2 Primitivas

Las primitivas son elementos básicos para dibujar cualquier geometría en el espacio tridimensional. XNA ofrece un conjunto de éstas, y cada una se explicará a continuación.

Así como en el capítulo anterior, se seguirá usando el **VertexBuffer** y el **DynamicBuffer** para los ejemplos.

Abra Visual Studio y cree un nuevo proyecto de **XNA Game Studio 3.1**, seleccionando la plantilla **Windows Game (3.1)**. Escriba la declaración del búfer de vértices y las variables de instancia de los vértices, en el archivo que **Game1.cs**; recuerde que se está trabajando en los archivos que se crean por default.

```
VertexDeclaration vertexDeclaration;
BasicEffect effect;
VertexBuffer vertexBuffer;

VertexPositionColor[] vertices ={
    new VertexPositionColor(new Vector3(-2.0F, 0.0F, 2.0F), Color.White),
    new VertexPositionColor(new Vector3(-1.0F, 2.0F, 2.0F), Color.White),
    new VertexPositionColor(new Vector3(0.0F, 0.0F, 2.0F), Color.White),
    new VertexPositionColor(new Vector3(1.0F, 1.5F, 2.0F), Color.White),
    new VertexPositionColor(new Vector3(1.5F, 0.0F, 2.0F), Color.White),
    new VertexPositionColor(new Vector3(2.2F, 1.0F, 2.0F), Color.White),
    new VertexPositionColor(new Vector3(2.5F, 0.0F, 2.0F), Color.White)};
VertexPositionColor[] abanico = {
    new VertexPositionColor(new Vector3(0.0F, 1.0F, 2.0F), Color.White),
    new VertexPositionColor(new Vector3(1.0F, 1.0F, 2.0F), Color.White),
    new VertexPositionColor(new Vector3(1.0F, 0.0F, 2.0F), Color.White),
    new VertexPositionColor(new Vector3(0.5F, -0.5F, 2.0F), Color.White),
    new VertexPositionColor(new Vector3(-0.5F, -1.0F, 2.0F), Color.White)};
```

En este caso se declararon dos arreglos de vértices, el arreglo **vertice** es para mostrar cinco de las seis primitivas de XNA, y el arreglo **abanico** es para mostrar la primitiva de abanico.

Dentro del constructor escriba las siguientes líneas para modificar algunas propiedades de la ventana en que se mostraran las geometrías, son las mismas propiedades que se explicaron en el capítulo anterior.

```
this.IsMouseVisible = true;
this.Window.AllowUserResizing = true;
this.Window.Title = "xintalalai Tutorial 02a";
```

En el método **Initialize** se crean las nuevas instancias del búfer de vértices y el efecto.

```
vertexDeclaration = new VertexDeclaration(GraphicsDevice,
VertexPositionColor.VertexElements);
vertexBuffer = new VertexBuffer(GraphicsDevice, 7 * VertexPositionColor.SizeInBytes,
BufferUsage.WriteOnly);
vertexBuffer.SetData<VertexPositionColor>(vertices, 0, vertices.Length);
effect = new BasicEffect(GraphicsDevice, null);
effect.VertexColorEnabled = true;
```

Lo único que cambia en esta declaración es el número de elementos que contendrá el búfer de vértices, así que se multiplica el número de elementos del arreglo por el tamaño en bytes de la estructura **VertexPositionColor**, es decir:

```
vertexBuffer = new VertexBuffer(GraphicsDevice, vertices.Length *
    VertexPositionColor.SizeInBytes, BufferUsage.WriteOnly);
```

En el llenado del búfer de vértices se utiliza el primer arreglo para mostrar las siguientes primitivas:

- **PointList**
- **LineList**
- **LineStrip**
- **TriangleList**
- **TriangleStrip**

2.1 PointList

La primitiva **PoinList** toma una lista de vértices y los muestra como puntos en el espacio, en el orden en que se encuentran en el búfer de vértices.

