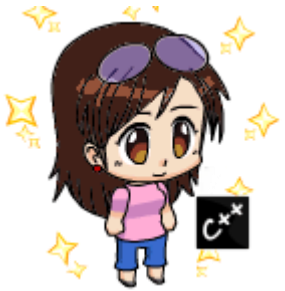
Windows 8でDirectXを使ってみよう！

わんくま同盟 大阪勉強会 #49

遥佐保（はるか・さお）
MetroStyleDeveloper



self-introduction

遥佐保（はるかさお）@hr_sao

某SIer勤務

Microsoft MVP for Client App Dev

勉強会

- MetroStyleDeveloper
- Silverlightを囲む会
- SQLWorld

本日の対象者

- SilverlightなどXAML系のアプリを作った際に、標準のフレームワークでは表現しにくい物を苦労して作ったことがある
- マルチメディア系、ゲーム系のアプリを作る際に、本来の描画処理とは別に、単純な描画のみを行ってくれるライブラリがあったらいいなと思ったことがある

本日の目標

Windows8のメトロスタイルアプリについて
XAMLとDirectXの組み合わせ技術の
種類を理解する

おまけ：C++11ラムダの利用方法を理解する

本日は紹介しない部分

XAMLについて

MetroStyleAppsについて

DirectX11.1の機能について

DirectXの描画処理について

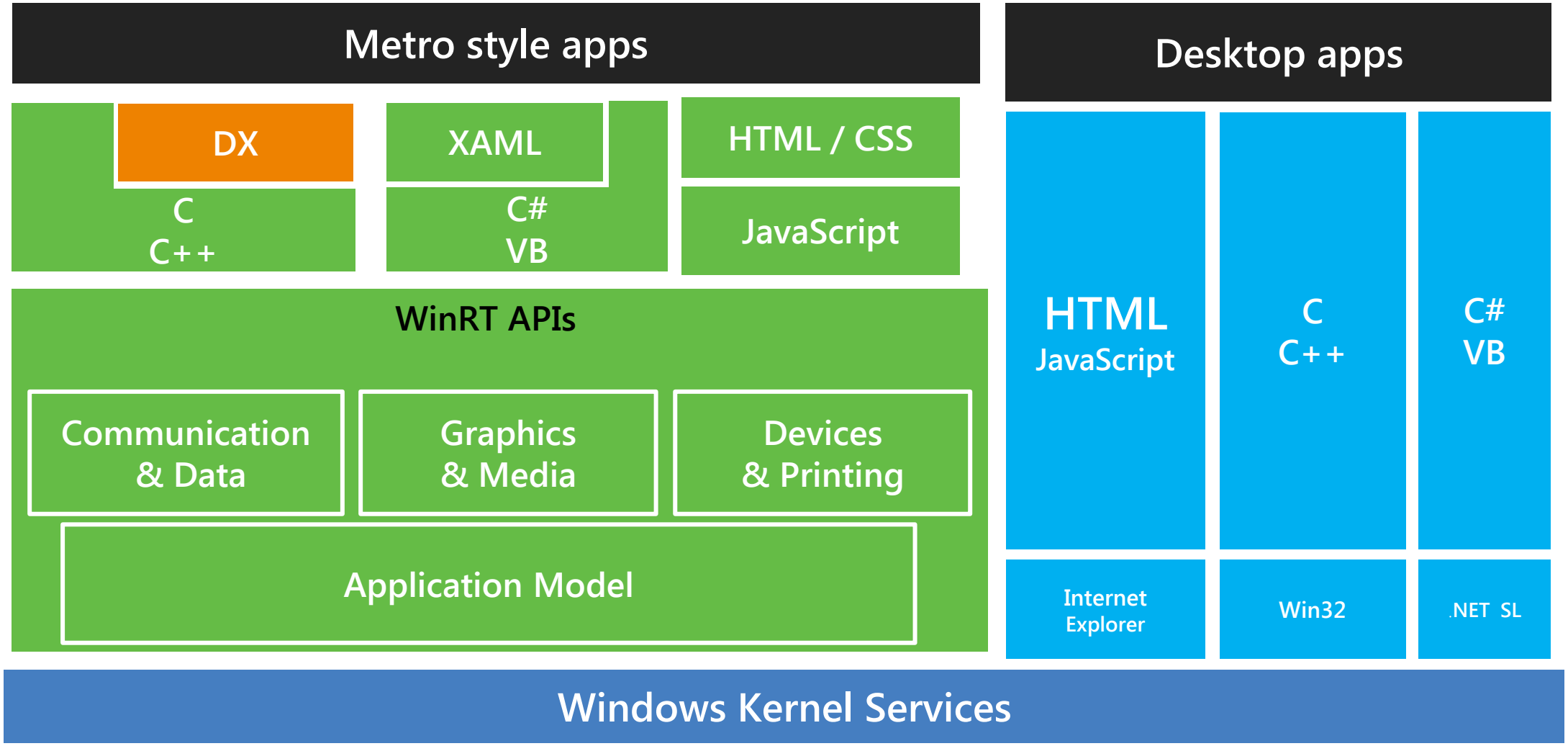
Metro Style Apps の DirectX

DirectX

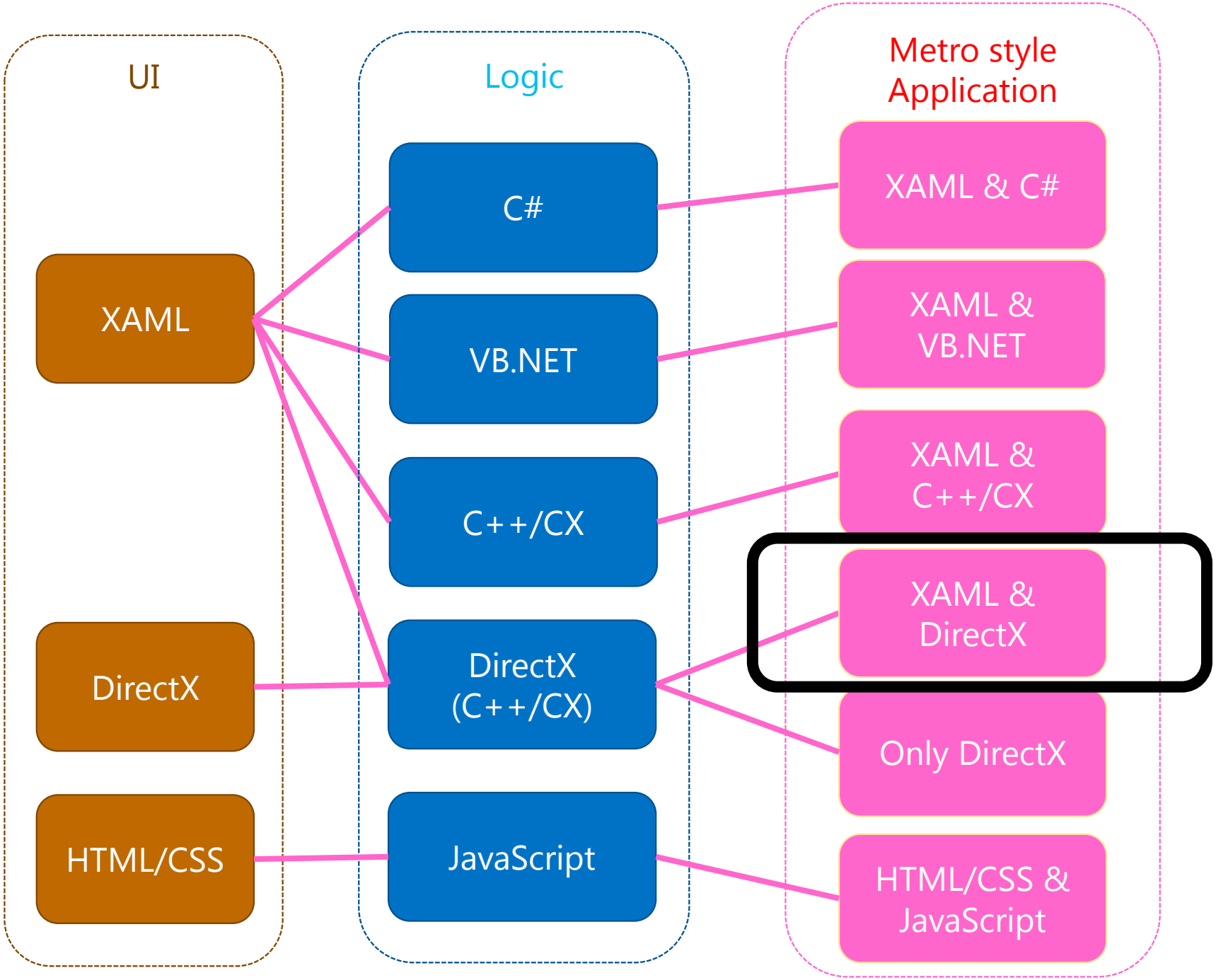
Microsoftの提供するマルチメディア系ＡＰＩ
主にゲームでの利用が有名
ＰＣ上でも、ゲーム専用機と同様の表現を可能にする

