Bitmap for WPF

2018.04.10 更新

グレープシティ株式会社

Bitmap for WPF

目次

Bitmap for WPF

ComponentOne には、画像を読み込み、保存、変換するためのクラスライブラリである **Bitmap for WPF** が導入されています。Bitmap を使用すると、イメージファイル上でクリップ、反転、拡大/縮小、回転、またはこれらの変換の任意の組み合わせを適用することができます。さらに、Bitmap ではさまざまな画像処理ニーズに対応するためにさまざまなコンテナ形式をサポートしており、画像のピクセル形式を変更することができます。

主な特長

以下に示すように、Bitmap は単純な画像の読み込みと保存だけでなく、多くの高度な画像処理機能を提供します。

- **画像の読み込み**
  Bitmap は、BMP、PNG、JPEG、JPEG-XR、ICO などのさまざまなコンテナ形式の画像を読み込みます。ビットマップは単一フレームの TIFF と GIF もサポートしています。さらに、Bitmap では C1Bitmap の同一インスタンス内に複数の画像をを1つずつ読み込むことができます。

- **画像の保存**
  読み込み時と同様に、Bitmap に読み込まれた画像は、ストレージファイル、メモリストリーム、または別の Bitmap オブジェクトに保存することができます。さらに、Bitmap はサポートされている各コンテナ形式に対して個別の **SaveAs** メソッドを提供します。

  > Bitmap は、ICO 形式での画像の保存をサポートしていません。

- **画像の変換**
  Bitmap を使用して、画像にさまざまな変換を適用できます。例えば、変換を適用して画像を簡単にクリップ、クロップ、回転、拡大/縮小することができます。

- **Direct2D エフェクトの適用**
  Bitmap では、画像に対して Direct2D エフェクトを適用し、さまざまなアニメーションやイメージングエフェクトを作成できます。

オブジェクトモデルの概要

Bitmap には、さまざまなクラス、オブジェクト、コレクション、および関連する画像処理用のメソッドおよびプロパティを提供する、リッチなオブジェクトモデルが付属しています。以下の表は、これらのオブジェクトの一部とその主要なプロパティを示します。

C1Bitmap
プロパティ: HasImage、HasMetadata、ImagingFactory、IsDisposed、NativeBitmap、PixelFormat、PixelHeight、PixelWidth **メソッド: Import、Load、Save、Transform**
Clipper
プロパティ: ImageRect
FlipRotator
プロパティ: TransformOptions
FormatConverter
プロパティ: DestinationFormat、Palette、PaletteTranslate
Scaler
プロパティ: DestinationHeight、DestinationWidth、InterpolationMode

