4 Textura

Las texturas sirven como papel tapiz que se “pegan” sobre los modelos 3D para darle un aspecto más real, sin embargo, existen técnicas más avanzadas para generar la sensación de realismo y que se basan en el shader pero que no reemplazan a las texturas, las mejoran.

En este capítulo se mostrará un ejemplo sencillo, un plano formado por dos triángulos y texturizado.

Las texturas son simplemente imágenes digitalizadas que deben tener ciertas características para que la tarjeta gráfica pueda soportarla. Anteriormente el tamaño de las imágenes no pasaba de los 512x512 pixeles, empero la evolución de las GPUs (Graphics Processing Unit) dió cabida para experimentar con mejores imágenes. Así que hasta ahora las imágenes deberán medir con potencias de 2, es decir, imágenes que tengan las siguientes medidas serán aceptadas: 512x512, 512x1024, 256x512, 2048x1024, 1024zx1024, etcétera.

Los formatos de imágenes que podrán agregar en la solución de un proyecto de XNA se muestran en la Tabla 4-1.[6]

Tabla 4-1

Formato	Significado
Bmp	Microsoft Windows bitmap.
Dds	DirectDrawSurface.
Dib	Microsoft Windows bitmap.
Hdr	High dynamic-range.
Jpg	Joint Photographic Experts Group (JPEG) compressed
Pfm	Portable float map.
Png	Portable Network Graphics.
Ppm	Portable pixmap.
Tga	Truevision Targa image.

Cada formato de imagen tiene sus características particulares que las hacen importantes para cada fin; mientras unos son prácticamente una matriz de datos, lo que los hacen mucho más grandes en bits y más completos; otros son menos grandes en bits pero con menor información y además agregan sus códigos para ser decodificados; otros tantos agregan funciones como el canal alfa para crear sprites. Así que en el momento de seleccionar el formato de las imágenes hay que tener en cuenta la calidad, el tamaño en bits y el tiempo que tarda en cargarse. Este texto no abarca este tipo de cuestiones, pues está fuera de su alcance.

4.1 El téxel

De la misma manera que las imágenes tienen al pixel como unidad de medida, para altura y anchura, las texturas tienen al téxel como su unidad, y en muchas de las veces es igual a uno. Además, pueden ser leídos

[6] Para mayor información sobre los formatos de imágenes visite la siguiente página web: http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.graphics.imagefileformat.aspx

o escritos desde una GPU. Específicamente, un téxel puede ser cualquiera de los formatos de textura disponible representados en la numeración **SurfaceFormat**.[7]

Para conocer qué parte de la textura le corresponde a un vértice es necesario asignarle una coordenada de textura; las coordenadas de texturas se llaman **u** y **v**. El origen de estas coordenadas se sitúa en la esquina superior izquierda de la imagen; la coordenada **u** aumenta hacia la derecha y la coordenada **v** aumenta hacia abajo. La Ilustración 4-1 muestra en cada esquina las coordenadas que le corresponderían, en el caso que la misma figura fuera la unidad.

La esquina superior izquierda el valor de las coordenadas son las mismas, cero; en la esquina superior derecha el valor de la coordenada **u** es uno; en la esquina inferior izquierda la coordenada **v** es uno y la coordenada **u** es cero; y por último, en la esquina inferior derecha las coordenadas tienen el mismo valor, uno.