Windows8のWinRTからはDirectX 11.1 がサポート

Windows8


Metro style apps
DX
C
C++
XAML
C#
VB
HTML / CSS
JavaScript
WinRT APIs
Communication & Data
Graphics & Media
Devices & Printing
Application Model
Desktop apps
HTML
JavaScript
C
C++
C#
VB
Internet Explorer
Win32
.NET SL
Windows Kernel Services

XAML and DirectX interop

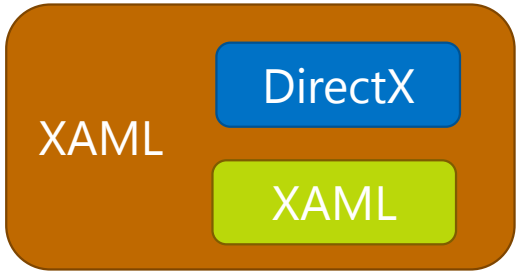

UI
Logic
Metro style Application
XAML
DirectX
HTML/CSS
C#
VB.NET
C++/CX
DirectX (C++/CX)
JavaScript
XAML & C#
XAML & VB.NET
XAML & C++/CX
XAML & DirectX
Only DirectX
HTML/CSS & JavaScript

DirectXとXAMLの組み合わせは３種類

http://msdn.microsoft.com/en-us/library/windows/apps/hh825871.aspx

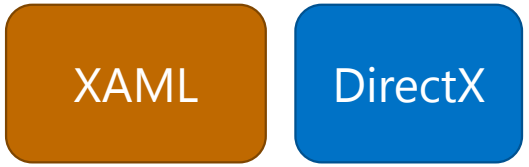
Windows::UI::Xaml::Media::Imaging::

- **SurfaceImageSource**
- **VirtualSurfaceImageSource**



Windows.UI::Xaml::Controls::

- **SwapChainBackgroundPanel**

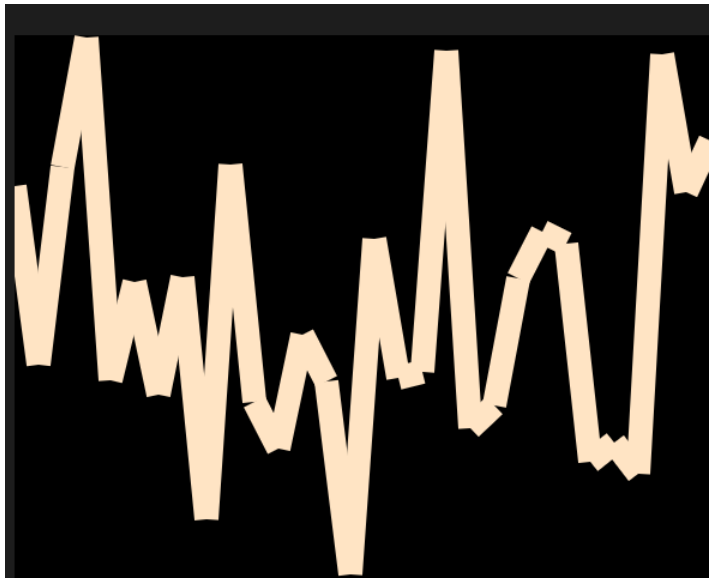


1. SurfaceImageSource

- 描画部分の流れ

  SetDevice() / BeginDraw() / EndDraw()

