

DirectX 9 Managed Code with VB.net: A crash course

By TzeJian Chear (base upon Riemer's tutorials)

Device Linking

1. Create a new windows application project; add reference to Microsoft.DirectX and Microsoft.DirectX.Direct3D. Make sure they have the same version.
2. Select the form and press F7. To use DirectX, add these lines before the class structure:

```
Imports Microsoft.DirectX
Imports Microsoft.DirectX.Direct3D
```

3. 3D drawings are managed by a Direct3D device, but you've to create it first. You must also give certain parameters (like setting up a canvas). Write these lines:

```
Public Class Form1
    Private device As Direct3D.Device
    Public Sub Initialize()
        Dim present As PresentParameters = New PresentParameters
        present.Windowed = True 'we'll draw on a window
        present.SwapEffect = SwapEffect.Discard 'discuss later
        device = New Direct3D.Device(0, DeviceType.Hardware, Me, _
            CreateFlags.SoftwareVertexProcessing, present)
    End Sub
End Class
```

4. Argument explanation in order: 0 for default adapter, use hardware acceleration, Me is the form D3D will create its canvas upon, and we use software to process vertices. Note: Call Initialize() only ONCE before anything else, preferably in Form_Load event.

Clearing screen

1. We *Clear* the form with black color whenever paint event is called (determined by OS) and always *Present* it so it'll get drawn. Add these lines to Form_Paint event:

```
device.Clear(ClearFlags.Target, Color.Black, 1.0, 0)
device.Present()
```

Perpetual painting

1. Paint event gets called only when the form needs to be redrawn, which we don't want. We want it to be redrawn perpetually. Add this line before calling Initialize():

```
Me.SetStyle(ControlStyles.AllPaintingInWmPaint Or _
ControlStyles.Opaque, True) 'Do not draw form's background
```

2. Add this as the last line in the Form_Paint event: `Me.Invalidate() 'redraw`

Drawing triangle

1. All objects are drawn using triangles made of 3 vertices (points) detailing position, color, etc. Add these codes before device.Clear to create the vertices.

```
Dim vertices As CustomVertex.TransformedColored() = New _
CustomVertex.TransformedColored(0 To 2) {} 'create an array of vertices
```

```

vertices(0).Position = New Vector4(150, 100, 0, 1)
vertices(0).Color = Color.Red.ToArgb 'encode color in Argb
vertices(1).Position = New Vector4(Me.Width/2 + 100,100,0,1)
vertices(1).Color = Color.Green.ToArgb
vertices(2).Position = New Vector4(250, 300, 0, 1)
vertices(2).Color = Color.Yellow.ToArgb

```

2. Add these codes after device.Clear and before device.Present.

```

device.BeginScene() 'all drawings after this line
device.VertexFormat = CustomVertex.TransformColored.Format
device.DrawUserPrimitives(PrimitiveType.TriangleList, 1, vertices)
device.EndScene() 'all drawings before this line

```

3. Vertex format tells what kind of vertices Direct3D is drawing. DrawUserPrimitives actually draws the vertices based on the vertex format.

Camera

1. The positions of the vertices above are already in the form of screen positions. We will use world coordinate to describe them and let Direct3D convert them to screen positions. Replace the vertex initialization codes with these:

```

Dim vertices As CustomVertex.PositionColored() = New _
CustomVertex.PositionColored(2) {}
    vertices(0).Position = New Vector3(0, 0, 0)
    vertices(0).Color = Color.Red.ToArgb
    vertices(1).Position = New Vector3(10, 0, 0)
    vertices(1).Color = Color.Green.ToArgb
    vertices(2).Position = New Vector3(5, 10, 0)
    vertices(2).Color = Color.Yellow.ToArgb

```

2. Tell Direct3D we're using a different vertex format. Change this in Paint event:

```

device.VertexFormat = CustomVertex.PositionColored.Format