Código 2-1

```
1.      GraphicsDevice.Clear(Color.Black);
2.      Single aspecto = GraphicsDevice.Viewport.AspectRatio;
3.      effect.World = Matrix.Identity;
4.      effect.View = Matrix.CreateLookAt(new Vector3(0.0F, 0.0F, 6.0F),
5.          Vector3.Zero,
6.          Vector3.Up);
7.      effect.Projection = Matrix.CreatePerspectiveFieldOfView(1.0F,
8.          aspecto, 1.0F, 10.0F);
9.      GraphicsDevice.RenderState.FillMode = FillMode.WireFrame;
10.     GraphicsDevice.VertexDeclaration = vertexDeclaration;
11.     GraphicsDevice.Vertices[0].SetSource(vertexBuffer, 0,
12.         VertexPositionColor.SizeInBytes);
13.
14.     effect.Begin();
15.     effect.CurrentTechnique.Passes[0].Begin();
16.     GraphicsDevice.DrawPrimitives(PrimitiveType.PointList, 0, 7);
17.     effect.CurrentTechnique.Passes[0].End();
18.     effect.End();
```

El Código 2-1 se deberá escribir dentro del método **Draw**, note que la mayoría del código es el mismo que en el capítulo anterior, excepto la línea 16 en donde el tipo de primitiva será **PointList** y el número máximo de elementos que se dibujarán serán 7; por lo que podrá cambiarlo por la propiedad **Lenght** del arreglo **vertices**.

Inicie la aplicación y verá siete puntos como en la Ilustración 2-1.

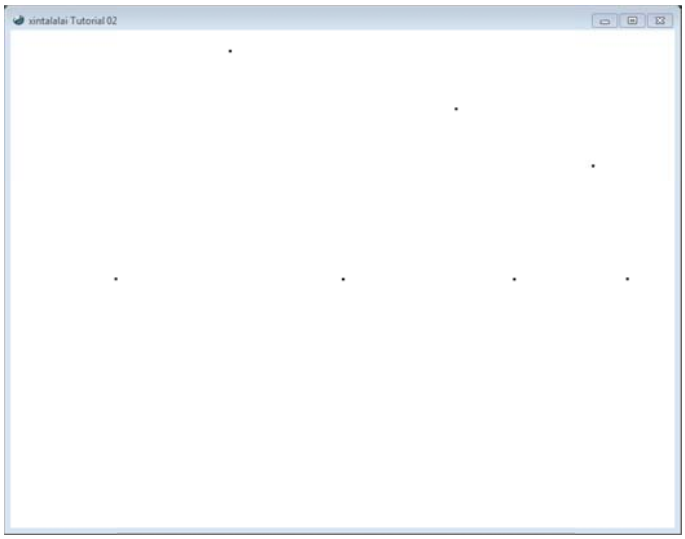

Ilustración 2-1 PointList

2.2 LineList

LineList es una primitiva que dibuja líneas rectas a partir de un par de vértices del búfer de vértices. En el ejemplo se tienen siete vértices con lo que se dibujarán tres líneas, el último no se ocupa.

Modifique el método **DrawPimitives** del Código 2-1, línea 16, cambiando el parámetro **PrimitiveType.PointList** por **PrimitiveType.LineList** y el número de elementos a tres.

```
GraphicsDevice.DrawPrimitives(PrimitiveType.LineList, 0, 3);
```

Oprima **F5** y verá algo similar a la Ilustración 2-2. Como se puede ver, las líneas van del **vertice** 0 al 1, del 2 al 3 y del 4 al 5.

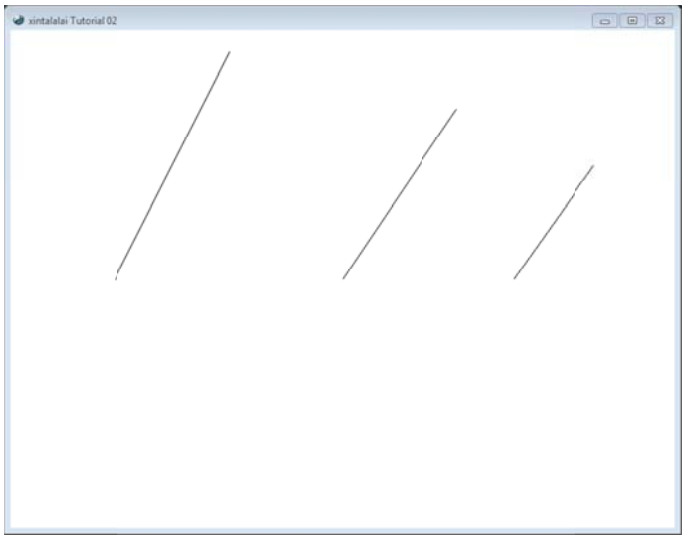

Ilustración 2-2 LineList

2.3 LineStrip

LinStrip es la primitiva que une dos puntos para crear una línea recta, tomando como punto de inicio el segundo punto de la línea anterior, excepto la primera línea que no le antecede otra.