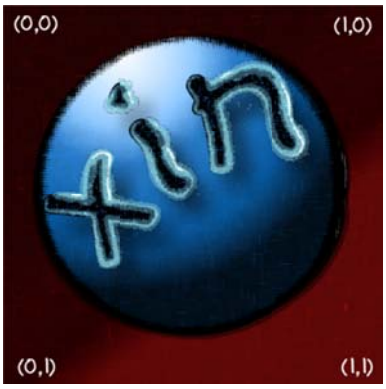

Ilustración 4-1 Textura

4.2 Filtros

Las geometrías no siempre coincidirán con el tamaño de las texturas, es decir, la distancia entre dos vértices puede medir más o medir menos, comparada con una unidad de textura, por lo que puede haber un mayor o menor número de pixeles asociados al téxel. Por lo tanto existen dos modos de sampleo o de muestreo, el de magnificación y el de minificación, que permiten seleccionar el color del pixel final. Los filtros se encargan de calcular ese color del pixel, y XNA ofrece varios de éstos, que están enumerados en **TextureFilter**[8], véase la Tabla 4-2. Tratar de explicar cómo funciona cada filtro está fuera del alcance de este texto; si quiere conocer más, acerca de los tipos de filtros, busque en libros de procesamiento digital de imágenes.

Tabla 4-2

Miembro	Descripción
Anisotropic	*Filtro de textura anisotrópico que se utiliza como filtro de ampliación o reducción de la textura. Este tipo de filtro compensa la distorsión producida por la diferencia de ángulo entre el polígono de la textura y el plano de la pantalla.*[9]
GaussianQuad	Es el filtro Gaussiano con máscara de 4 x 4 para la magnificación y la

[7] Para mayor información visite la siguiente página web: http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.graphics.surfaceformat.aspx

[8] Para mayor información visite la siguiente página web: http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.graphics.texturefilter.aspx

[9] Texto tomado de: http://msdn.microsoft.com/es-es/library/microsoft.windowsmobile.directx.direct3d.texturefilter(VS.85).aspx

	minificación.
Linear	*Filtro de interpolación bilineal utilizado como un filtro de ampliación o reducción de la textura. Se utiliza un promedio ponderado de un área 2x2 de téxels (elementos de textura de un píxel) alrededor del píxel deseado. El filtro de la textura utilizado entre los niveles de mipmap es una interpolación de mipmaps trilineal en la que la impresora de trama realiza la interpolación lineal del color del píxel, utilizando los téxels de las dos texturas de mipmap más cercanas.*
None	*Los mipmaps están deshabilitados. En su lugar, la impresora de trama utiliza el filtro de ampliación.*
Point	*Filtro de punto utilizado como un filtro de ampliación o reducción de la textura. Se utiliza el téxel con las coordenadas más próximas al valor de píxel deseado. El filtro de textura utilizado entre los niveles de mipmap se basa en el punto más cercano; es decir, la impresora de trama utiliza el color del téxel de la textura de mipmap más cercana.*[10]
PyramidalQuad	Usa un filtro paso banda con máscara de 4x4 para la magnificación y minificación de la textura.

4.3 Mipmaps

Los mipmaps son una secuencia de texturas que parten de una original, cada textura es la mitad de su tamaño de su antecesora, excepto la primera. Esto ayuda a mejorar la imagen final, quitando ese parpadeo cuando hay cambios bruscos, u otros problemas visuales.

4.4 Plano con textura

Ahora que ya se ha explicado un poco sobre las texturas en el mundo de la computación gráfica, es momento de revisar un ejemplo que maneje una textura sobre una geometría simple. Dos triángulos adyacentes formaran un cuadrado con una textura.

Comience por abrir Visual Studio 2008 y cree un nuevo proyecto para XNA, seleccione la plantilla **Windows Game (3.1)**.

Tomando parte del código del ejemplo anterior, solo se explicaran las nuevas líneas de código.

Ahora agregue las siguientes variables de instancia, dentro de la definición de la clase pero fuera de todo método, en la clase **Game1** (a menos que le haya cambiado el nombre)que hereda de la clase Game.