```

3. Set our camera's position and direction to point at the triangle's direction. Otherwise the triangle can't be seen. Add these lines right after we created the device:

```

device.Transform.Projection = Matrix.PerspectiveFovLH(CSng(Math.PI/4), _
Me.Width / Me.Height, 1, 50) 'sets field of view, aspect ratio, etc
device.Transform.View = Matrix.LookAtLH(New Vector3(0, 0, 30), _
New Vector3(0, 0, 0), New Vector3(0, 1, 0)) 'position and direction

```

4. The triangle is invisible because there's no light to illuminate it. We tell Direct3D to disregard lighting conditions. Add this line after we created the device:

```

device.RenderState.Lighting = False

```

5. Note: Direct3D uses left-hand coordinates, so the green vertex, which is defined on a positive x-axis, appears on left instead. If we instead place the camera behind the triangle (negative z-position for camera), that vertex should appear on right. But Direct3D draws only triangles whose vertices are defined clockwise relative to the camera, so the triangle

won't be drawn if the camera's behind (the vertices become counter-clockwise relative to the hind camera). Either we reinitialize the vertices to be relatively clockwise, or we tell Direct3D to draw ALL triangles indiscriminately (slows down the machine, therefore not recommended). Should you want to draw all triangles, add this line after creating device:

```
device.RenderState.CullMode = Cull.None 'no triangle is culled
```

Vertices Initialization

1. If you notice, we re-initialize the same vertices every time before we draw. We should do this only once. Add this line right after `Public Class Form1`.

```
Private vertices As CustomVertex.PositionColored() 'an array of vertices
```

2. Remove the vertices initialization code in Paint event and add this procedure:

```
Private Sub InitVertices()  
    vertices = New CustomVertex.PositionColored(2) {}  
    vertices(0).Position = New Vector3(0, 0, 0)  
    vertices(0).Color = Color.Red.ToArgb  
    vertices(1).Position = New Vector3(10, 0, 0)  
    vertices(1).Color = Color.Green.ToArgb  
    vertices(2).Position = New Vector3(5, 10, 0)  
    vertices(2).Color = Color.Yellow.ToArgb  
End Sub
```

3. Call `InitVertices()` only ONCE right after calling `Initialize()` procedure.

Rotation and Translation

1. To keep track of rotation angle, add this code after `Public Class Form1`:

```
Private angle As Single = 0.0
```

2. To actually rotate, add these lines right before calling `DrawUserPrimitives`:

```
angle = angle + 0.1  
device.Transform.World = Matrix.RotationZ(angle)
```

4. To translate (move around) the vertices, say, to (-3, 5, 2), replace with this line:

```
device.Transform.World = Matrix.Translation(-3, 5, 2)
```

5. To rotate around its self, FIRST TRANSLATE such that the triangle's center is at origin, and THEN ROTATE it. They must be multiplied together. Replace with this line:

```
device.Transform.World = Matrix.Translation(-5, -10 * 1 / 3, 0) * _  
Matrix.RotationZ(angle)
```

6. To rotate around a custom-axis, replace `RotationZ` with this:

```
Matrix.RotationAxis(New Vector3(angle*0.2, angle*1.5, angle*3), angle)
```

Shared Vertices and Indices

1. For many vertices, it's better to put all in a *vertex buffer*, and stream them to the device in one go. We use a separate buffer (an *index buffer*), to tell Direct3D the vertices' sequence (which vertex comes next). Add these lines after `Public Class Form1`:

```
Private indices As Short()  
Private vb As VertexBuffer  
Private ib As IndexBuffer
```