XAMLオブジェクトの中がDirectXの世界

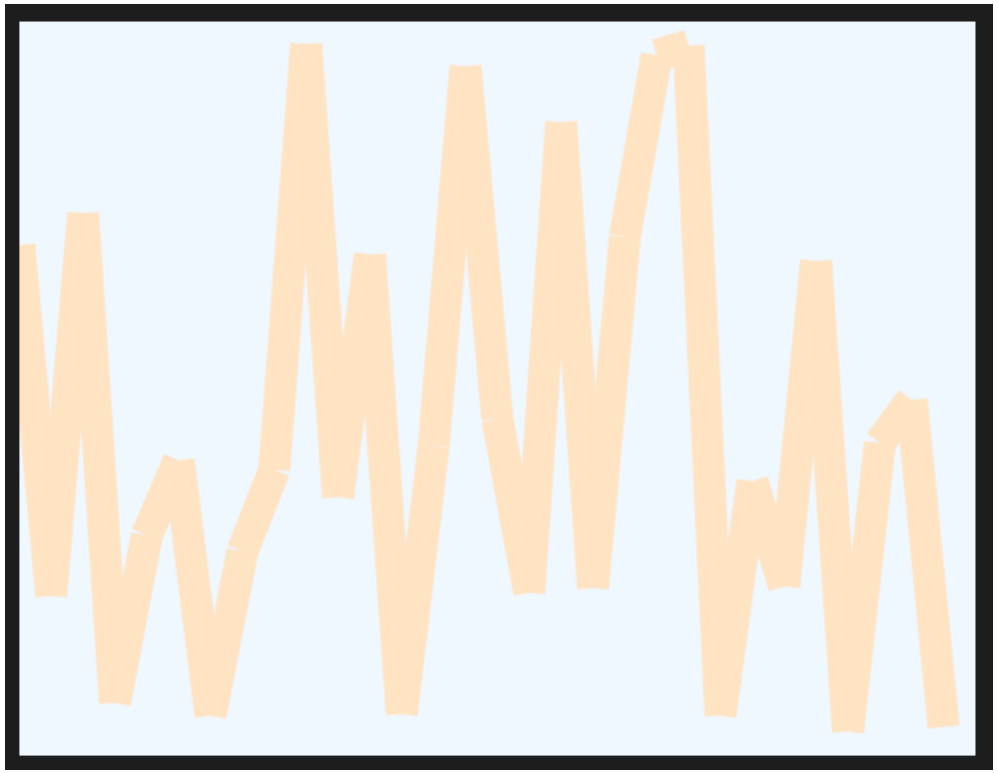


Demo

SurfaceImageSource

Windows8 RC

1. SurfaceImageSource



- XAML

```
<Rectangle x:Name="fillTarget"
        Width="1024" Height="768"
        Fill="AliceBlue"/>
```

- cpp

```
this->imageSource=ref new SurfaceImageSource(600, 460);
ImageBrush^ imageBrush = ref new ImageBrush();
this->polyLine->Fill = imageBrush;
```

