSmoke;
13.             Encender = false;
14.         }
15.
16.         /// <summary>
17.         /// Propiedad que obtiene o establece el color de la luz.
18.         /// </summary>
19.         public Color Color
20.         {
21.             get { return color; }
22.             set
23.             {
24.                 color = value;
```

```
25.                 estructuraLuz.StructureMembers["Color"].SetValue(
26.                     color.ToVector4());
27.             }
28.         }// fin de la propiedad Color
29.
30.         /// <summary>
31.         /// Propiedad que obtiene o establece el vector de dirección de la luz.
32.         /// </summary>
33.         public Vector3 Direccion
34.         {
35.             get { return direccion; }
36.             set
37.             {
38.                 direccion = (value != Vector3.Zero) ? value : new Vector3(-1.0F,
39.                     -1.0F, -1.0F);
40.                 estructuraLuz.StructureMembers["Direccion"].SetValue(direccion);
41.             }
42.         }// fin de la propiedad Direccion
43.
44.         /// <summary>
45.         /// Propiedad que enciende o apaga la luz.
46.         /// </summary>
47.         public Boolean Encender
48.         {
49.             get { return encender; }
50.             set
51.             {
52.                 encender = value;
53.                 estructuraLuz.StructureMembers["Encender"].SetValue(
54.                     (encender) ? 1 : 0);
55.             }
56.         }// fin de la propiedad Encender
57.     }// fin de la clase LuzDireccional
```

Agregue una nueva clase en la solución en Visual Studio cuyo nombre sea **FxMultiluces**, y escriba el Código 8-12.

Cada una de las variables del efecto tienen su álter ego en las variables de instancia, líneas 3 – 10. Como se ha manejado anteriormente no todas las variables corresponden al mismo tipo de dato, sin embargo la integridad se conserva en las propiedades.

En el constructor se inicializan los colores del material, el coeficiente especular y su potencia; se inhabilita la textura y se le asigna el parámetro correspondiente al arreglo que tiene las estructuras de la luz direccional, línea 21.

La definición de las propiedades, en su mayoría, son las mismas que en la clase **FxDireccional**, así que solo se explicara el método **DrawModelMeshPart**, líneas 131 – 168

Este shader contiene dos pasadas por técnica y deben ser llamadas una a una en el método **DrawModelMeshPart** de la clase **FxAmbiental**, para sumar los efectos de ambas pasadas. Para lograr dicho efecto se debe inhabilitar la escritura sobre el **DepthBuffer** y habilitar el canal alfa; luego se deben regresar a un estado anterior, para no afectar a las demás geometrías. El cambio debe hacerse solo para la clase **FxMultiLuces**, así que se deberá reescribir el método **DrawModelMeshPart** que se hereda de la clase **FxAmbiental**, líneas 131 – 168.

En el método **DrawModelMeshPart** se hace llamar dos veces el método **DrawIndexedPrimitives** de la instancia **graphicsDevice**, líneas 138 -141, y líneas 158 – 161, cada una de ellas se encuentra entre los métodos **Begin** y **End** de cada elemento **EffectPass** que corresponde a **Ambiental** y **Multiluces** del shader.

Despúes de llamar por primera vez al método **DrawIndexedPrimitives**, se habilita la transparencia, línea 145, y junto con ella todas las propiedades que van de la mano del **blending** .

Enseguida se llama el método **DrawIndexedPrimitives**, no sin antes propagar el efecto anterior al dispositivo gráfico antes del renderizado. Para lograr dicho fin, se llama el método **CommitChanges**, línea 153.

El método **CommitChanges**, debe llamarse dentro del par de métodos **Begin** y **End** de la pasada actual, y antes de varios métodos **DrawPrimitives**, que sirven como propagadores de los cambios hacia el dispositivo gráfico.

Antes de llamar el método **End**, línea 164, se inhabilita el **blending** para no afectar las goemetrias posteriores al llamado del método **DrawModelMeshPart**.

Código 8-12