2. Replace your `InitVertices()` procedure with this:

```
Private Sub InitVertices()  
    vb = New VertexBuffer(GetType(CustomVertex.PositionColored), _  
        5, device, Usage.WriteOnly Or Usage.Dynamic, _  
        CustomVertex.PositionColored.Format, Pool.Default)  
    vertices = New CustomVertex.PositionColored(4) {} '5 vertices  
    vertices(0).Position = New Vector3(0, 0, 0)  
    vertices(0).Color = Color.Yellow.ToArgb  
    vertices(1).Position = New Vector3(5, 0, 0)  
    vertices(1).Color = Color.Yellow.ToArgb  
    vertices(2).Position = New Vector3(10, 0, 0)  
    vertices(2).Color = Color.Yellow.ToArgb  
    vertices(3).Position = New Vector3(5, 5, 0)  
    vertices(3).Color = Color.Yellow.ToArgb  
    vertices(4).Position = New Vector3(10, 5, 0)  
    vertices(4).Color = Color.Yellow.ToArgb  
    vb.SetData(vertices, 0, LockFlags.None)  
End Sub
```

3. Argument Explanation for `VertexBuffer`: specify the type of vertex, number of vertices, create it on our created device, with the buffer write only and can be dynamically moved between AGP and System memory, the vertex format, and default buffer location, which is AGP memory. All we did is just creating a vertex buffer and *SetData* the vertices to it. `LockFlags.None` means Direct3D doesn't have to lock the buffer while we write on it.

4. Now add this procedure to initialize the index buffer:

```
Private Sub InitIndices()  
    ib = New IndexBuffer(GetType(Short), 6, device, Usage.WriteOnly, _  
        Pool.Default)  
    indices = New Short(5) {} '6 indices for two triangles' points  
    indices(0) = 0  
    indices(1) = 1 '← this index points to vertex 1  
    indices(2) = 3  
    indices(3) = 1 'vertex 1 is reused again  
    indices(4) = 2  
    indices(5) = 4  
    ib.SetData(indices, 0, LockFlags.None)  
End Sub
```

5. We have only 5 vertices but 6 indices because one of the vertices is reused in the second triangle. Now replace `DrawUserPrimitives` in paint event with these lines:

```

device.RenderState.FillMode = FillMode.WireFrame
device.SetStreamSource(0, vb, 0)
device.Indices = ib
device.DrawIndexedPrimitives(PrimitiveType.TriangleList, 0, 0, 5, 0, 2)

```

6. We want to draw wire frame instead of solid. We set the source of the stream (vb) and let the device know as well the sequence (ib).

Procedural Initialization

1. It is good practice to put different things in different procedures. Move the device render states, Projection and View transformation codes into this procedure:

```

Private Sub InitCamera()
device.Transform.Projection = Matrix.PerspectiveFovLH(Math.PI / 4, _
aspect, 1, 50)
device.Transform.View = Matrix.LookAtLH(New Vector3(0, 0, 30), _
New Vector3(0, 0, 0), New Vector3(0, 1, 0))
device.RenderState.Lighting = False
device.RenderState.FillMode = FillMode.WireFrame
End Sub

```

2. Create an InitRes procedure that calls InitCamera, InitVertices and InitIndices in order. Call InitRes after the device is created.

Resizing

1. All resources (vertices, device, etc) are “tailor-made” for the settings and size you started out with; they disappear and become lost when you change the form size. We should reset them when this happens. Change Direct3D device object declaration to this:

```

Private WithEvents Device As Direct3D.Device'WithEvents: handle events

```

2. The DeviceReset event occurs during resizing, standby, lost focus, etc. Add these lines:

```

Private Sub DevReset(ByVal sender As Object, ByVal e As _
System.EventArgs) Handles device.DeviceReset
InitRes()'btw, we can do this in device_DeviceReset event too
End Sub

```

Terrain Creation

1. A terrain is a grid formed by many connected triangles with varying heights. Add these in the form declaration section, just below the line `Public Class Form1`:

```

Private TWIDTH As Integer = 4
Private THEIGHT As Integer = 3
Private heightData(,) As Integer

```

2. TWIDTH and THEIGHT specify the size of the terrain. Add this sub to the class:

```

Private Sub LoadHeightData()
ReDim heightData(TWIDTH, THEIGHT)

