Parte II Manual

1 Vertex Buffer

1.1 VertexBuffer

Los modelos tridimensionales que se pueden ver en los videojuegos están conformados por un conjunto de planos triangulares en el espacio tridimensional. Cada plano está constituido por una terna de puntos y una triada de aristas. Los puntos, alejándonos del término matemático, almacenan información sobre su posición, color, textura, normal, tangente y binormal. Los puntos en el espacio, por lo menos deben contener un conjunto de tres coordenadas (x, y, z). Dado la posible información que representan estos puntos, se les ha denominado vértices.

El vertex buffer es una región de memoria, en la tarjeta gráfica, para almacenar vértices. Para establecer dichos datos en el búfer de vértices, XNA ofrece las clases **VertexBuffer** y **DynamicVertexBuffer**, esta última hereda de la primera y VertexBuffer no se recomienda para el uso sobre el Xbox360[3].

En el ejemplo siguiente, se muestra el uso del VertexBuffer para dibujar un triángulo. Comience por abrir Visual Studio y cree un nuevo proyecto; para ello, vaya al menú principal y seleccione **Archivo\Nuevo\Proyecto**. Enseguida se abrirá una ventana de diálogo para seleccionar el tipo de proyecto y plantilla instalados. En tipos de proyecto, seleccione **XNA Game Studio 3.1** y en la parte de plantillas tome **Windows Game (3.1)**, véase Ilustración 1-1.

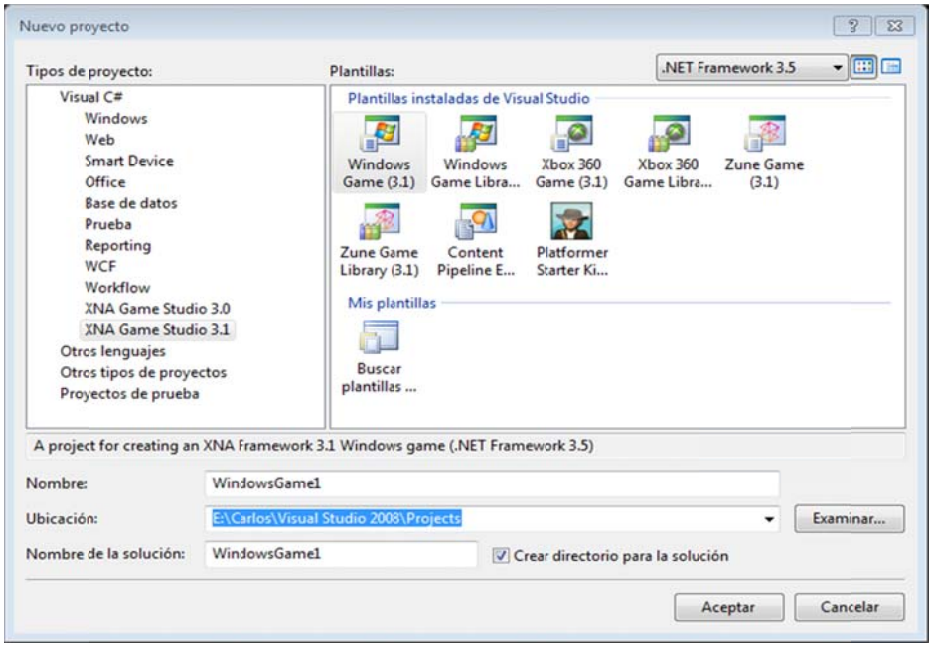

Ilustración 1-1 Ventana de diálogo: Nuevo proyecto

Al crear su nueva aplicación, podrá visualizar que automáticamente se crea todo lo necesario para comenzar, es más, puede oprimir F5 para ejecutarla y ver una ventana azul, véase Ilustración 1-2.