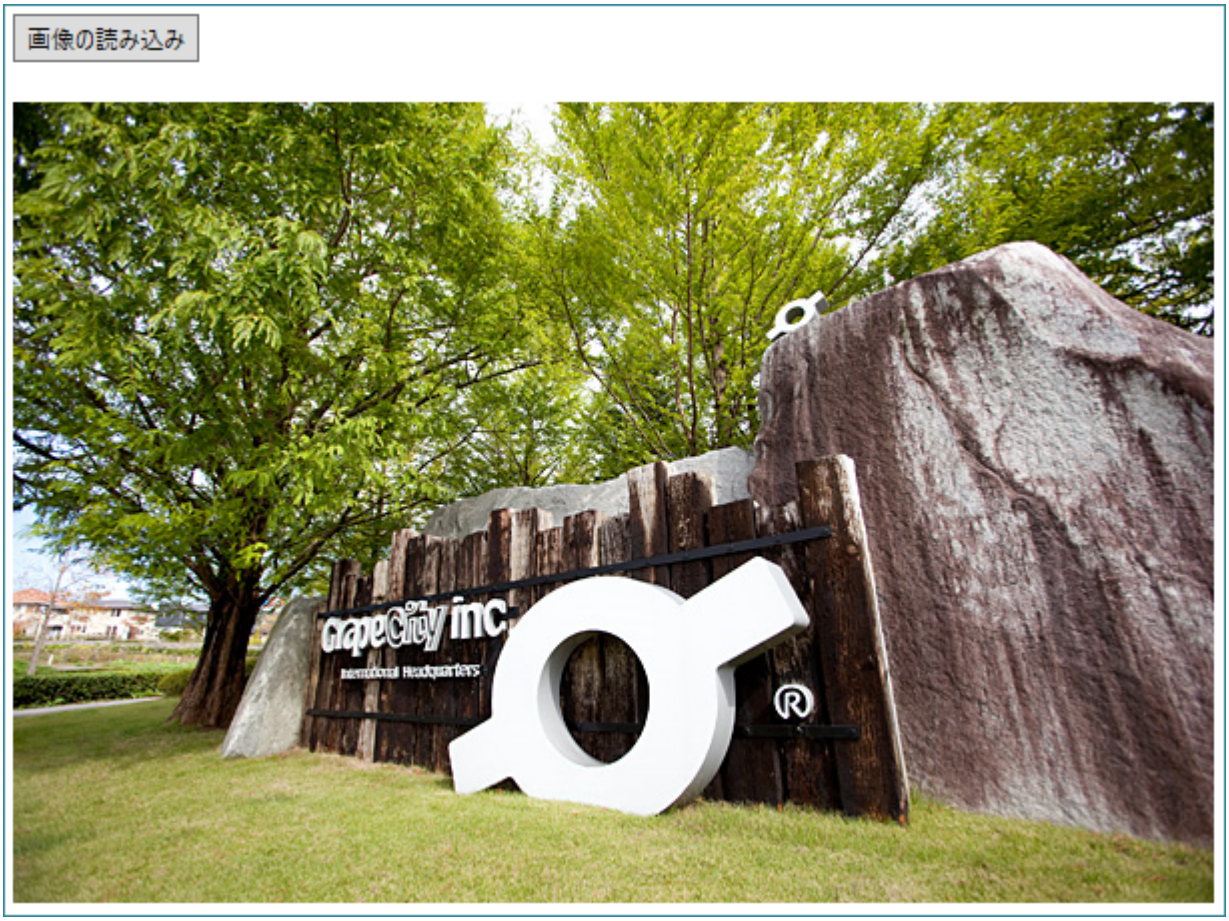
クイックスタート

このクイックスタートでは、**Bitmap** を使用して画像を読み込む方法を説明します。Visual Studio で WPF アプリケーションを作成し、サンプル画像をアプリケーションに追加し、Bitmap を使用して標準のイメージコントロールにサンプル画像を読み込むコードを追加することから始めます。このセクションに記載されたコードは、ストリームオブジェクトを介して Bitmap に画像を読み込む方法を示しています。

Bitmap を使用して標準のイメージコントロールに画像を読み込むには、以下の手順を実行してください。

1. **アプリケーションの設定とサンプル画像の追加**
2. **Bitmap を使用して画像を読み込むコードの追加**

以下の画像は、アプリケーションがボタンクリックによって Bitmap 内に読み込まれた画像を表示する例を示しています。



手順1：アプリケーションの設定とサンプル画像の追加

1. Visual Studioで **WPF** アプリケーションを作成します。
2. アプリケーションに以下の参照を追加します。
   - C1.WPF.4
   - C1.WPF.Bitmap.4
   - C1.WPF.Automation.4
   - C1.WPF.DX.4
3. 「**ソリューションエクスプローラ**」内で、プロジェクト名を右クリックして、[**追加**]→[**新しいフォルダ**]を選択し、「**Resources**」という名前を付けます。
4. Visual Studioで、サンプル画像を Resources フォルダに追加し、プロパティウィンドウにて「**ビルド アクション**」プロパティを「**埋め込みリソース**」に設定します。
5. クリックされたらサンプル画像を読み込むための標準の **Button** コントロールと、サンプル画像を MainWindow 上に表示するための **image** コントロールを追加します。

6. XAML ビューにて、ボタンの **Content** プロパティに適切なテキストを設定します。

手順2：Bitmap を使用して画像を読み込むコードの追加

1. コードビューに切り替えて、以下の import ステートメントを追加します。
   - **Visual Basic**

```
Imports C1.WPF
Imports C1.WPF.Bitmap
Imports C1.Util.DX
Imports System.Reflection
Imports System.IO
```

   - **C#**

```
using C1.WPF;
using C1.WPF.Bitmap;
using C1.Util.DX;
using System.Reflection;
using System.IO;
```