Código 4-1

```
1.      VertexDeclaration vertexDeclaration;
2.      VertexPositionNormalTexture[] vertices ={
3.          new VertexPositionNormalTexture(new Vector3(-1.0F, -1.0F, -1.0F),
4.              new Vector3(0.0F,0.0F,-1.0F), new Vector2(0.0F,1.0F)),
5.          new VertexPositionNormalTexture(new Vector3(-1.0F,1.0F,-1.0F),
6.              new Vector3(0.0F,0.0F,-1.0F), new Vector2(0.0F,0.0F)),
7.          new VertexPositionNormalTexture(new Vector3(1.0F,1.0F,-1.0F),
8.              new Vector3(0.0F,0.0F,-1.0F), new Vector2(1.0F,0.0F)),
9.          new VertexPositionNormalTexture(new Vector3(1.0F,-1.0F,-1.0F),
10.             new Vector3(0.0F,0.0F,-1.0F), new Vector2(1.0F,1.0F))
11.     };
12.     Int32[] indices = { 0, 1, 2, 0, 2, 3 };
```

[10] Texto tomado de: http://msdn.microsoft.com/es-es/library/microsoft.windowsmobile.directx.direct3d.texturefilter(VS.85).aspx

```
13.     Texture2D textura;
14.     BasicEffect efecto;
15.     Vector3 posicion = new Vector3(0.0F, 0.0F, 2.0F);
16.     Vector3 vista = Vector3.Zero;
17.     Vector3 traslacion = Vector3.Zero;
18.     Single escala = 1.0F;
19.     Single rotacionX = 0.0F;
20.     Single rotacionY = 0.0F;
21.     Single rotacionZ = 0.0F;
```

El tipo de vértice en esta ocasión es **VertexPositionNormalTexture**, línea 2, debe contener dos **Vector3** que contendrán las coordenadas espaciales y la normal respectivamente; y un **Vector2** para las coordenadas de textura.

El objeto **textura**, línea 13, representa una malla 2D de téxeles y cada téxel es direccionado por un vector con coordenadas **u** y **v**.

En el método **Initialize** de la clase **Game1**, agregue las siguientes líneas de código para inicializar el efecto y el dispositivo gráfico.

```
vertexDeclaration = new VertexDeclaration(GraphicsDevice,
VertexPositionNormalTexture.VertexElements);
efecto = new BasicEffect(GraphicsDevice, null);
efecto.TextureEnabled = true;   // se habilita la textura del efecto básico

// modos de sampleo
GraphicsDevice.SamplerStates[0].MagFilter = TextureFilter.Linear;
GraphicsDevice.SamplerStates[0].MinFilter = TextureFilter.Linear;
GraphicsDevice.SamplerStates[0].MipFilter = TextureFilter.Linear;
```

En este caso necesitamos habilitar la propiedad **Textura** del **EffectBasic** que nos proporciona XNA; como puede ver no se ha habilitado el color de los vértices, pues no tiene sentido al carecer éstos de dicha información, pero haga la prueba para que vea que es lo que pasa.

Se inicializa el modo de sampleo del dispositivo gráfico con **SamplerStates**, que recupera una colección de objetos **SamplerState** del **GraphicsDevice**. Y se le asignan a cada una de las propiedades **MagFilter**, **Minfilter** y **MipFilter**; el filtro de tipo **Linear**.

Antes de continuar con el código es necesario añadir la imagen en el proyecto, así que para agregar un nuevo elemento haga clic secundario sobre **Content**, en el **Explorador de Soluciones** de Visual Studio y seleccione **Agregar**, véase Ilustración 4-2.