[3] Para mayor información acerca del vertex buffer consulte: http://msdn.microsoft.com/en-us/library/bb198836.aspx

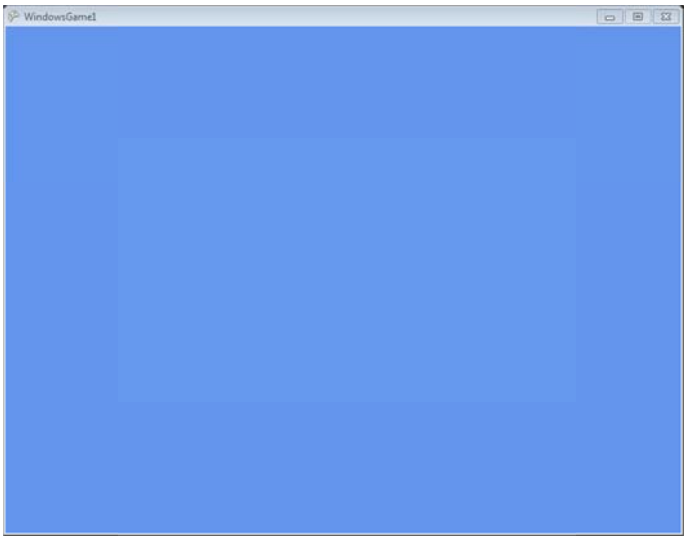

Ilustración 1-2 Programa predefinido

Claro está, que esto hace más rápido el desarrollo de la aplicación, sin embargo, hay que explicar unas cosas antes de continuar, y sólo serán las necesarias para ver el triángulo dibujado en la ventana.

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

Estas son las directivas que se necesitan para crear el triángulo y que tiene que colocar al inicio del archivo Game1.cs creado por Visual Studio.

- **System** es el espacio de nombre que contiene las clases fundamentales que definen los datos, los eventos, las interfaces, etcétera.[4]
- **Microsoft.XNA.Framework** es el espacio de nombre que contiene las clases necesarias para crear juegos.
- **Microsoft.XNA.Framework.Graphics** es el espacio de nombre que contiene los métodos de la API de bajo nivel.

```
namespace TutorialX01
{
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
    }
}
```

Visual Studio crea el **namespace**, cuyo nombre corresponde al del proyecto; la clase **Game1**, que hereda de la clase **Game** y la variable de instancia **GraphicsDeviceManager** que maneja la configuración y administración del dispositivo gráfico, que es la tarjeta de video de la computadora. La clase **Game** provee de la inicialización básica del dispositivo gráfico, de la lógica del juego y el código del render.

Para el ejemplo se añaden las siguientes líneas, después de la declaración del dispositivo gráfico **GraphicsDeviceManager**.

```
VertexDeclaration vertexDeclaration;
```

[4] Para mayor información acerca de System visite la siguiente dirección: http://msdn.microsoft.com/es-es/library/system.aspx

```
BasicEffect effect;
VertexBuffer vertexBuffer;
```

VertexDeclaration es la clase que representa la declaración de un vértice, debido a que existen diferentes tipos de vértices definidos en XNA, estos pueden ser:

- **VertexPositionColor**. Esctructura personalizada que contiene posición y color.
- **VertexPositionColorTexture**. Estructura personalizada que contiene posición, color y textura.
- **VertexPositionNormalTexture**. Estructura personalizada que contiene posición, normal y textura.
- **VertexPositionTexture**. Estructura personalizada que contiene posición y textura.

En este ejemplo se utiliza **VertexPositionColor**, más adelante se mostraran los demás tipos de vértices. En DirectX se definía la estructura del tipo de vértice, sin embargo, XNA da estos cuatro tipos de estructuras como parte ésta, sin más que llamarlas como cualquier tipo de dato.

BasicEffect representa un shader versión 1.1, dando soporte para el color de los vértices, textura e iluminación. En XNA es necesario utilizar un tipo de shader para mostrar en pantalla el dibujo, a diferencia de DirectX u OpenGL.

VertexBuffer es la clase que representa el buffer de vértices para manipular los recursos.