2. MainWindow クラスのビットマップを初期化します。
   - **Visual Basic**

```
'ビットマップを初期化します
Private bitmap As C1Bitmap

Public Sub New()

    ' デザイナーによって必要とされる呼び出し
    InitializeComponent()
    bitmap = New C1Bitmap()
    ' InitializeComponent()呼び出しの後に任意の初期化を追加します。

End Sub
```

   - **C#**

```
//ビットマップを初期化します
C1Bitmap bitmap = new C1Bitmap();
```

3. ボタンのクリックイベントをサブスクライブし、ストリームオブジェクトからビットマップにサンプル画像を読み込むために、以下のコードを追加します。
   - **Visual Basic**

```
'ボタンをクリックした際に、画像をストリームにロードします
Private Sub Btn_Load_Click(sender As Object, e As RoutedEventArgs) _
    Handles Btn_Load.Click
    Dim t As Type = Me.GetType
    Dim asm As Assembly = t.Assembly
    Dim stream As Stream =
        asm.GetManifestResourceStream(t, "GrapeCity.png")
    bitmap.Load(stream,
                New FormatConverter(PixelFormat.Format32bppPBGRA))
    UpdateImage()
End Sub
```

   - **C#**

```
//ボタンをクリックした際に、画像をストリームにロードします
private void Button_Click(object sender, RoutedEventArgs e)
{
    Assembly asm = typeof(MainWindow).Assembly;
    using (Stream stream =
        asm.GetManifestResourceStream("Bitmap.Resources.GrapeCity.png"))
    {
        bitmap.Load(stream,
            new FormatConverter(PixelFormat.Format32bppPBGRA));
    }

    UpdateImage();
}
```

4. サンプル画像を表示する UpdateImage メソッドを定義するために、以下のコードを追加します。
    - **Visual Basic**

```
'ロードされた画像を表示します
Private Sub UpdateImage()
    Me.image.Source = bitmap.ToWriteableBitmap()
    Me.image.Width = bitmap.PixelWidth
    Me.image.Height = bitmap.PixelHeight
End Sub
```

    - **C#**

```
//ロードされた画像を表示します
private void UpdateImage()
{
    this.image.Source = bitmap.ToWriteableBitmap();
    this.image.Width = bitmap.PixelWidth;
    this.image.Height = bitmap.PixelHeight;
}
```

機能

Bitmap は、ユーザーのプロセスを支援し画像を処理するための多くの機能をサポートしています。

画像の読み込みおよび保存
　読み込みおよび保存をコード内に実装する方法を学びます。
変換の適用
　さまざまな変換をコード内に適用する方法を学びます。

画像の読み込みおよび保存

Bitmap には画像を読み込むためのさまざまな方法があります。**C1Bitmap** クラスは、ファイルやメモリストリームなどのさまざまなソースから画像を読み込むための、いくつかの **Load** オーバーロードメソッドを提供します。また、画像のメタデータを読み込み、画像のサイズ、ピクセル形式、または解像度（1インチあたりのドット数）を決定するために使用できます。

読み込まれた画像は、ファイルまたはメモリストリームに保存することができます。C1Bitmap クラスは、コンテナ形式を引数として受け入れる一般的な **Save** メソッドを提供します。C1Bitmap はまた、サポートされている各コンテナ形式に対して個別の **SaveAs** メソッドを提供します。

以下のコードは、ボタンのクリック時に任意の画像を読み込んで保存する方法を示しています。コード例では、**OpenFileDialog** および **SaveFileDialog** を使用して、ユーザーのマシンの任意の場所に保存されている画像ファイルにアクセスします。ストリームオブジェクトから画像を読み込む方法については、「クイックスタート」を参照してください。