```

```

heightData(0, 0) = 0
heightData(1, 0) = 0
heightData(2, 0) = 0
heightData(3, 0) = 0

heightData(0, 1) = 1
heightData(1, 1) = 0
heightData(2, 1) = 2
heightData(3, 1) = 2

heightData(0, 2) = 2
heightData(1, 2) = 2
heightData(2, 2) = 4
heightData(3, 2) = 2
End Sub

```

3. Now call this sub before anything else in the InitRes sub: LoadHeightData()

4. We need 12 vertices for our 4x3 terrain. Change your InitVertices sub:

```

vb = New VertexBuffer(GetType(CustomVertex.PositionColored), _
    TWIDTH * THEIGHT, _Device, Usage.Dynamic Or Usage.WriteOnly, _
    CustomVertex.PositionColored.Format, Pool.Default)
vertices = New CustomVertex.PositionColored(TWIDTH * THEIGHT - 1) {}
Dim x, y As Integer
For x = 0 To TWIDTH - 1
    For y = 0 To THEIGHT - 1
        vertices(x + y * TWIDTH).Position = New Vector3(x, y, 0)
        vertices(x + y * TWIDTH).Color = Color.White.ToArgb
    Next
Next
vb.SetData(vertices, 0, LockFlags.None)

```

5. We'll create the triangles by renumbering the indices. Change your InitIndices sub:

```

ib = New IndexBuffer(GetType(Short), (TWIDTH-1)*(THEIGHT-1)*3, _
    Device, Usage.WriteOnly, Pool.Default)
indices = New Short((TWIDTH - 1) * (THEIGHT - 1) * 3 - 1) {}
Dim x, y As Integer
For x = 0 To TWIDTH - 2 'below: triangle vertices in counter clock-wise
    For y = 0 To THEIGHT - 2
        indices((x+(y*TWIDTH-1))*3)=(x+1)+(y+1)*TWIDTH 'top right
        indices((x+(y*TWIDTH-1))*3+1)=(x+1)+y*TWIDTH 'bottom right
        indices((x+(y*TWIDTH-1))*3+2)=x+y*TWIDTH 'bottom left
    Next
Next
ib.SetData(indices, 0, LockFlags.None)

```

6. Next we change the codes between BeginScene and EndScene in Render sub:

```

Device.VertexFormat = CustomVertex.PositionColored.Format
Device.SetStreamSource(0, vb, 0)
Device.Indices = ib
Device.DrawIndexedPrimitives(PrimitiveType.TriangleList, 0, 0,
    TWIDTH * THEIGHT, 0, indices.Length / 3)

```

7. Change camera position to (0, 0, 15) and CullMode to Cull.None. At this point you should see 6 triangles. Now add height to the vertices by changing a line in InitVertices:

```
vertices(x + y * TWIDTH).Position = New Vector3(x, y, heightData(x, y))
```

8. We draw the second set of triangles by adding more indices. Change your InitIndices:

```
ib = New IndexBuffer(GetType(Short), (TWIDTH-1)*(THEIGHT-1)*6, _
    Device, Usage.WriteOnly, Pool.Default)
indices = New Short((TWIDTH - 1) * (THEIGHT - 1) * 6 - 1) {}
Dim x, y As Integer
For x = 0 To TWIDTH - 2 'below: triangle vertices in counter clock-wise
    For y = 0 To THEIGHT - 2
        indices((x+(y*TWIDTH-1))*6) = (x+1)+(y+1)*TWIDTH 'top right
        indices((x+(y*TWIDTH-1))*6+1) = (x+1)+y*TWIDTH 'bottom right
        indices((x+(y*TWIDTH-1))*6+2) = x+y*TWIDTH 'bottom left
        'second set of triangles
        indices((x+y*(TWIDTH-1))*6+3) = (x+1)+(y+1)*TWIDTH 'top right
        indices((x+y*(TWIDTH-1))*6+4) = x+y*TWIDTH 'bottom left
        indices((x+y*(TWIDTH-1))*6+5) = x+(y+1)*TWIDTH 'top left
    Next
Next
ib.SetData(indices, 0, LockFlags.None)
```

Loading Terrain from File