Ahora se instancia los vértices que crearan el triángulo, por lo que se usa un arreglo de **VertexPositionColor**. El constructor sería el siguiente:

```
public VertexPositionColor (Vector3 position, Color color)
```

Tabla 1-1

Propiedad	Descripción
position	Es un Vector3, por lo que representa la posición del vértice en coordeanas x, y ,z.
color	Es el color del vértice representado por los colores R,G,B.

```
private readonly VertexPositionColor[] vertices =
{
    new VertexPositionColor(new Vector3(0.0F, 1.0F, 0.0F), Color.White),
    new VertexPositionColor(new Vector3(1.0F, -1.0F, 0.0F), Color. White),
    new VertexPositionColor(new Vector3(-1.0F, -1.0F, 0.0F), Color. White)
};
```

En el constructor, de la clase Game1, se inicializa el dispositivo gráfico y algunas propiedades de la ventana en donde se mostrará todo el mundo que se creé, en la Tabla 1-2 se muestran algunas de estas propiedades[5].

```
public Game1()
{
    graphics = new GraphicsDeviceManager(this);
    this.IsMouseVisible = true;
    this.Window.Title = "xintalalai Tutorial 00";
    this.Window.AllowUserResizing = true;
```

[5] Para mayor información de los miebros de la clase Game consulte la siguiente dirección: http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.game_members.aspx

```
}
```

Tabla 1-2

Propiedad	Descripción
IsMouseVisible	Muestra u oculta el puntero del mouse sobre la venta, por default está oculto.
Window.Title	Es el título de la ventana, por default el título es el nombre que se le dió a la solución en el momento de crearla.
Window.AllowUserResizing	Permite máximizar o redimensionar el tamaño de la venta.

En el método **Initialize** es donde se inicializa toda lógica o cualquier recurso no gráfico. Por default dentro del método se tiene la inicialización base de la clase **Game**, así que hay que agregar unas cuantas líneas para crear la declaración del vértice, el efecto así como el búfer de vértices.

```
protected override void Initialize()
{
    base.Initialize();
}
```

La declaración de **VertexDeclaration** necesita de dos parámetros; el primero es el dispositivo gráfico en el que se asocian los vértices, el segundo es un arreglo de elementos de vértices, que ya contiene la estructura de **VertexPositionColor**.

```
protected override void Initialize()
{
    vertexDeclaration = new VertexDeclaration(GraphicsDevice,
        VertexPositionColor.VertexElements);
    vertexBuffer = new VertexBuffer(GraphicsDevice, 3 * VertexPositionColor.SizeInBytes,
        BufferUsage.WriteOnly);

    effect = new BasicEffect(GraphicsDevice, null);
    effect.VertexColorEnabled = true;

    vertexBuffer.SetData<VertexPositionColor>(vertices, 0, vertices.Length);

    base.Initialize();
}
```

El constructor del **VertexBuffer** está sobrecargado, el que se tomó en cuenta es el que específica el tamaño y su uso.

```
VertexBuffer (GraphicsDevice, Int32, BufferUsage)
```

Tabla 1-3

Parámetros	Descripción
graphicsDevice	Es el dispositivo gráfico asociado con el vertex buffer.
sizeInBytes	Es el número de bytes alojados en el vertex buffer.

Usage	Es la opción que identifica el comportamiento del vertex buffer.

Para inicializar el efecto básico que proporciona XNA se utiliza:

```
public BasicEffect (GraphicsDevice device, EffectPool effectPool)
```

Tabla 1-4

Parámetros	Descripción
device	Es el dispositivo gráfico que creará el efecto.
effectPool	Especifica un conjunto de recursos para compartir entre los efectos.

BasicEffect.VertexColorEnabled es una propiedad que habilita el uso de vertices con colores.

```
public void SetData<T> (T[] data, int startIndex, int elementCount)
```

El método **SetData** de **VertexBuffer** adquiere los datos a copiar en el búfer de vértices, donde **T** es un tipo de dato en el búfer.

Tabla 1-5

Parámetros	Descripción
data	Es un arreglo que será copiado al vertex buffer.
starIndex	Indica el índice a partir del cual se copiarán los datos.
elementCount	Es el número de elementos máximos a copiar.