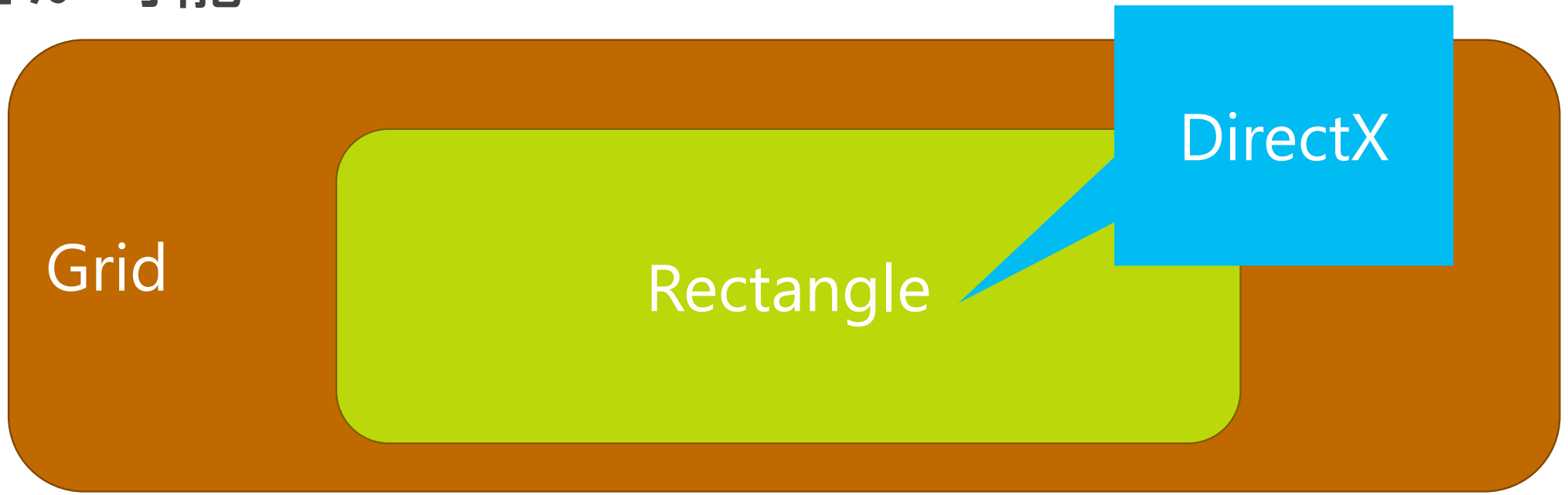
1. SurfaceImageSource

手軽に利用できる！

- 利用イメージ

XAMLでオブジェクトの設定をしておく

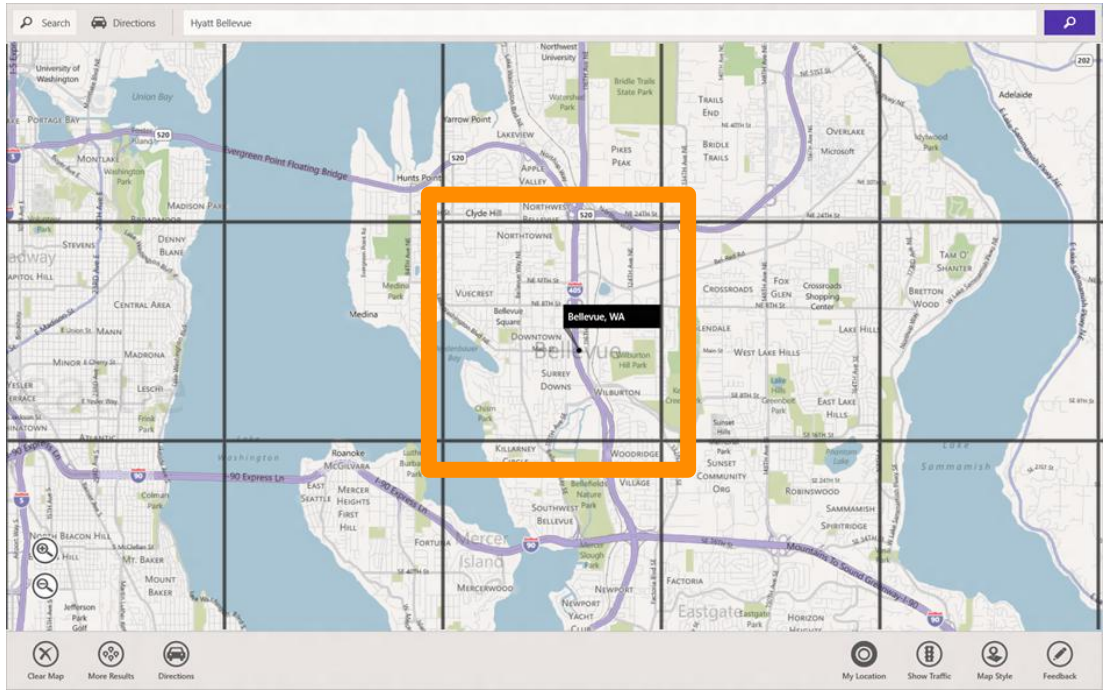
必要であればオブジェクトの中身をDirectXで記載することが可能



2. VirtualSurfaceImageSource

SurfaceImageSourceの拡張版
例えば、地図の様に広い範囲の画像に対して
今の描画領域はここ！という指定が出来る

データのやり取りは
バインディングで！



Demo
VirtualSurfaceImageSource

Windows8 RC

2. VirtualSurfaceImageSource


```
<FlipView x:Name="FlipView" ManipulationMode="TranslateInertia">
    <FlipView.ItemTemplate>
        <DataTemplate>
            <ScrollViewer ZoomMode="Disabled"
                VerticalScrollMode="Enabled" IsVerticalRailEnabled="True">
                <ScrollViewer.Background>
                    <ImageBrush ImageSource="{Binding Background}"/>
                </ScrollViewer.Background>
                <Image Source="{Binding Content}" Width="{Binding ContentWidth}"
                    Height="{Binding ContentHeight}" HorizontalAlignment="Left" />
            </ScrollViewer>
        </DataTemplate>
    </FlipView.ItemTemplate>
</FlipView>
```

2. VirtualSurfaceImageSource

- 描画の変更が起きた時のコールバックを指定

```
void HogeHoge::Initialize()
{
    Microsoft::WRL::ComPtr<IVirtualSurfaceImageSourceNative> m_image;
    auto renderer = m_document->GetRenderer();
    ComPtr<IDXGIDevice> dxgiDevice;
    renderer->GetDXGIDevice(&dxgiDevice);
    m_image->SetDevice(dxgiDevice.Get());
    m_image->RegisterForUpdatesNeeded(this);
}
```

IVirtualSurfaceUpdates CallbackNative *

2. VirtualSurfaceImageSource

- 更新したいタイルセット

```
DX::ThrowIfFailed(