El número de rectas que se pueden dibujar dado un número de vértices será igual a:

$$Número\ de\ LineStrip = Número\ de\ vértices - 1$$

$$Número\ de\ vértices \geq 2$$

Modifique el tipo de primitiva del Código 2-1, línea 16, por **PrimitiveType.LineStrip** y el número de elementos a dibujar por seis, o por el número de elementos del arreglo vértice menos uno.

```
GraphicsDevice.DrawPrimitives(PrimitiveType.LineStrip, 0, 6);
```

Oprima **F5** y vera una imagen similar a la Ilustración 2-3.

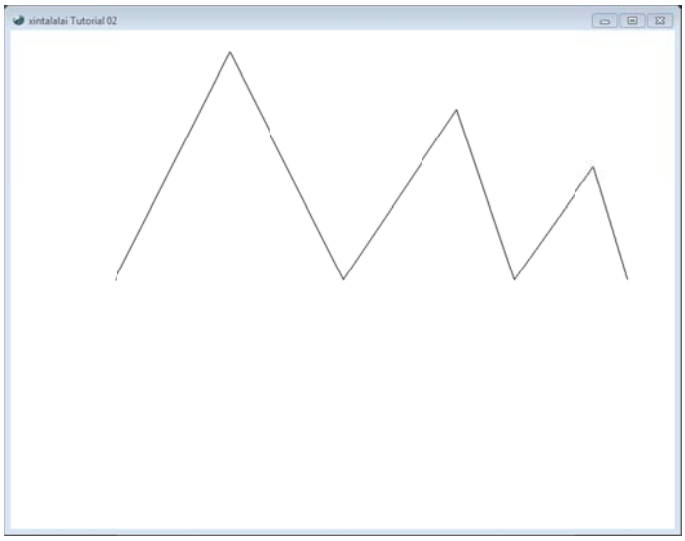

Ilustración 2-3LineStrip

2.4 Modos de Culling

El culling es un algoritmo que determina la visibildad de las caras, sirve para dibujar sólo aquella geometría que se alcanza a ver, por ejemplo, en un cubo a lo más que podemos ver son tres de sus caras. La manera de conocer qué caras se dibujarán es por medio de la normal de ésta. La normal es un vector perpendicular al plano y la dirección de éste dependerá del sentido en que se vayan dibujando las primitivas.

Por default XNA tiene el modo **CullCounterClockwiseFace** activado, es decir, oculta aquellas caras que por orden en el búfer de vértices se asemejan al sentido contrario a las manecillas del reloj, más adelante se verá un ejemplo. El modo **CullClockwiseFace**, oculta aquellas caras que en el búfer de vértices tengan el orden semejante al sentido de las manecillas del reloj. El modo **None** hace caso omiso del modo de culling, para dibujar todas las caras.

Escriba dentro del método Draw en el Código 2-1, línea 13, el modo de culling **None**.

```
GraphicsDevice.RenderState.CullMode = CullMode.None;
```

Una manera sencilla de conocer qué caras no son dibujadas, es por medio de la regla de la mano derecha para el caso de **CullCounterClockwiseFace** y la regla de la mano izquierda para el caso **CullClockwiseFace**.

2.5 TriangleList

TriangleList es la primitiva que toma triadas de vértices para dibujar un triángulo. La manera de saber qué número de elementos se pueden dibujar dado un número de vértices es la siguiente:

$$Número\ de\ TriangleList = \frac{Número\ de\ Vértices}{3}$$

$$Número\ de\ TriangleList\ \geq 1$$

$$Número\ de\ TriangleList \in \mathbb{N}$$

El número de elementos será igual a la parte entera del resultado de la división, además de que debe ser mayor a cero.

Modifique el método **Draw** en el Código 2-1, línea 16; el primer parámetro cámbielo por **Primitive.TriangleList** y el tercer parámetro por 2.

```
GraphicsDevice.DrawPrimitives(PrimitiveType.TriangleList, 0, 2);
```

Oprima **F5** para iniciar la aplicación y verá una imagen similar a la Ilustración 2-4.