Siguiendo con el código generado automáticamente, por Visual Studio, tenemos los siguientes métodos:

```
protected override void LoadContent()
{
}

protected override void UnloadContent()
{
}

protected override void Update(GameTime gameTime)
{
base.Update(gameTime);
}
```

- **LoadContent** es llamado cuando los recursos gráficos necesitan ser cargados.
- **UnloadContent** es llamado cuando los recursos gráficos necesitan ser liberados.
- **Update** actualiza el estado de la sesión de multiplayer, como actualizar o supervisar el estado de los dispositivos de entrada, como el gamepad.

Como parte final del archivo **Game1.cs** se tiene el método que dibujará las geometrías, y otras cosas más en pantalla, que por el momento será un triángulo.

```
protected override void Draw(GameTime gameTime)
{
```

```
    GraphicsDevice.Clear(Color.Black);

    Single apecto = GraphicsDevice.Viewport.AspectRatio;
    effect.World = Matrix.Identity;
    effect.View = Matrix.CreateLookAt(new Vector3(0, 0, 5),
        Vector3.Zero,
        Vector3.Up);
    effect.Projection = Matrix.CreatePerspectiveFieldOfView(1, apecto, 1, 10);

    GraphicsDevice.RenderState.FillMode = FillMode.WireFrame;
    GraphicsDevice.VertexDeclaration = vertexDeclaration;
    GraphicsDevice.Vertices[0].SetSource(vertexBuffer, 0, VertexPositionColor.SizeInBytes);

    effect.Begin();
        effect.CurrentTechnique.Passes[0].Begin();
        GraphicsDevice.DrawPrimitives(PrimitiveType.TriangleList, 0, 1);
        effect.CurrentTechnique.Passes[0].End();
    effect.End();

    base.Draw(gameTime);
}
```

La primera línea limpia el **viewport** especificando un color, en este caso negro. Se asignan valores a la matriz de mundo del efecto, en este caso con la matriz identidad, esta matriz sirve para hacer cabios de posición del modelo. Así también se le asigna un valor a la matriz de vista, que sirve para cambiar la posición y dirección de la cámara, por medio del método estático **CreateLookAt**. Por el momento no se explicaran a fondo estos términos, pues se verá en los siguientes ejemplos, por ahora sólo escríbalos.

Projection: es la matriz para cambiar la imagen 3D a 2D que será dibujada en la pantalla de la computadora; hay diferentes formas de proyecciones que se verán posteriormente.

Fillmode: es un numerador que especifica cómo se rellena el triángulo; existen tres modos de relleno de las geometrías, los cuales pueden ser punto, malla o sólido.

En el modo de relleno **Point** se dibujan los vértices de la geometría sin conectarlos entre sí, véase Ilustración 1-3.

