Effecten creëren met pixel shaders

En ze uitvoeren op de grafische kaart

Ed van de Pitte

Met de introductie van .NET 3.5sp1 worden de BitmapsEffects vervangen door effecten die op de grafische kaart worden uitgevoerd. Behalve de standaard aanwezige effecten kun je ook zelf fraaie grafische effecten schrijven die op de grafische kaart uitgevoerd kunnen worden. Dit artikel geeft een introductie van het maken van eigen effecten.

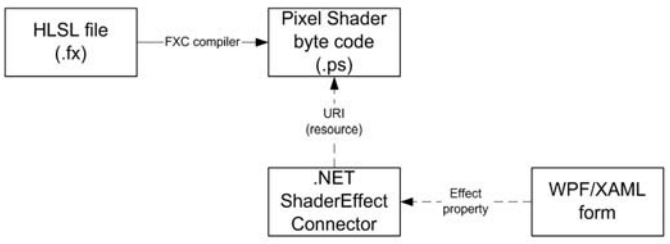
Met 3D grafische kaarten kunnen vele operaties parallel uitgevoerd worden. Dit levert een enorm voordeel op qua performance, maar ook qua responsiveness van je applicatie, omdat de operaties niet (meer) in de main-thread van de applicatie worden uitgevoerd. De grafische kaart kent twee hoofdstappen, de vertex shader die 3D omzet naar 2D en de pixel shader die de kleur voor ieder punt op de monitor bepaald. .NET 3.5sp1 levert de mogelijkheid om effecten te maken die als pixel shader op de grafische kaart uitgevoerd worden. Indien de grafische kaart geen ondersteuning biedt voor pixel shaders, dan zal DirectX de operaties via software uitvoeren.

Het maken van een effect

Randvoorwaarden voor het ontwikkelen van eigen pixel shaders zijn:

- .NET 3.5 service pack 1 of hoger
- DirectX 9 of hoger
- DirectX SDK
- Visual Studio 2008

Figuur 1 geeft een schematisch overzicht van de stappen om een eigen pixel shader effect te maken en te gebruiken.



FIGUUR 1: STAPPEN OM EEN PIXEL SHADER EFFECT TE MAKEN.

Het maken van een eigen effect bestaat uit de volgende stappen:

- 1. Schrijven van de pixel shader code. De pixel shader code wordt beschreven in High Level Shader Language (HLSL). Dit is een taal die erg lijkt op de programmeertaal 'C'.
- 2. De HLSL code wordt door de FXC compiler omgezet in bytecode die door de grafische kaart uitgevoerd kan worden. Deze bytecode moet als resource in de assembly opgenomen worden. Let op dat de FXC compiler alleen ASCII files kan compileren. Er zullen onverwachte foutmeldingen gerapporteerd worden als de HLSL file unicode is.
- 3. Er moet een .NET class worden gemaakt die de koppeling tussen de pixel shader bytecode en WPF verzorgd. Deze class wordt verder de ShaderEffect-Connector genoemd.
- 4. De WPF of XAML code moet de ShaderEffect-connector toekennen aan de effect property van het UIElement waarop het effect moet worden toegepast.

1. Schrijven van de pixel shader

De HLSL code hieronder is een pixel shader effect waarbij het aandeel per kleur bepaald kan worden. Als parameters worden de volgende zaken doorgegeven:

- Het image dat 'bewerkt' zal worden.
  Images worden via 'sampler2D' registers doorgegeven. Het voorbeeld gebruikt sampler register s0 voor het input-image.
- De factoren per kleur.
  De factoren worden doorgegeven in 'constant' registers. In het voorbeeld wordt register c0 voor de rood-factor, c1 voor de groen-factor en c2 voor de blauw-factor gebruikt.