```
1.      public class FxMultiluces : FxAmbiental, IEspecular, IDifuso
2.      {
3.          private Color colorDifuso;
4.          private Color colorEspecular;
5.          private Boolean habilitarEspecular;
6.          private Single ks;
7.          private Single n;
8.          private Int16 numeroLuces;
9.          private EffectParameter direccionales;
10.         private Vector3 posicionCamara;
11.
12.         public FxMultiluces(Effect effect)
13.             : base(effect)
14.         {
15.             ColorDifuso = Color.Black;
16.             ColorDifuso = Color.WhiteSmoke;
17.             ColorEspecular = Color.WhiteSmoke;
18.             Ks = 10.0F;
19.             N = 20.0F;
20.             HabilitarTextura = false;
21.             direccionales = effect.Parameters["direccionales"];
22.         }
23.
24.         #region IEspecular Members
25.
26.         /// <summary>
27.         /// Propiedad que obtiene o establece el coeficiente especular.
28.         /// </summary>
29.         public float Ks
30.         {
31.             get { return ks; }
32.             set
33.             {
34.                 ks = value;
35.                 base.effect.Parameters["ks"].SetValue(ks);
36.             }
37.         }// fin de la propiedad Ks
38.
39.         /// <summary>
40.         /// Propiedad que obtiene o establece la potencia especular.
41.         /// </summary>
42.         public float N
43.         {
44.             get { return n; }
45.             set
46.             {
47.                 n = value;
48.                 base.effect.Parameters["n"].SetValue(n);
49.             }
50.         }// fin de la propiedad N
51.
52.         /// <summary>
53.         /// Propiedad  que obtiene o establece el color especular.
54.         /// </summary>
55.         public Color ColorEspecular
56.         {
57.             get { return colorEspecular; }
58.             set
59.             {
60.                 colorEspecular = value;
61.                 base.effect.Parameters["colorMaterialEspecular"].SetValue(
```

```
62.                     colorEspecular.ToVector4());
63.             }
64.         }// fin de la propiedad ColorEspecular
65.
66.         /// <summary>
67.         /// Propiedad que habilita o inhabilita el modelo de iluminación especular.
68.         /// </summary>
69.         public bool HabilitarEspecular
70.         {
71.             get { return habilitarEspecular; }
72.             set
73.             {
74.                 habilitarEspecular = value;
75.                 effect.Parameters["habiEspec"].SetValue(habilitarEspecular);
76.             }
77.         }// fin de la propiedad HabilitarEspecular
78.
79.         /// <summary>
80.         /// Propiedad que obtiene o establece la posicion de la cámara.
81.         /// </summary>
82.         public Vector3 PosicionCamara
83.         {
84.             get { return posicionCamara; }
85.             set
86.             {
87.                 posicionCamara = value;
88.                 effect.Parameters["posicionCamara"].SetValue(posicionCamara);
89.             }
90.         }// fin de la propiedad PosicionCamara
91.
92.         #endregion
93.
94.         #region IDifuso Members
95.
96.         /// <summary>
97.         /// Propiedad que obtiene o establece el color difuso.
98.         /// </summary>
99.         public Color ColorDifuso
100.        {
101.            get { return colorDifuso; }
102.            set
103.            {
104.                colorDifuso = value;
105.                base.effect.Parameters["colorMaterialDifuso"].SetValue(
106.                    colorDifuso.ToVector4());
107.            }
108.        }// fin de la propiedad ColorDifuso
109.
110.        #endregion
111.
112.        /// <summary>
113.        /// Propiedad que obtiene o establece el número de luces.
114.        /// </summary>
115.        public Int16 NumeroLuces
116.        {
117.            get { return numeroLuces; }
118.            set
119.            {
120.                numeroLuces = value;
121.                base.effect.Parameters["numLuces"].SetValue(numeroLuces);
122.            }
123.        }// fin de la propiedad NumeroLuces
124.
125.        /// <summary>
126.        /// Propiedad que obtiene el parámetro de la estructura
127.        /// que corresponde a las luces direccionales.
128.        /// </summary>
129.        public EffectParameter Direccionales { get { return direccionales; } }
130.
131.        public override void DrawModelMeshPart(ModelMeshPart modelMeshPart,
132.            GraphicsDevice graphicsDevice)
```