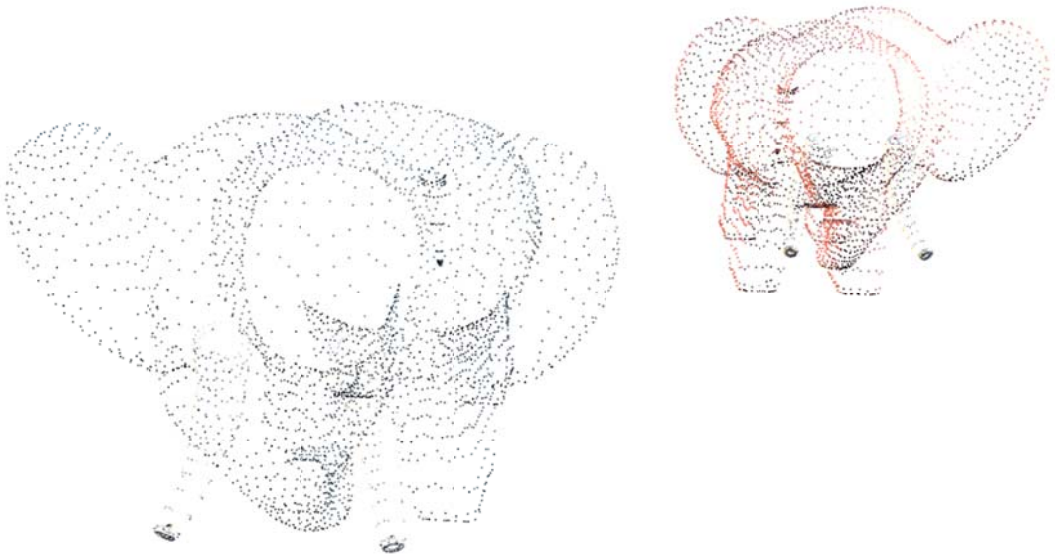

Ilustración 1-3 Fillmode.Point

En el modo de relleno **WireFrame**, sólo se dibujan los lados que conectan los vértices de la geometría, véase Ilustración 1-4.

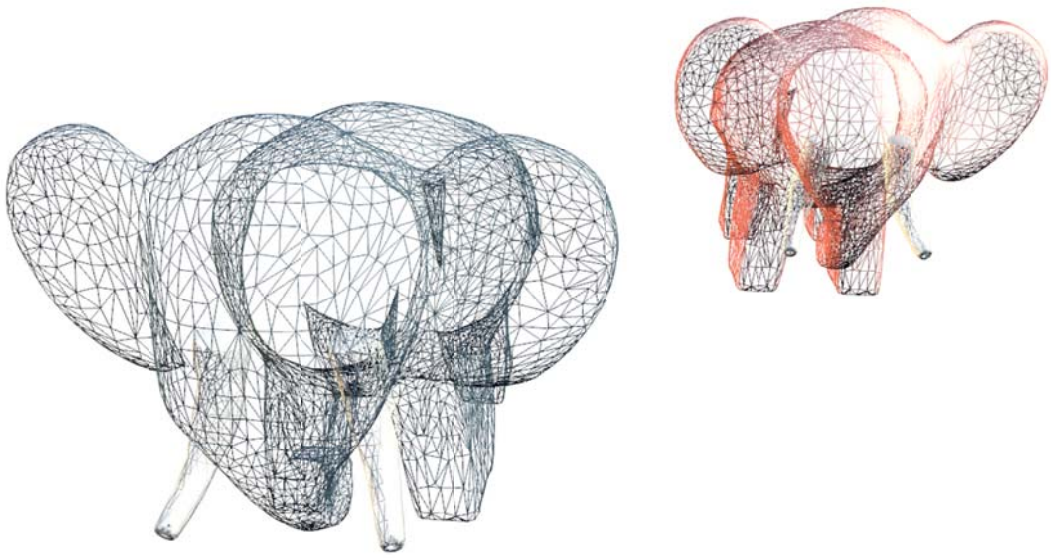

Ilustración 1-4 Fillmode.WireFrame

En el modo de relleno **Solid**, se dibujan los lados que conectan los vértices y los rellena, como en la Ilustración 1-5.