De pixel shader zal de 'main' functie (die met COLOR is aangemerkt) aanroepen voor ieder punt dat op het scherm terecht komt.

```
// Texture/image dat doorgegeven wordt.
sampler2D implicitInput : register(s0);
```

```
// Parameters. Per kleur wordt een register gekoppeld dat het
// percentage van die kleur aangeeft.
float factorRed : register(c0);
float factorGreen : register(c1);
float factorBlue : register(c2);

// Functie die uitgevoerd wordt voor ieder punt.
// [in] uv: Relatieve positie van het punt dat berekend wordt.
// [out] float4: Resulterende kleur (r, g, b, a). Iedere kleur is
//           een waarde tussen 0 en 1.
float4 main (float2 uv : TEXCOORD) : COLOR
{
    // Haal huidige kleur op van het punt dat bewerkt wordt.
    float4 color = tex2D(implicitInput, uv);

    // Bereken nieuwe kleur.
    color.r *= factorRed;
    color.g *= factorGreen;
    color.b *= factorBlue;

    return color;
}
```

CODEVOORBEELD 1: PIXEL SHADER OM AANDEEL PER KLEUR TE BEPALEN.

2. Compileren van de HLSL file.
De pixel shader code moet worden gecompileerd naar bytecode die door de grafische kaart gebruikt kan worden. Deze compilatie gebeurd met de FXC-compiler die onderdeel is van de DirectX SDK. Indien Codevoorbeeld 1 opgeslagen is in de (ASCII) file 'RGBFilterEffect.fx', dan ziet compilatie als volgt uit:

```
"%DXSDK_DIR%"\Utilities\Bin\x86\FXC.EXE /T ps_2_0 /Fo RGBFilter-
Effect.ps RGBFilterEffect.fx
```

Deze regel compileert de file RGBFilterEffect.fx naar (/Fo) RGB-FilterEffect.ps, waarbij pixel shader versie 2.0 byte code gegenereerd zal worden (/T ps_2_0).
Na succesvolle compilatie zal een RGBFilterEffect.ps file aangemaakt zijn. Deze bytecode moet als resource in je assembly worden opgenomen. De compilatie van de pixel shader code kan in de pre-build events van je assembly uitgevoerd worden.

3. De ShaderEffect-connector class maken.
Om de pixel shader bytecode in WPF te gebruiken moet een C# class worden gemaakt die de parameters van de applicatie aan de pixel shader doorgeeft en die aangeeft welke pixel shader bytecode moet worden gebruikt. Voor dit doel is aan .NET 3.5sp1 de ShaderEffect base class toegevoegd waarvan een afgeleide class moet worden gemaakt om de koppeling met de eigen pixel shader te realiseren.
De assembly waarin de ShaderEffect-connector geïmplementeerd wordt moet de volgende referenties hebben:

- PresentationCore
- PresentationFramework
- WindowsBase

Codevoorbeeld 2 toont de code die de koppeling legt tussen WPF en de 'RGBFilterEffect' pixel shader. De aanname is dat de pixel shader bytecode en de ShaderEffect-connector class samen opgenomen zijn in een eigen assembly met als naam 'ShaderEffects'.

```
using System;
using System.Windows.Media.Effects;
using System.Windows;
using System.Windows.Media;

namespace ShaderEffects
{
    public class RGBFilterEffect : ShaderEffect
    {

private static PixelShader _pixelShader = new PixelShader() {
UriSource = new Uri(@"pack://application:,,,/ShaderEffects;component/
RGBFilterEffect.ps") };

        public RGBFilterEffect()
        {
            // Koppel base PixelShader met eigen pixelshader.
            base.PixelShader = _pixelShader;
            // Upload de initiele waardes.
            UpdateShaderValue(InputProperty);
            UpdateShaderValue(RedFactorProperty);
            UpdateShaderValue(GreenFactorProperty);
            UpdateShaderValue(BlueFactorProperty);
        }

        // Input image.
        public static readonly DependencyProperty InputProperty =
ShaderEffect.RegisterPixelShaderSamplerProperty("Input",
typeof(RGBFilterEffect), 0);
        public Brush Input
        {
            get { return (Brush)GetValue(InputProperty); }
            set { SetValue(InputProperty, value); }
        }

        // Red factor.

public static readonly DependencyProperty RedFactorProperty = De-
pendencyProperty.Register("RedFactor", typeof(double),
typeof(RGBFilterEffect), new UIPropertyMetadata(1.0, PixelShader-
ConstantCallback(0)));
        public double RedFactor
        {
            get { return (double)GetValue(RedFactorProperty); }
            set { SetValue(RedFactorProperty, value); }
        }

        // Green factor.

public static readonly DependencyProperty GreenFactorProperty =
DependencyProperty.Register("GreenFactor", typeof(double),
typeof(RGBFilterEffect), new UIPropertyMetadata(1.0, PixelShader-
ConstantCallback(1)));
        public double GreenFactor
        {
            get { return (double)GetValue(GreenFactorProperty); }
            set { SetValue(GreenFactorProperty, value); }
        }

        // Blue factor.

public static readonly DependencyProperty BlueFactorProperty =
DependencyProperty.Register("BlueFactor", typeof(double),
typeof(RGBFilterEffect), new UIPropertyMetadata(1.0, PixelShader-
ConstantCallback(2)));
        public double BlueFactor
        {
            get { return (double)GetValue(BlueFactorProperty); }
            set { SetValue(BlueFactorProperty, value); }
        }
    }
}
```