```
133.        {
134.            effect.Begin();
135.            #region Effect.Begin
136.
137.            effect.CurrentTechnique.Passes["Ambiental"].Begin();
138.            graphicsDevice.DrawIndexedPrimitives(PrimitiveType.TriangleList,
139.                modelMeshPart.BaseVertex, 0,
140.                modelMeshPart.NumVertices, modelMeshPart.StartIndex,
141.                modelMeshPart.PrimitiveCount);
142.            effect.CurrentTechnique.Passes["Ambiental"].End();
143.
144.            // se habilita el blending
145.            graphicsDevice.RenderState.AlphaBlendEnable = true;
146.            graphicsDevice.RenderState.BlendFunction =
147.                BlendFunction.Add;
148.            graphicsDevice.RenderState.DestinationBlend = Blend.One;
149.            graphicsDevice.RenderState.SourceBlend = Blend.One;
150.
151.            effect.CurrentTechnique.Passes["MultiLuces"].Begin();
152.
153.            effect.CommitChanges();
154.
155.            graphicsDevice.DrawIndexedPrimitives(PrimitiveType.TriangleList,
156.                modelMeshPart.BaseVertex, 0,
157.                modelMeshPart.NumVertices, modelMeshPart.StartIndex,
158.                modelMeshPart.PrimitiveCount);
159.            effect.CurrentTechnique.Passes["MultiLuces"].End();
160.
161.             // valor por default de XNA
162.            graphicsDevice.RenderState.AlphaBlendEnable = false;
163.            #endregion
164.            effect.End();
165.        }// fin del método DrawModelMeshPart
166.    }// fin de la clase FxMultiluces
```

Para implementar esta última clase, se debe crear el objeto de la clase **Multiluces** y las luces de tipo direccional. En la clase **Game1** de la solución, escriba las siguientes variables de instancia que indican dichos objetos.

```
FxMultiluces fxMultiluces;
LuzDireccional[] lucesDireccionales;
```

Y en el método **LoadContent**, de la clase **Game1**, se carga el shader en el constructor de la clase **FxMultiluces**, línea 1, Código 8-13. Luego se inicializan las propiedades del objeto **fxMultilices**, líneas 2 – 10.

La creación de las luces direccionales se hace dentro del mismo método **LoadContent**, pues su constructor necesita la dirección de memoria de cada elemento que indica una luz del shader, líneas 12 – 24. Luego se inicializan las propiedades de cada fuente de iluminación, líneas 26 – 37.

Código 8-13

```
1.      fxMultiluces = new FxMultiluces(Content.Load<Effect>(@"Shaders\MulDirec"));
2.      fxMultiluces.NumeroLuces = 6;
3.      fxMultiluces.Textura2D = textura;
4.      fxMultiluces.HabilitarEspecular = true;
5.      fxMultiluces.HabilitarTextura = true;
6.      fxMultiluces.ColorDifuso = new Color(3, 3, 3);
7.      fxMultiluces.ColorAmbiental = Color.Black;
8.      fxMultiluces.Ks = 10;
9.      fxMultiluces.N = 20;
10.     fxMultiluces.PosicionCamara = posicion;
11.
12.     lucesDireccionales = new LuzDireccional[fxMultiluces.NumeroLuces];
13.     lucesDireccionales[0] = new LuzDireccional(
14.     fxMultiluces.Direccionales.Elements[0], new Vector3(0, -1, 0));
15.     lucesDireccionales[1] = new LuzDireccional(
16.     fxMultiluces.Direccionales.Elements[1], new Vector3(0, 1, 0));
```

```
17.     lucesDireccionales[2] = new LuzDireccional(
18.     fxMultiluces.Direccionales.Elements[2], new Vector3(0, 0, 1));
19.     lucesDireccionales[3] = new LuzDireccional(
20.     fxMultiluces.Direccionales.Elements[3], new Vector3(0, 0, -1));
21.     lucesDireccionales[4] = new LuzDireccional(
22.     fxMultiluces.Direccionales.Elements[4], new Vector3(1, 0, 0));
23.     lucesDireccionales[5] = new LuzDireccional(
24.     fxMultiluces.Direccionales.Elements[5], new Vector3(-1, 0, 0));
25.
26.     lucesDireccionales[0].Encender = true;
27.     lucesDireccionales[0].Color = Color.Green;
28.     lucesDireccionales[1].Encender = true;
29.     lucesDireccionales[1].Color = Color.LawnGreen;
30.     lucesDireccionales[2].Encender = true;
31.     lucesDireccionales[2].Color = Color.Indigo;
32.     lucesDireccionales[3].Encender = true;
33.     lucesDireccionales[3].Color = Color.Brown;
34.     lucesDireccionales[4].Encender = true;
35.     lucesDireccionales[4].Color = Color.Chocolate;
36.     lucesDireccionales[5].Encender = true;
37.     lucesDireccionales[5].Color = Color.LemonChiffon;
```

Para concluir con este ejemplo, en el método **Draw** de la clase **Game1**, Código 8-14, se establecen las matrices de mundo, línea 17; vista, línea 10; y proyección, línea 11; del objeto **fxMultiluces**. Y se llama al método **DrawModelMeshPart** de la instancia **fxMultiluces**, línea 32.

Ejecute el programa, y corrija cualquier error de compilación que surgiese y vuelva a intentarlo, verá algo parecido a la ilustración 8 – 7.

Código 8-14