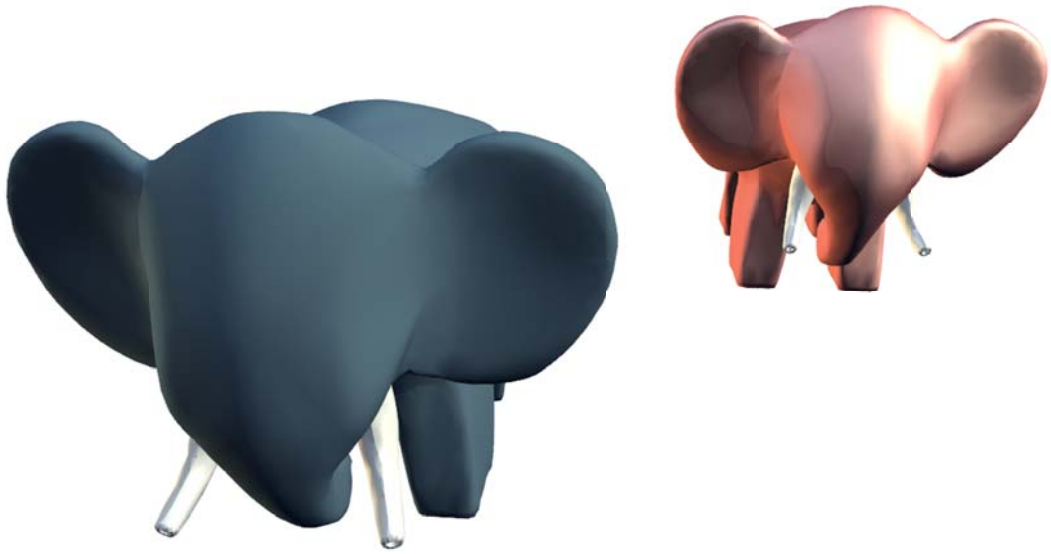

Ilustración 1-5 Fillmode.Solid

A **GraphicsDevice.VertexDeclaration** se le asigna la declaración del vértice al dispositivo gráfico. A **GraphicsDevice.Vertices[0].SetSource** se le asigna el contenido del búfer de vértices.

```
public void SetSource (VertexBuffer vb, int offsetInBytes,int
vertexStride)
```

Tabla 1-6

Parámetro	Descripción

vb	El verttex buffer de donde se tomaran los datos.
offsetInBytes	Es el byte a partir del cual serán copiados los datos.
vertexStride	Es el tamaño en bytes de los elementos en el vertex buffer.

Para dibujar la geometría se utilizan los métodos del efecto, así que se envuelve entre un **Begin**, método que comienza el efecto; y un **End**, método que finaliza el efecto, al método **GraphicsDevice.DrawPrimitives.**

También se debe envolver entre las técnicas que contiene el efecto y sus pasadas, en este caso solo tienen una y también comienza con un **Begin** y termina con un **End**.

```
public void DrawPrimitives(PrimitiveType primitiveType, int startVertex,
int primitiveCount)
```

DrawPrimitives dibuja una secuencia no indexada de la geometría, especificando el tipo de primitiva. En el siguiente capítulo se verán los distintos tipos de primitivas que ofrece XNA.

Tabla 1-7

Parámetro	Descripción
primitiveType	Describe el tipo de primitiva a dibujar.
startVertex	Indica el primer vértice a partir del cual comenzará a dibujar.
primitiveCount	Es el número de primitivas a dibujar.

Para concluir este capítulo, en el archivo **Program.cs**, que crea Visual Studio, se encuentra el método **Main**, el cual corresponde al punto de inicio del programa.

```
static void Main(string[] args)
{
    using (Game1 game = new Game1())
    {
        game.Run();
    }
}
```

El método **Run** de la clase **Game** es para inicializar el juego, para mantener en un bucle el dibujo y para comenzar el procesamiento de eventos de la aplicación.

Genere el proyecto en el menú **Generar** y corrija cualquier excepción de compilación que haya ocurrido. Si todo salió bien corra la aplicación con **F5** o vaya al menú **Depurar** para iniciar el programa, y verá un triángulo como el que se muestra en la Ilustración 1-6.

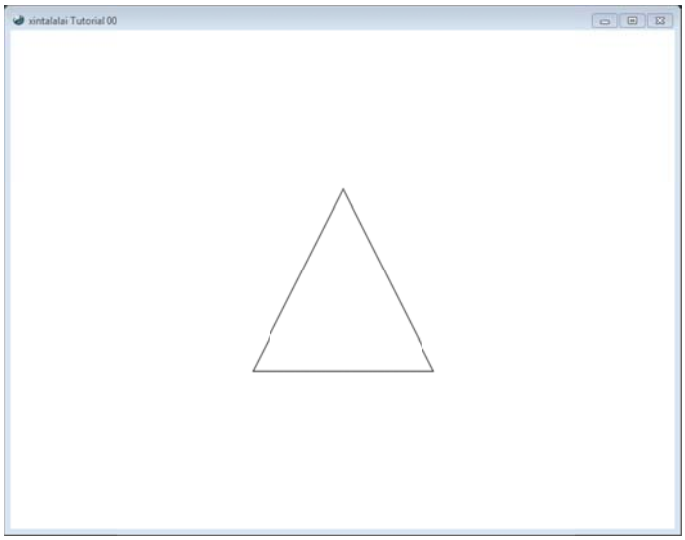

Ilustración 1-6 Triángulo

En el Código 1-1 se muestra el enlistado completo que debería tener **Game1.cs**.