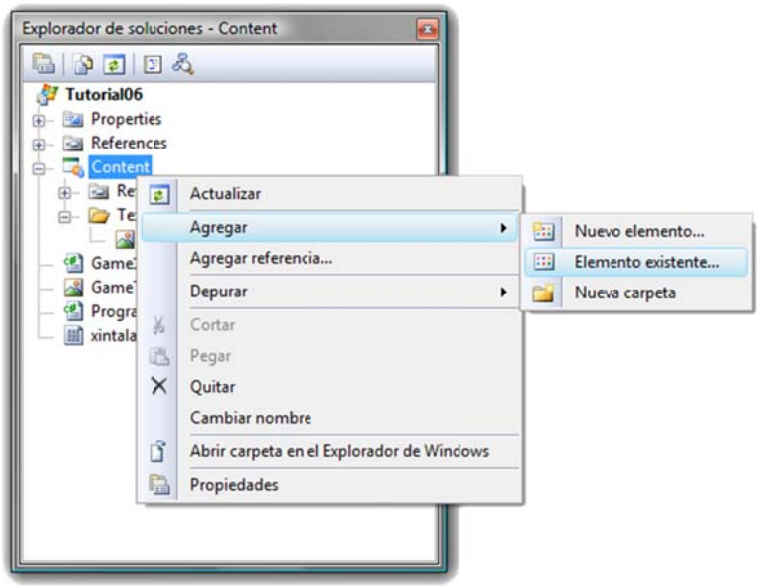

Ilustración 4-2 Agregar\ Elemento

En este caso se agregó una nueva carpeta, llamada **Texturas**, para almacenar las imágenes, y es que es mejor tener las cosas ordenadas. Luego haga lo mismo con la carpeta creada en **Content**, o sea la nueva carpeta creada con el nombre **Texturas**, y agregue **Elemento existente**, véase Ilustración 4-2.

Siguiendo con el código, dentro del método **LoadContent** escriba las siguientes líneas para cargar la textura y generar los **MipMaps**.

```
// Lectura de la textura
textura = Content.Load<Texture2D>(@"Texturas\Xin");
textura.GenerateMipMaps(TextureFilter.Linear);  // creación de los MipMaps
```

Hay dos maneras de cargar una textura desde un archivo, una es por medio el método genérico y estático **ContentManager.Load**[11]. Éste carga un recurso para ser procesado por el **Content Pipeline**. Y el otro es por medio del método **Texture2D.TextureFromFile**.

Lo recomendable usar es el **Content.Load**, el cual recibe un **String** del **Asset Name**. El **Asset Name** es el nombre que se usará como referencia en tiempo de ejecución, por lo que no es necesario colocar la extensión del archivo.

Para saber el **Asset Name**, seleccione el archivo en el **Explorador de soluciones** de Visual Studio, y verá en la ventana **Propiedades** el **String** del **Asset Name**, véase Ilustración 4-3.

[11] Para mayor información revise la siguiente página web: http://msdn.microsoft.com/en-us/library/bb197848.aspx

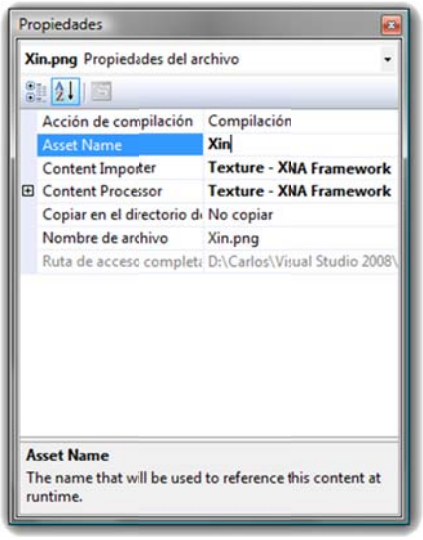

Ilustración 4-3 Asset Name

El **string** que debe recibir el método **LoadContent**, debe incluir los nombres de las carpetas anidadas en directorio asociado en el **ContentManager**, en donde se deposita el archivo creado por un **Digital Content Creation** (DCC)[12]. El **ContentManager** es una clase que carga el contenido del **Content Pipeline** en tiempo de ejecución[13].