        m_image->GetUpdateRects(
            drawingBounds.get(), drawingBoundsCount)
        );
```

RECT *pUpdates, …こんな形の■が
DWORD count …(例) 3 つ欲しい

- 描画

```
m_image->Invalidate(drawingBounds[i])
```

2. VirtualSurfaceImageSource

- 利用イメージ

地図など大きな画像の一部をクリッピング表示できる
オブジェクトはXAMLで宣言して、
中身はDirectXで描いたものをバインディングする

3. SwapChainBackgroundPanel

描画エリアがXAMLとDirectXで完全に分かれている
描画領域のオーバーレイ、フルスクリーン

Demo

SwapChainBackgroundPanel

Windows8 RC

3. SwapChainBackgroundPanel

- XAML

```
<SwapChainBackgroundPanel x:Class="MyApp.MainPage"
      x:Name="SwapChainPanel"
      PointerMoved="OnPointerMoved">

       <Slider x:Name="ScaleX" Grid.Row="1"
          Grid.Column="2" Minimum="0" Maximum="4“
          StepFrequency="0.01“
          ValueChanged="OnScaleXValueChanged"/>
</SwapChainBackgroundPanel>
```

3. SwapChainBackgroundPanel

- 変更された値は自前処理

```
void MainPage::OnScaleXValueChanged(Object^ sender,
RangeBaseValueChangedEventArgs^ args)
{
    SetTransformProperty(
        TransformProperty::ScaleX,
        static_cast<float>(args->NewValue)
        );
}
```

3. SwapChainBackgroundPanel

```
DX::ThrowIfFailed(
            dxgiFactory->CreateSwapChainForComposition(
                m_d3dDevice.Get(),
                &swapChainDesc,
                nullptr,
                &m_swapChain
                )
            );
ComPtr<ISwapChainBackgroundPanelNative> Panel;
reinterpret_cast<IUnknown*>(m_swapChainPanel)->QueryInterface(
            IID_PPV_ARGS(&Panel));
DX::ThrowIfFailed( Panel->SetSwapChain(m_swapChain.Get()));
```