Código 1-1

```
1.      using System;
2.      using Microsoft.Xna.Framework;
3.      using Microsoft.Xna.Framework.Graphics;
4.
5.      namespace TutorialX01
6.      {
7.          public class Game1 : Microsoft.Xna.Framework.Game
8.          {
9.              GraphicsDeviceManager graphics;
10.             VertexDeclaration vertexDeclaration;
11.             BasicEffect effect;
12.             VertexBuffer vertexBuffer;
13.
14.             private readonly VertexPositionColor[] vertices =
15.             {
16.                 new VertexPositionColor(new Vector3(0.0F, 1.0F, 0.0F), Color.White),
17.                 new VertexPositionColor(new Vector3(1.0F, -1.0F, 0.0F), Color.White),
18.                 new VertexPositionColor(new Vector3(-1.0F, -1.0F, 0.0F), Color.White)
19.             };
20.
21.
22.             public Game1()
23.             {
24.                 graphics = new GraphicsDeviceManager(this);
25.                 this.IsMouseVisible = true;
26.                 this.Window.Title = "xintalalai Tutorial 00";
27.                 this.Window.AllowUserResizing = true;
28.                 Content.RootDirectory = "Content";
29.             }
30.
31.             protected override void Initialize()
32.             {
33.                 vertexDeclaration = new VertexDeclaration(GraphicsDevice,
34.                     VertexPositionColor.VertexElements);
35.                 vertexBuffer = new VertexBuffer(GraphicsDevice, 3 *
36.                     VertexPositionColor.SizeInBytes, BufferUsage.WriteOnly);
37.
38.                 effect = new BasicEffect(GraphicsDevice, null);
39.                 effect.VertexColorEnabled = true;
40.
41.                 vertexBuffer.SetData<VertexPositionColor>(vertices, 0, vertices.Length);
42.
```

```
43.                 base.Initialize();
44.             }
45.
46.             protected override void LoadContent()
47.             {
48.             }
49.
50.             protected override void UnloadContent()
51.             {
52.             }
53.
54.             protected override void Update(GameTime gameTime)
55.             {
56.                 base.Update(gameTime);
57.             }
58.
59.             protected override void Draw(GameTime gameTime)
60.             {
61.                 GraphicsDevice.Clear(Color.Black);
62.
63.                 Single apecto = GraphicsDevice.Viewport.AspectRatio;
64.                 effect.World = Matrix.Identity;
65.                 effect.View = Matrix.CreateLookAt(new Vector3(0, 0, 5),
66.                     Vector3.Zero,
67.                     Vector3.Up);
68.                 effect.Projection = Matrix.CreatePerspectiveFieldOfView(1,
69.                     apecto, 1, 10);
70.
71.                 GraphicsDevice.RenderState.FillMode = FillMode.WireFrame;
72.                 GraphicsDevice.VertexDeclaration = vertexDeclaration;
73.                 GraphicsDevice.Vertices[0].SetSource(vertexBuffer, 0,
74.                     VertexPositionColor.SizeInBytes);
75.
76.                 effect.Begin();
77.                 effect.CurrentTechnique.Passes[0].Begin();
78.                 GraphicsDevice.DrawPrimitives(PrimitiveType.TriangleList, 0, 1);
79.                 effect.CurrentTechnique.Passes[0].End();
80.                 effect.End();
81.
82.                 base.Draw(gameTime);
83.             }
84.         }
85.     }
```

1.2 DynamicVertexBuffer

Al igual que la clase **VertexBuffer**, la clase **DynamicVertexBuffer** sirve para almacenar una lista de vértices, sin embargo, esta clase funciona para un arreglo dinámico de vértices, mientras que el otro es para un arreglo no dinámico. Así mismo lo recomiendan para el uso de aplicaciones sobre el Xbox360, en vez del **VetertexBuffer**; porque se puede sobre pasar el tamaño de la memoria de 10MB del EDRAM de la consola.