Enseguida se generan los mipmaps de la textura con la constante **TextureFilter.Linear** con el que se filtra cada nivel del mipmap, por medio del método **Texture.GenerateMipMaps**.

```
public void GenerateMipMaps(TextureFilter filterType)
```

En el método **UnloadContent** se añade la siguiente línea para liberar el recurso ocupado, que en este caso es la textura.

```
protected override void UnloadContent()
{
    textura.Dispose();
}
```

Para poder apreciar que en realidad se ha colocado la textura sobre una geometría se toman las mismas líneas de transformaciones del ejemplo anterior y se escriben dentro del método **Update**.

```
if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
    this.Exit();
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.Left))
    posicion.X -= 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.Right))
    posicion.X += 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.Up))
    posicion.Y += 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.Down))
    posicion.Y -= 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.PageUp))
    posicion.Z += 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.PageDown))
```

[12] La DCC es una herramienta de creación de contenido digital.

[13] Para mayor información revise la siguientes páginas web: http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.content.contentmanager.aspx
http://msdn.microsoft.com/es-es/library/bb447756.aspx

```
    posicion.Z -= 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.D))
    traslacion.X -= 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.A))
    traslacion.X += 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.W))
    traslacion.Y += 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.S))
    traslacion.Y -= 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.Z))
    traslacion.Z += 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.X))
    traslacion.Z -= 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.E))
    escala += 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.R))
{
    escala -= 0.1F;
    if (escala < 0.1F)
        escala = 0.1F;
}
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.J))
    rotacionY -= 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.L))
    rotacionY += 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.I))
    rotacionX += 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.K))
    rotacionX -= 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.U))
    rotacionZ += 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.O))
    rotacionZ -= 0.1F;
if (Keyboard.GetState(PlayerIndex.One).IsKeyDown(Keys.Escape))
    this.Exit();
```

La asignación de la textura a la propiedad **Texture** de la instancia **efecto**, línea 13, Código 4-2, es la textura que será aplicada hacia la geometría. El tipo de dato es un **Texture2D**, como lo indica la sintaxis siguiente.

```
public Texture2D Texture { get; set; }
```

El método **EnableDefaultLighting**, línea 14, habilita la iluminación por omisión de este efecto. La propiedad **PreferPerPixelLighting**, línea 15, obtiene o establece que la iluminación por píxel puede ser soportada por el dispositivo gráfico, o sea la GPU. Así que si no tiene una GPU con soporte mínimo para Pixel Shader 2.0 no escriba esta línea de código.

Código 4-2

```
1.      protected override void Draw(GameTime gameTime)
2.      {
3.          GraphicsDevice.Clear(color);
4.
5.          Single aspecto = GraphicsDevice.Viewport.AspectRatio;
6.          efecto.World = Matrix.Identity * Matrix.CreateScale(escala) *
7.              Matrix.CreateRotationX(rotacionX) *
8.              Matrix.CreateRotationY(rotacionY) * Matrix.CreateRotationZ(rotacionZ) *
9.              Matrix.CreateTranslation(traslacion);
10.         efecto.View = Matrix.CreateLookAt(posicion, vista, arriba);
11.         efecto.Projection = Matrix.CreatePerspectiveFieldOfView(1, aspecto, 1, 1000);
12.
13.         efecto.Texture = textura;
14.         efecto.EnableDefaultLighting(); // Iluminación por default del efecto básico
15.         efecto.PreferPerPixelLighting = true;   // iluminación por pixel
16.
17.         GraphicsDevice.RenderState.FillMode = FillMode.Solid;
18.         GraphicsDevice.VertexDeclaration = vertexDeclaration;
```

```
19.         GraphicsDevice.RenderState.CullMode = CullMode.None;
20.
21.         // comienza el trazado de la geometría
22.         efecto.Begin();
23.         efecto.CurrentTechnique.Passes[0].Begin();
24.         GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionNormalTexture>(
25.             PrimitiveType.TriangleList,
26.             vertices, 0, vertices.Length, indices, 0, 2);
27.         efecto.CurrentTechnique.Passes[0].End();
28.         efecto.End();
29.
30.         base.Draw(gameTime);
31.     }
```

Ahora inicie la depuración oprimiendo **F5** para poder ver el plano con la textura pegada a ésta. Si todo salió bien, podrá ver una imagen similar a la Ilustración 4-4, si tiene algún error de compilación corrija y vuelva a intentar.