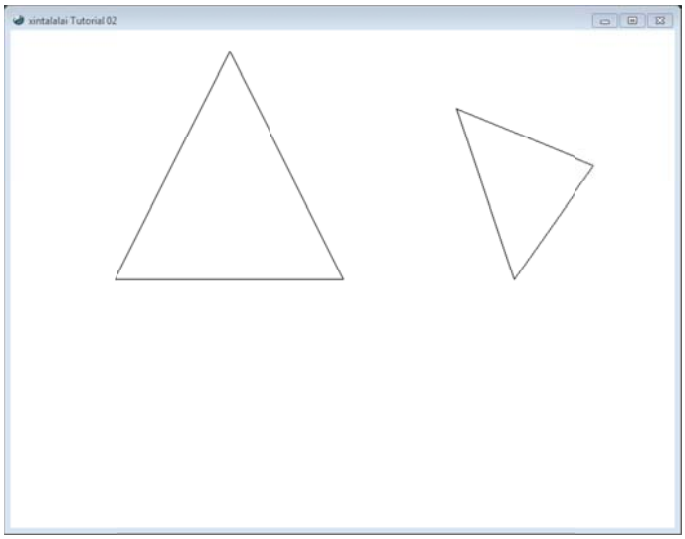

Ilustración 2-4 CullMode.None

Como puede ver, la primitiva toma los tres primeros vértices para crear el triángulo de la derecha, posteriormente toma los siguientes tres vértices para dibujar el triángulo de la izquierda, por lo que no se toma el último vértice.

Ahora modifique la línea 13 del Código 2-1 por **CullClockwiseFace**, para que oculte el triángulo cuya primitiva se dibuje en el sentido de la manecillas del reloj.

```
GraphicsDevice.RenderState.CullMode = CullMode.CullClockwiseFace;
```

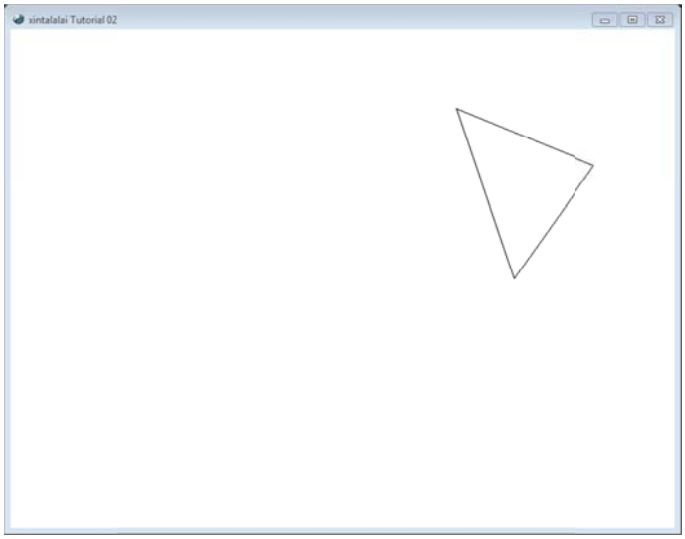

Ilustración 2-5 CullMode.CullClockwiseFace

Después cambie el modo del culling a **CullCounterClockwiseFace** para ocultar el triángulo cuya primitiva se dibuje en el sentido contrario a las manecillas del reloj.

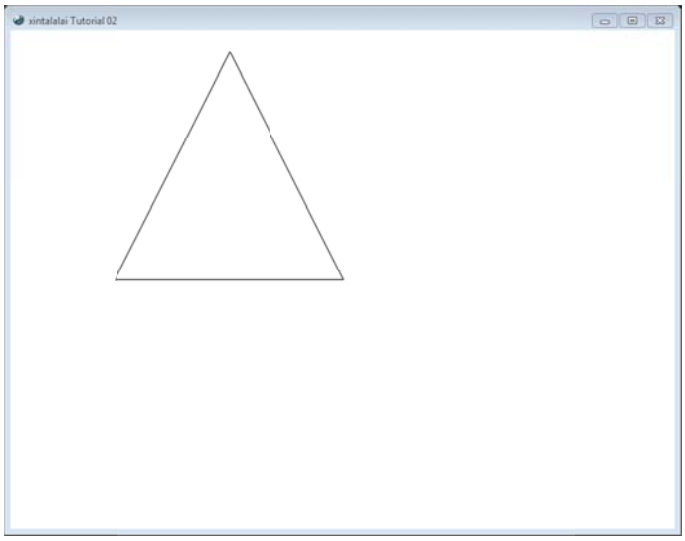

Ilustración 2-6 CullMode.CullCounterClockwiseFace

2.6 TriangleStrip

TriangleStrip es la primitiva que permite dibujar con eficiencia triángulos, pues toma dos de los últimos vértices del triángulo anterior más un nuevo vértice para crearlo, excepto la primera triada de **vertices** del búfer de vértices. Es decir, el primer triángulo toma los vértices 1, 2 y 3; el segundo toma los vértices 2, 3 y 4; el tercer triángulo toma los vértices 3, 4 y 5.hasta concluir con la lista contenida en el búfer de vértices.