CODEVOORBEELD 2: SHADEREFFECT-CONNECTOR VOOR RGBFILTEREFFECT PIXEL SHADER.

De code doet het volgende:

- Creëer een PixelShader instantie die verwijst naar de URI waar de pixel shader byte code te vinden is (de bytecode is in stap 1 opgenomen als resource in je assembly)
- Assign de eigen PixelShader-instantie aan de base class Shader-Effect PixelShader property.
- Definieer een dependency property voor het source-image. Ge-

bruik de 'ShaderEffect.RegisterPixelShaderSamplerProperty' functie om het image te koppelen aan het sampler register van de pixel shader (register s0 in het voorbeeld).

- Definieer dependency properties om de overige (niet image) variabelen door te geven. Dit gebeurt met de 'PixelShaderConstantCallback' functie waarbij het nummer van het constant
- register opgegeven wordt (registers c0, c1 en c2 in het voorbeeld).

4. Het pixel shader effect gebruiken in XAML.
Het nieuwe effect wordt gebruikt door de ShaderEffect-connector toe te kennen aan de Effect-property van het UIElement waarop het effect moet worden toegepast.

Maak in XAML ook een namespace verwijzing naar de ShaderEffect-connect. In onderstaand voorbeeld wordt het RGBFilterEffect toegepast op een image. Het resultaat is te zien in Figuur 2.

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:pixelshadereffect="clr-namespace:ShaderEffects;assembly=Sha
derEffects"
  x:Class="RGBFilterEffectWPF.RGBGilterEffectWindow"
  x:Name="MainWindow"
  Title="RGBFilterEffect Pixel Shader">

<Grid x:Name="LayoutRoot">
<Image Source="TutAmon.jpg" Stretch="Fill">
<Image.Effect>
<pixelshadereffect:RGBFilterEffect
RedFactor="0.5"
GreenFactor="1.0"
BlueFactor="0.3" />
</Image.Effect>
</Image>
</Grid>
</Window>
```

CODEVOORBEELD 3: WPF CODE OM HET EFFECT TE GEBRUIKEN.

FIGUUR 2: RESULTAAT VAN DE PIXEL SHADER.

In plaats van aparte variabelen per kleur kan ook een enkele variabele waarin een Color wordt doorgegeven. Doordat een Color bestaat uit 4 delen (R, G, B en Alpha) moet de pixel shader code aan-

gepast worden zodat deze een kleur structuur kan ontvangen. Voor een Color moet je gebruik maken van het float4 type. .NET zorgt zelf voor de conversie van .NET kleuren naar pixel shader kleuren.

```
// Source Texture/image.
sampler2D implicitInput : register(s0);

// De filter kleur wordt als 1 Color structuur doorgegeven
float4 filterColor : register(c0);

// Pixel Shader functie.
float4 main (float2 uv : TEXCOORD) : COLOR
{
    // Haal kleur op van het punt dat bewerkt moet worden.
    float4 color = tex2D(implicitInput, uv);

    // Vermenigvuldig de kleuren in 1 operatie.
    color *= filterColor;

    return color;
}
```

CODEVOORBEELD 4: PIXEL SHADER CODE WAARBIJ EEN COLOR DOORGEGEVEN WORDT.

Het koppelen van het Color type aan de dependency property gebeurd op precies dezelfde manier als voor een enkelvoudig datatype.