VertexBuffer.SetData no será necesario para escribir los datos en el búfer de vértices. Para lo anterior se cuentan con otros métodos de entrada de datos, pero que por ahora sólo se mostrara el adecuado para el triángulo.

```
public void DrawUserPrimitives<T> (PrimitiveType primitiveType, T[]
vertexData, int vertexOffset, int primitiveCount)
```

Tabla 1-8

Parámetro	Descripción
primitiveType	Describe el tipo de primitiva a dibujar
vertexData	Es el arreglo de vértices
vertexOffset	Es el índice del vértice a partir del cual se copiaran los datos al vertex buffer.
primitiveCount	Es el número de primitivas máximo que se dibujaran.

En la línea 65 del Código 1-2 se muestra el fácil uso de **DrawUserPrimitive**. El resto del código es el mismo que en el ejemplo anterior.

Código 1-2

```
1.      public class Game1 : Microsoft.Xna.Framework.Game
2.      {
3.          GraphicsDeviceManager graphics;
4.
5.          VertexDeclaration vertexDeclaration;
6.          BasicEffect effect;
7.          private readonly VertexPositionColor[] vertices =
8.          {
9.              new VertexPositionColor(new Vector3(0.0F, 1.0F, 0.0F), Color.Blue),
10.             new VertexPositionColor(new Vector3(1.0F, -1.0F, 0.0F), Color.Green),
11.             new VertexPositionColor(new Vector3(-1.0F, -1.0F, 0.0F), Color.Red)
12.         };
13.
14.         public Game1()
15.         {
16.             graphics = new GraphicsDeviceManager(this);
17.             Content.RootDirectory = "Content";
18.             this.IsMouseVisible = true;
19.             this.Window.Title = "xintalalai Tutorial 01";
20.             this.Window.AllowUserResizing = true;
21.         }
22.
23.         protected override void Initialize()
24.         {
25.             vertexDeclaration = new VertexDeclaration(GraphicsDevice,
26.                 VertexPositionColor.VertexElements);
27.
28.             effect = new BasicEffect(GraphicsDevice, null);
29.             effect.VertexColorEnabled = true;
30.
```

```
31.             base.Initialize();
32.         }
33.
34.         protected override void LoadContent()
35.         {
36.         }
37.
38.         protected override void UnloadContent()
39.         {
40.         }
41.
42.         protected override void Update(GameTime gameTime)
43.         {
44.             base.Update(gameTime);
45.         }
46.
47.         protected override void Draw(GameTime gameTime)
48.         {
49.             GraphicsDevice.Clear(Color.Black);
50.
51.             Single apecto = GraphicsDevice.Viewport.AspectRatio;
52.             effect.World = Matrix.Identity;
53.             effect.View = Matrix.CreateLookAt(new Vector3(0, 0, 5),
54.                 Vector3.Zero,
55.                 Vector3.Up);
56.             effect.Projection = Matrix.CreatePerspectiveFieldOfView(1, apecto, 1, 10);
57.
58.             GraphicsDevice.RenderState.FillMode = FillMode.WireFrame;
59.
60.             GraphicsDevice.VertexDeclaration = vertexDeclaration;
61.
62.             effect.Begin();
63.             effect.CurrentTechnique.Passes[0].Begin();
64.             GraphicsDevice.DrawUserPrimitives<VertexPositionColor>(
65.                 PrimitiveType.TriangleList, vertices, 0, 1);
66.             effect.CurrentTechnique.Passes[0].End();
67.             effect.End();
68.
69.             base.Draw(gameTime);
70.         }
71.     }
```