- **Visual Basic**

```
Partial Public Class MainWindow
    Inherits Window

    'ビットマップのグローバル変数を定義します
    Dim bitmap As New C1Bitmap()

    'ボタンクリック時にピクチャボックスに任意の画像をロードするイベント
    Private Sub Button_Click(sender As Object, e As RoutedEventArgs)
        Dim ofd = New OpenFileDialog()
        ofd.Filter = "Image Files|*.ico;*.bmp;*.gif;" +
                "*.png;*.jpg;*.jpeg;*.jxr;*.tif;*.tiff"

        If ofd.ShowDialog().Value Then
            bitmap.Load(ofd.FileName,
                        New FormatConverter(PixelFormat.Format32bppPBGRA))
            Image.Source = bitmap.ToWriteableBitmap()
            Image.Width = bitmap.PixelWidth
            Image.Height = bitmap.PixelHeight
        End If
    End Sub

    'ボタンクリック時にピクチャーボックスに表示される画像をファイルとして保存するイベント
    Private Sub Button_Click_1(sender As Object, e As RoutedEventArgs)
        Dim sfd As New SaveFileDialog()
        sfd.Filter = "Png Files (*.png)|*.png"
        sfd.CheckPathExists = True

        If sfd.ShowDialog().Value Then
            bitmap.Save(sfd.FileName, ContainerFormat.Png)
        End If
    End Sub
End Class
```

- **C#**

```
public partial class MainWindow : Window
{
    //ビットマップのグローバル変数を定義します
    C1Bitmap bitmap;

    public MainWindow()
    {
        InitializeComponent();

        //ビットマップを初期化します
        bitmap = new C1Bitmap();
    }

    //ボタンクリック時にピクチャボックスに任意の画像をロードするイベント
    private void Button_Click(object sender, RoutedEventArgs e)
    {
        var ofd = new OpenFileDialog();
        ofd.Filter = "Image Files|*.ico;*.bmp;*.gif;" +
            "*.png;*.jpg;*.jpeg;*.jxr;*.tif;*.tiff";

        if (ofd.ShowDialog().Value)
        {
            bitmap.Load(ofd.FileName, new
                FormatConverter(PixelFormat.Format32bppPBGRA));
            image.Source = bitmap.ToWriteableBitmap();
            image.Width = bitmap.PixelWidth;
            image.Height = bitmap.PixelHeight;
        }
    }

    //ボタンクリック時にピクチャーボックスに表示される画像をファイルとして保存するイベント
    private void Button_Click_1(object sender, RoutedEventArgs e)
    {
        SaveFileDialog sfd = new SaveFileDialog();
        sfd.Filter = "Png Files (*.png)|*.png";
        sfd.CheckPathExists = true;

         if (sfd.ShowDialog().Value)
         {
             bitmap.Save(sfd.FileName, ContainerFormat.Png);
         }

    }
}
```

変換の適用

Bitmap は、クリッピング、反転、拡大/縮小、回転など、画像に対してさまざまな変形を適用できます。これらの変換とその実装方法について学びます。

画像のクリッピング
　コード内にクリッピング処理を実装する方法を学びます。
画像の反転
　コード内に反転処理を実装する方法を学びます。
画像の回転
　コード内に回転処理を実装する方法を学びます。
画像の拡大/縮小
　コード内に拡大/縮小処理を実装する方法を学びます。

画像のクリッピング

2次元画像において、選択されたフレーム領域の境界内のピクセルの選択的レンダリングを提供するにあたっては、クリッピングが必須の要件となります。Bitmap は、クリッパー変換を使用してソース画像をクリップし、画像全体の一部を読み込むことができます。

以下の画像は、クリッピング機能を示しています。

コード内で Bitmap を使用してイメージをクリップするには、次の手順を実行します。

1. 次の import ステートメントを追加します。
   - **Visual Basic**

```
Imports System.IO
```

   - **C#**

```
using System.IO;
```

2. Form1 クラスにて、矩形と点をグローバル変数として初期化します。
   - **Visual Basic**

```
'変数を初期化します
Dim start As Point
Dim selection As Rect
Dim dragHelper As C1DragHelper
```

   - **C#**

```
//変数を初期化します
Point start;
Rect selection;
C1DragHelper dragHelper;
```

3. MainWindow コンストラクタ内で、ドラッグジェスチャーを制御する変数を初期化し、ドラッグイベントをサブスクライブしま

す。

- **Visual Basic**

```
'ドラッグジェスチャーのためにドラッグヘルパーを初期化します
dragHelper = New C1DragHelper(image)

'ドラッグイベントを登録します
dragHelper.DragDelta += dragHelper_DragDelta()
```