```
1.      protected override void Draw(GameTime gameTime)
2.      {
3.          GraphicsDevice.Clear(Color.White);
4.
5.          fxAmbiental.View = view;
6.          fxAmbiental.Projection = projection;
7.
8.          fxDireccional.View = fxAmbiental.View;
9.          fxDireccional.Projection = fxAmbiental.Projection;
10.         fxMultiluces.View = fxAmbiental.View;
11.         fxMultiluces.Projection = fxAmbiental.Projection;
12.
13.         foreach (ModelMesh mesh in elefante.Meshes)
14.         {
15.             fxAmbiental.World = transformaciones[mesh.ParentBone.Index];
16.             fxDireccional.World = fxAmbiental.World;
17.             fxMultiluces.World = fxAmbiental.World;
18.
19.
20.             GraphicsDevice.Indices = mesh.IndexBuffer;
21.
22.             foreach (ModelMeshPart meshPart in mesh.MeshParts)
23.             {
24.                 GraphicsDevice.Vertices[0].SetSource(mesh.VertexBuffer,
25.                         meshPart.StreamOffset, meshPart.VertexStride);
26.                 GraphicsDevice.VertexDeclaration = meshPart.VertexDeclaration;
27.
28.                 fxAmbiental.DrawModelMeshPart(meshPart, GraphicsDevice);
29.                 fxDireccional.World *= Matrix.CreateTranslation(-100, 0, -200);
30.                 fxDireccional.DrawModelMeshPart(meshPart, GraphicsDevice);
31.                 fxMultiluces.World *= Matrix.CreateTranslation(100, 0, 200);
32.                 fxMultiluces.DrawModelMeshPart(meshPart, GraphicsDevice);
33.             }
34.         }
35.
36.         base.Draw(gameTime);
37.     }
```

Este último ejemplo demuestra que tan complicado se puede hacer la comunicación entre los datos de XNA hacia el shader, además representa un gran número de líneas de código en comparación con el primer ejemplo que se vió en este capítulo, pero que sin duda es mejor.

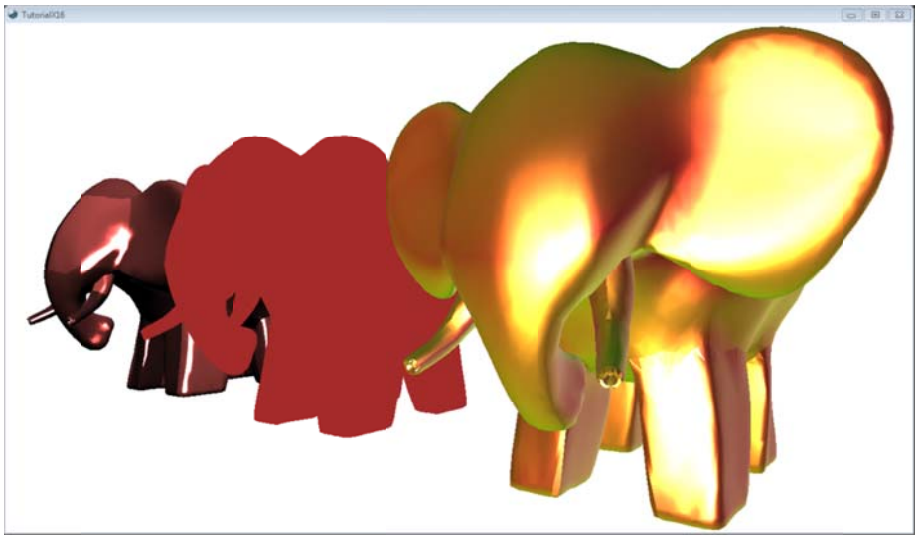

Ilustración 8-7 Integración del shader Multiluces en XNA

Como ejercicio para el lector se deja crear la clase que comunique los datos desde XNA hacia el shader MultiLuces que se vio en el apartado Iluminación, en donde se tienen diferentes fuentes de iluminación.