Ilustración 4-4 Plano con textura

Sobre todo mueva el modelo con el teclado para que pueda apreciar los cambios, como se muestra en la Ilustración 4-5.

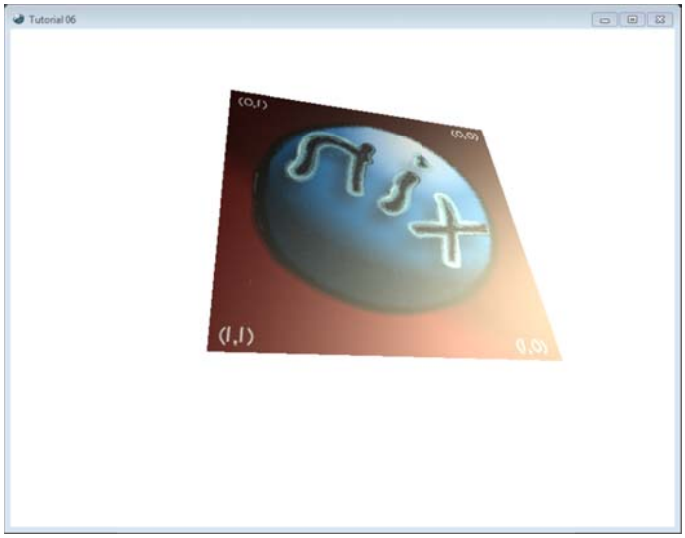

Ilustración 4-5 Transformaciones sobre el plano texturizado

4.5 Modos de direccionamiento

Hasta ahora las coordenadas de textura asignadas tienen valores de cero o uno, para cubrir toda la geometría con la imagen, pero ¿qué pasaría si estos valores fueran mayores a uno? Bueno la manera en que la textura ahora envolverá a la geometría dependerá del tipo de direccionamiento que se le asigne a las coordenadas **u** y **v**.

Para poder ver los modos de direccionamiento se cambiaran las coordenadas de textura por las siguientes.

```
VertexPositionNormalTexture[] vertices ={
    new VertexPositionNormalTexture(new Vector3(-1.0F, -1.0F, -1.0F),
         new Vector3(0.0F,0.0F,-1.0F),
         new Vector2(0.0F,5.0F)),
    new VertexPositionNormalTexture(new Vector3(-1.0F,1.0F,-1.0F),
        new Vector3(0.0F,0.0F,-1.0F),
        new Vector2(0.0F,0.0F)),
    new VertexPositionNormalTexture(new Vector3(1.0F,1.0F,-1.0F),
        new Vector3(0.0F,0.0F,-1.0F),
        new Vector2(5.0F,0.0F)),
    new VertexPositionNormalTexture(new Vector3(1.0F,-1.0F,-1.0F),
        new Vector3(0.0F,0.0F,-1.0F),
        new Vector2(5.0F,5.0F))
};
```

El modo de direccionamiento **Wrap** será el primero en revisar, éste sirve para repetir la textura sobre la geometría de manera que cubra toda.

En el método **Initialize** agregue las siguientes líneas de código, después de la asignación del filtro lineal.

```
GraphicsDevice.SamplerStates[0].AddressU = TextureAddressMode.Wrap;
GraphicsDevice.SamplerStates[0].AddressV = TextureAddressMode.Wrap;
```

En este caso se deja el mismo modo de direccionamiento para las coordenadas **u** y **v**, pero no significa que así sea.

AddressU y **AddressV** son propiedades que obtienen o establecen el direccionamiento sobre las coordenadas **u** y **v**, respectivamente. **TextureAddressMode** es una numeración que define el modo de direccionamiento de la textura.