```
// Color property.
public static readonly DependencyProperty ColorFilterProperty =
DependencyProperty.Register("ColorFilter", typeof(Color),
typeof(RGBFilterEffect),
new UIPropertyMetadata(Colors.White,PixelShaderConstantCallbac
k(0)));
```

CODEVOORBEELD 5: DEPENDENCY PROPERTY OM EEN COLOR DOOR TE GEVEN.

Multi image pixel shader

In plaats van een enkel image kunnen ook meerdere images worden gebruikt. Ook deze images worden via een sampler2D register doorgegeven. Als voorbeeld van een Multi image pixel shader staat hieronder een Mask-filter dat het tweede image als masker over het eerste image heen zal leggen.

```
// Source image.
sampler2D implicitInput : register(s0);
// Mask image.
sampler2D maskInput : register(s1);

// Factor die aangeeft in welke mate de mask toegepast wordt.
// 0.0 = Mask wordt niet gebruikt, 1.0 = Mask wordt 100% gebruikt.
float maskFactor : register(c0);

float4 main (float2 uv : TEXCOORD) : COLOR
{
    float4 sourceColor = tex2D(implicitInput, uv);
    float4 maskColor = tex2D(maskInput, uv);

    // Pas de mask factor toe. Als Factor=0, dan moet alles
    // doorgelaten worden (=wit pixel in mask). Als factor=1
    // dan wordt het masker maximaal toegepast.
    maskColor += (1 - maskFactor);
    // Resulterende waardes moeten tussen 0 en 1 liggen, anders
    // treedt er een versterking op.
    maskColor = clamp (maskColor, 0, 1);

    // Pas de mask toe op het source.
    sourceColor *= maskColor;

    return sourceColor;
}
```

CODEVOORBEELD 6: MASK FILTER PIXEL SHADER.

Er kan niet alleen een enkel image worden gebruikt, maar ook meerdere images kunnen van een effect voorzien worden.

Codevoorbeeld 6 gebruikt register 's1' om het mask-image door te geven. Het koppelen van dit register aan een .NET image gebeurd, net als voor het source image, door middel van de 'RegisterPixelSamplerProperty' functie.

```
public class MaskEffect : ShaderEffect
{
    private static PixelShader _pixelShader = new PixelShader() {
UriSource = new Uri(@"pack://application:,,,/
ShaderEffects;component/MaskFilterEffect.ps") };

    public MaskEffect()
    {
        PixelShader = _pixelShader;

        UpdateShaderValue(InputProperty);
        UpdateShaderValue(MaskProperty);
        UpdateShaderValue(MaskFactorProperty);
    }

    public static readonly DependencyProperty InputProperty =
ShaderEffect.RegisterPixelShaderSamplerProperty("Input",
typeof(MaskEffect), 0);
    public Brush Input
    {
        get { return (Brush)GetValue(InputProperty); }
        set { SetValue(InputProperty, value); }
    }

    public static readonly DependencyProperty MaskProperty = Sha-
derEffect.RegisterPixelShaderSamplerProperty("Mask",
typeof(MaskEffect), 1);
    public Brush Mask
    {
        get { return (Brush)GetValue(MaskProperty); }
        set { SetValue(MaskProperty, value); }
    }