Cambie el método **DrawPrimitives** en el Código 2-1, línea 16, en el primer y último parámetro.

```
GraphicsDevice.DrawPrimitives(PrimitiveType.TriangleStrip, 0, 5);
```

Inicie la aplicación con la tecla **F5**, y verá algo similar a la Ilustración 2-7.

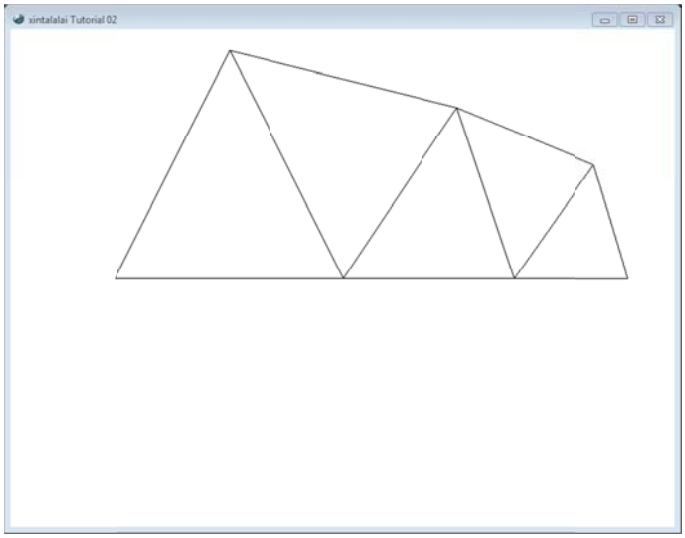

Ilustración 2-7 TriangleStrip

Para conocer la cantidad máxima de **TriangleStrip** que se pueden dibujar a partir de un número dado de vértices se resta el número de vértices menos dos.

$$Número\ de\ TriangleStrip = Número\ de\ vértices - 2$$

2.7 TriangleFan

Esta última primitiva utiliza el primer vértice del búfer de vértices, el último vértice del triángulo anterior y el vértice siguiente. Es decir, el primer triángulo está formado por los vértices 1, 2 y 3; el segundo triángulo por los vértices 1, 3 y 4; el tercer triángulo está constituido por los vértices 1, 4 y 5.

De nueva cuenta, cambie los parámetros uno y tres del método **DrawPimitives**, del Código 2-1, línea 16.

```
GraphicsDevice.DrawPrimitives(PrimitiveType.TriangleFan, 0, 3);
```

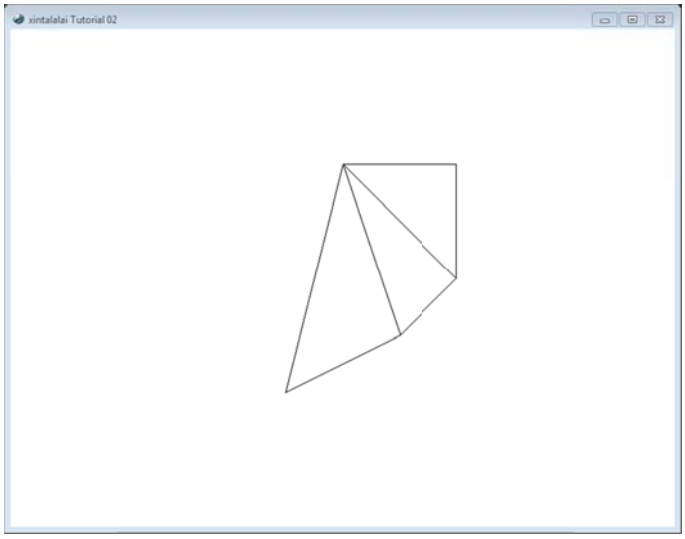

Ilustración 2-8 TriangleFan

El número de **TriangleFan** máximos que se pueden extraer de una lista de vértices, se calcula al restar dos, al número de vértices del búfer, con lo cual se obtiene la siguiente fórmula:

$$Número\ de\ TriangleFan = Número\ de\ vértices - 2$$