Ejecute el programa y corrija cualquier error de compilación que sucediera, si tiene éxito logrará algo similar a la Ilustración 4-6.

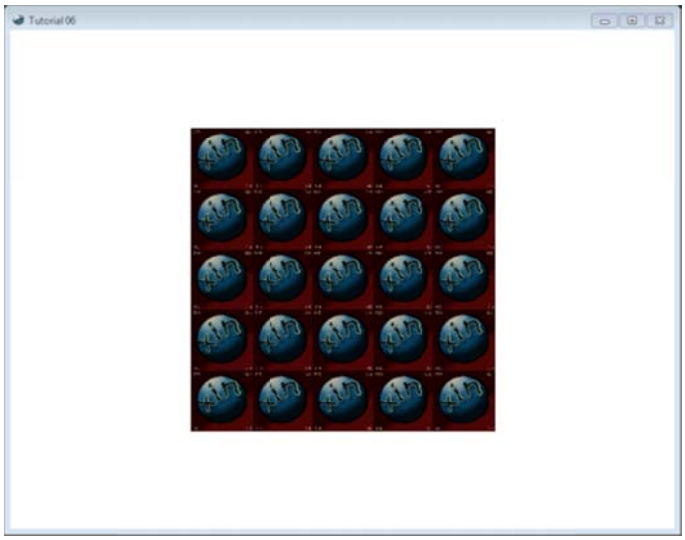

Ilustración 4-6 Wrap

El siguiente modo de direccionamiento corresponde a **Clamp**, las coordenadas de textura fuera del rango [0.0, 1.0] se definen con el color de la última columna y renglón de la textura para rellenar. Cambie la numeración de **TextureAddressMode.Wrap** a **TextureAddressMode.Clamp**.

```
GraphicsDevice.SamplerStates[0].AddressU = TextureAddressMode.Clamp;
GraphicsDevice.SamplerStates[0].AddressV = TextureAddressMode.Clamp;
```

Inicie el programa y corrija cualquier error de compilación que suceda, si tiene éxito podrá ver algo similar a la Ilustración 4-7.

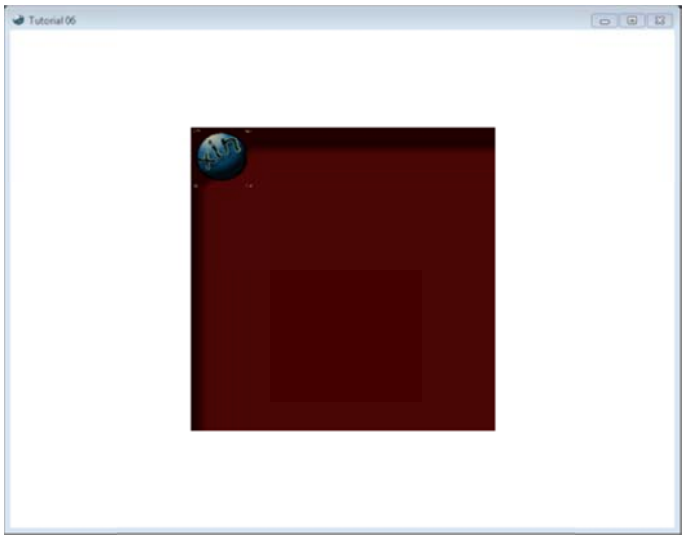

Ilustración 4-7 Clamp

El direccionamiento **Mirror** es similar al **Wrap**, sin embargo, en el momento de cubrir la geometría la siguiente imagen estará invertida, como si se estuviera viendo en un espejo; esto sucede hasta que se termine de cubrir la geometría.