SwapChainbackgroundPanel

3. SwapChainBackgroundPanel

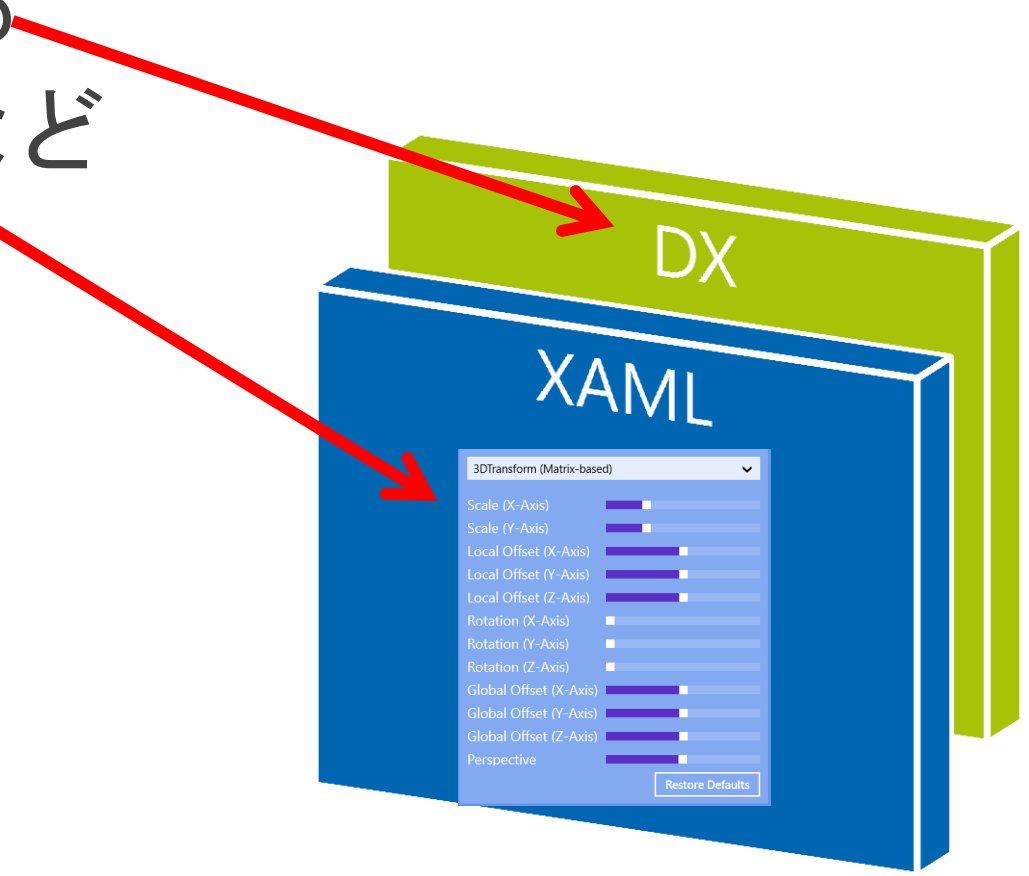
```
DX::ThrowIfFailed(
        m_3DPerspectiveTransformEffect->SetValue(
            D2D1_3DPERSPECTIVETRANSFORM_PROP_LOCAL_OFFSET,
            localOffset          ));
.....
m_d2dContext->DrawImage(
        m_3DPerspectiveTransformEffect.Get(),
        D2D1::Point2F(          screenCenter.x - (m_imageSize.width / 2.0f),
                                screenCenter.y - (m_imageSize.height / 2.0f)
            )
        );
```

いわゆるDirectXで描画

3. SwapChainBackgroundPanel

- 利用イメージ

基本的にDirectXで全て描画する
ただしコックピット的な表現など
わざわざDirectXで描画を
しなくても良いものは
XAMLにお任せする



XAMLとDirectXの連携所感

WindowsPhoneやSilverlight5の

XNA × Silverlight

に雰囲気が似ている

→DrawingSurfaceクラス相当の機能が
バージョンアップしている感じ

特徴

- SurfaceImageSource

- VirtualSurfaceImageSource

- SwapChainBackgroundPanel

Resources

- **Combining XAML and DirectX in Metro style apps**

http://channel9.msdn.com/Events/Windows-Camp/Developing-Windows-8-Metro-style-apps-in-Cpp/Building-Apps-with-Cpp-XAML-and-DirectX

- **Metro, Direct2D and XAML – Burst of Performance in Windows 8**

http://www.codertakeout.com/v/35674/52T24/Metro_Direct2D_and_XAML_Burst_of_Performance_in_Windows_8.html