    // Mask Factor. 0 = 0%, 1.0 = 100%
    public static readonly DependencyProperty MaskFactorProperty =
DependencyProperty.Register("MaskFactor", typeof(double),
typeof(MaskEffect),
                                new UIPropertyMetadata(1.0,
PixelShaderConstantCallback(0)));
    public double MaskFactor
    {
        get { return (double)GetValue(MaskFactorProperty); }
        set { SetValue(MaskFactorProperty, value); }
    }
}
```

CODEVOORBEELD 7: SHADEREFFECT-CONNECTOR VOOR MASK EFFECT PIXEL SHADER.

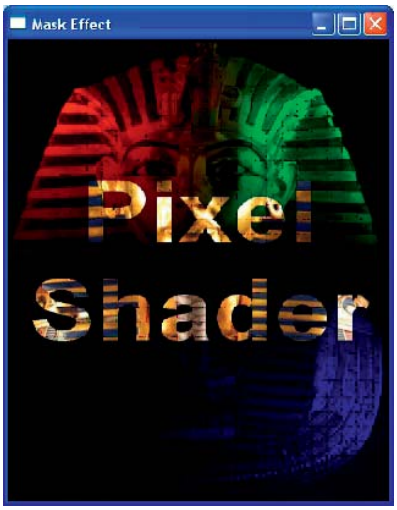
Standaard wordt de 'Input' property van de ShaderEffect-connector gebruikt om het source image te koppelen met de pixel shader. Als er meerdere images doorgegeven worden, dan moeten de overige images expliciet worden gezet.

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:pixelshadereffect="clr-namespace:ShaderEffects;
assembly=ShaderEffects"
  x:Class="MaskFilterEffectWPF.MaskEffectWPF"
  x:Name="MainWindow"
  Title="Mask Effect">

   <Window.Resources>
<ImageBrush x:Key="MaskBitmap" ImageSource="PixelShaderMask.jpg"
/>
    </Window.Resources>

    <Grid x:Name="LayoutRoot" Height="Auto" Width="Auto">
     <Image Source="TutAmon.jpg" Stretch="Fill">
       <Image.Effect>
         <pixelshadereffect:MaskEffect
           MaskFactor="1.0"
           Mask="{StaticResource MaskBitmap}" />
       </Image.Effect>
     </Image>
    </Grid>
</Window>
```

CODEVOORBEELD 8: XAML CODE VOOR MASK EFFECT.



FIGUUR 3: RESULTAAT VAN HET MASK EFFECT.

Transformatie Effecten

Tot dusver zijn alleen effecten besproken waarbij de kleur wordt aangepast. Pixel shaders kunnen images ook vervormen. Als voorbeeld staat hieronder een pixel shader dat het image opschuift in x en/of y richting.

```
// Texture/image dat doorgegeven wordt.
sampler2D implicitInput : register(s0);

// Relatieve verplaatsing (0 = 0%, 1 = 100%)
float shiftX : register(c0);
float shiftY : register(c1);

float4 main (float2 uv : TEXCOORD) : COLOR
{
    // Pas punt aan waar data vanaf gehaald moet worden.
    // Gebruik alleen het getal achter de comma (fractional)
    // om in de range tussen 0 en 1 te blijven.
    uv.x = frac(uv.x - shiftX);
    uv.y = frac(uv.y - shiftY);

    return tex2D(implicitInput, uv);
}
```

CODEVOORBEELD 9: PIXEL SHADER CODE VOOR X,Y SHIFT VAN SOURCE IMAGE.

Pixel shaders kunnen niet alleen kleuren aanpassen, maar kunnen ook images vervormen.

De C# en XAML code voor dit effect bevatten geen nieuwigheden en zijn daarom niet opgenomen in dit artikel.

FIGUUR 4: RESULTAAT VAN X,Y SHIFT PIXEL SHADER.

Shaders op het net

Ondanks het feit dat de pixel shaders voor WPF nog maar kort bruikbaar zijn, zijn er op internet al veel tutorials, shaders en tools te vinden. Op codeplex staat al een hele assembly met kant en klare effecten (http://www.codeplex.com/wpffx). Het Shazzam-tool (http://shazzam-tool.com/) is een tool waarmee pixel shaders en de ShaderEffect connectoren gemaakt kunnen worden.

Links

http://blogs.msdn.com/greg_schechter/
http://www.codeplex.com/wpffx
http://shazzam-tool.com/
http://www.paradoxalpress.com/LayoutSamples/HLSLReference.pdf

Ed van de Pitte, is softwarearchitect en competence leader bij Task24 waar hij betrokken is bij mechatronica applicaties. Voor vragen en opmerkingen is Ed bereikbaar via Ed.van.de.Pitte@Task24.nl