Cambie la numeración **TextureAddressMode.Clamp** por **TextureAddressMode.Mirror** en las propiedades **AddresU** y **AddresV**.

```
GraphicsDevice.SamplerStates[0].AddressU = TextureAddressMode.Mirror;
GraphicsDevice.SamplerStates[0].AddressV = TextureAddressMode.Mirror;
```

Oprima **F5** para comenzar con la depuración y así correr el programa, si ocurre cualquier error de compilación o de tiempo de ejecución soluciónelo y vuelva a intentar. Si lo ha conseguido verá una imagen como la que se muestra en la Ilustración 4-8.

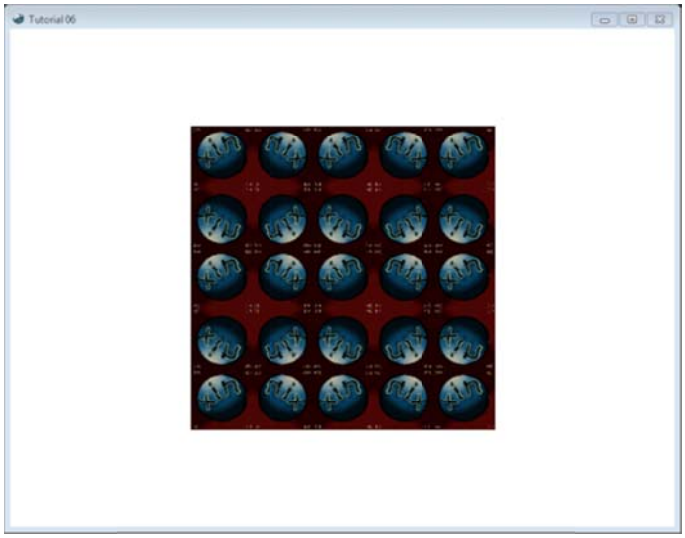

Ilustración 4-8 Mirror

Modo de direccionamiento **Border**, aquí se coloca una vez la textura sobre la geometría y el resto, si es que tiene, será rellenado por un color.

Cambie la numeración del modo de direccionamiento de las coordenadas **u** y **v** por **Border**, y agregue el color al borde. **BorderColor** es una propiedad que obtiene o establece el color del borde.

```
GraphicsDevice.SamplerStates[0].AddressU = TextureAddressMode.Border;
GraphicsDevice.SamplerStates[0].AddressV = TextureAddressMode.Border;
GraphicsDevice.SamplerStates[0].BorderColor = Color.LightBlue;
```

Inicie con la depuración del programa con **F5**, y corrija cualquier error para poder ver algo similar a la Ilustración 4-9.

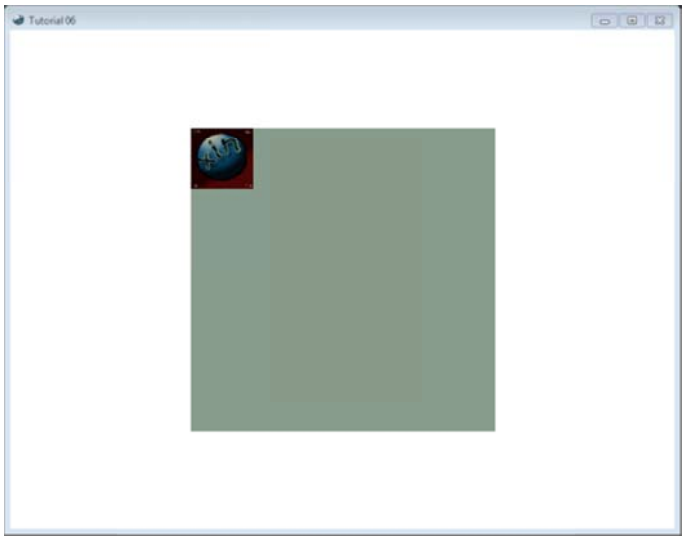

Ilustración 4-9 Border

Regularmente los modos de direccionamiento son iguales para **u** y **v**, sin embargo, pueden crearse combinaciones entre las distintas propiedades.