- **DirectX and XAML interop (Metro style apps using C++ and DirectX)**

http://msdn.microsoft.com/en-us/library/windows/apps/hh825871.aspx

- **Windows 8 Release Preview Metro style app samples - C#, VB.NET, C++, JavaScript**

http://code.msdn.microsoft.com/windowsapps/Windows-8-Modern-Style-App-Samples

- **XAML と DirectX の統合**

http://blogs.msdn.com/b/windowsappdev_ja/archive/2012/03/22/xaml-directx.aspx

C++11 lambda

江添亮さんのブログ引用

※2009年11月時点の情報なのでその時点のドラフトを元として記載した内容とのことです。実際のC++11の仕様は各自で別途確認した方が良いとお言葉を頂きました

http://cpplover.blogspot.jp/2009/11/lambda.html

Lambda関数

- Lamdbaとは、関数オブジェクト
- 最小定義

  []{};
- 呼び出し方

  []{}();
- 文法
  - [] : 中にラムダキャプチャを記述できる
  - () : 関数の引数
  - {} : 関数の本体、コードを中に書く
  - () : 関数呼び出し

Hello World：基本

[]　：ラムダキャプチャ
()　：関数の引数
{}　：関数の本体、コード
()　：関数呼び出し

```
int main()
{
        []{ std::cout << “Hello World!” << std::endl;  }();
}
```

Hello World：変数代入

Lambdaは関数オブジェクトなので
変数に代入出来る

```
int main()
{
        // f に代入
        auto f=[]{ std::cout << “Hello World!” << std::endl;  };
        // 代入されたラムダを関数呼び出し
        f();
}
```

[]　：ラムダキャプチャ
()　：関数の引数
{}　：関数の本体、コード
()　：関数呼び出し

Hello World：引数渡し

別の関数に、引数として渡せる

[] ：ラムダキャプチャ
() ：関数の引数
{} ：関数の本体、コード
() ：関数呼び出し

```
int main()
{
	// hoge関数の呼び出し
	hoge( []{ std::cout << “Hello World!” << std::endl;  } );
}
void hoge( Func func )
{
	func();
}
```

Hello World：引数を取る

[] ：ラムダキャプチャ
() ：関数の引数
{} ：関数の本体、コード
() ：関数呼び出し

引数を受け取れる

```
int main()
{
        []( std::string const &str )                    // 引数
        { std::cout << str << std::endl;  }             // コード
        ( “Hello World!” );                             // 呼び出し
}
```

その他文法：返り値

[]　：ラムダキャプチャ
()　：関数の引数
{}　：関数の本体、コード
()　：関数呼び出し

返り値を返せる

```
int main()
{
	auto a = []{ return ( 0 );  }();		// 戻り値推測
}
int main()
{
	// 戻り値を明示的に書く
	auto a = []()->float { return ( 3.14 );  }();
}
```

ラムダキャプチャ

```
int main() {
        int a = 0, b = 0 ;
        [ a, &b ](){ a = 1 ; b = 1 ; }() ;

        // 0
        std::cout << a << std::endl ;
        // 1
        std::cout << b << std::endl ;
}
```

[]　：ラムダキャプチャ
()　：関数の引数
{}　：関数の本体、コード
()　：関数呼び出し

ラムダキャプチャ

```
int main() {
        int a = 0, b = 0 ;

        [ & ](){ a = 1 ; b = 1 ; }() ;       // 参照渡し
        [ = ](){ a = 1 ; b = 1 ; }() ;       // コピー渡し
}
```

[]　：ラムダキャプチャ
()　：関数の引数
{}　：関数の本体、コード
()　：関数呼び出し

C++11 Lambda

[] ラムダキャプチャ、スコープ内の外部変数の扱い

参照は[&]、コピーは[=]

() 関数の引数

{} 関数の本体、コードを記載

() 関数呼び出し実施

戻り値の型を明示的に記載する場合は

```
[]()->type{}();
```

クラスを書かなくても良いし、定義場所も離れない

本日の目標 達成できたでしょうか？

XAMLとDirectXの組み合わせ技術

C++11ラムダ

ありがとうございました