- **C#**

```
//ドラッグジェスチャーのためにドラッグヘルパーを初期化します
dragHelper = new C1DragHelper(image);

//ドラッグイベントを登録します
dragHelper.DragDelta += dragHelper_DragDelta;
```

4. クリッパー変換を適用するために、以下のコードを追加します。

- **Visual Basic**

```
'変換を適用するためのTransformメソッド
Private Sub ApplyTransform(t As BaseTransform)
    Dim newBitmap = bitmap.Transform(t)
    bitmap.Dispose()
    bitmap = newBitmap
    UpdateImage()
End Sub

'ボタンクリック時にクリッパー変換を適用するイベント
Private Sub Button_Click(sender As Object, e As RoutedEventArgs)
    Dim cropRect = DirectCast(selection, RectD).Round()
    ApplyTransform(New Clipper(New ImageRect(cropRect)))
End Sub
```

- **C#**

```
//変換を適用するメソッド
void ApplyTransform(BaseTransform t)
{
    var newBitmap = bitmap.Transform(t);
    bitmap.Dispose();
    bitmap = newBitmap;
    UpdateImage();
}

//ボタンクリック時にクリッパー変換を適用するイベント
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    var cropRect = ((RectD)selection).Round();
    ApplyTransform(new Clipper(new ImageRect(cropRect)));
}
```

5. クリップされる画像の一部分を選択するために、以下のコードを追加します。

- **Visual Basic**

```
'画像から画像の一部を選択するイベント
Private Sub dragHelper_DragDelta(sender As Object, e As C1DragDeltaEventArgs)
    Dim pos = e.GetPosition(image)
    pos = New Point(Math.Max(0, Math.Min(pos.X, bitmap.PixelWidth)),
                    Math.Max(0, Math.Min(pos.Y, bitmap.PixelHeight)))

    selection = New Rect(Math.Round(Math.Min(start.X, pos.X)),
                         Math.Round(Math.Min(start.Y, pos.Y)),
                         Math.Round(Math.Abs(start.X - pos.X)),
                         Math.Round(Math.Abs(start.Y - pos.Y)))
End Sub

Private Sub image_MouseLeftButtonDown(sender As Object, e As MouseButtonEventArgs) _
    Handles image.MouseLeftButtonDown
    MyBase.OnMouseLeftButtonDown(e)
    Dim pos = e.GetPosition(image)
    start = New Point(Math.Max(0, Math.Min(pos.X, bitmap.PixelWidth)),
                      Math.Max(0, Math.Min(pos.Y, bitmap.PixelHeight)))
```

```
End Sub

Private Sub image_MouseLeftButtonUp(sender As Object, e As MouseButtonEventArgs) _
    Handles image.MouseLeftButtonUp
    MyBase.OnMouseLeftButtonUp(e)
    Dim pt = e.GetPosition(image)
    If Math.Abs(pt.X - start.X) < 4 AndAlso Math.Abs(pt.Y - start.Y) < 4 Then
        selection = New Rect(0, 0, bitmap.PixelWidth, bitmap.PixelHeight)
    End If
End Sub
```

- **C#**

```
//画像から画像の一部を選択するイベント
private void image_MouseLeftButtonDown(object sender,
    System.Windows.Input.MouseButtonEventArgs e)
{
   base.OnMouseLeftButtonDown(e);
    var pos = e.GetPosition(image);
    start = new Point(
    Math.Max(0, Math.Min(pos.X, bitmap.PixelWidth)),
    Math.Max(0, Math.Min(pos.Y, bitmap.PixelHeight)));
}

private void image_MouseLeftButtonUp(object sender,
    System.Windows.Input.MouseButtonEventArgs e)
{
    base.OnMouseLeftButtonUp(e);
    var pt = e.GetPosition(image);
    if (Math.Abs(pt.X - start.X) < 4 && Math.Abs(pt.Y - start.Y) < 4)
    {
        selection = new Rect(0, 0, bitmap.PixelWidth, bitmap.PixelHeight);
    }
}

void dragHelper_DragDelta(object sender, C1DragDeltaEventArgs e)
{
    var pos = e.GetPosition(image);
    pos = new Point(Math.Max(0, Math.Min(pos.X, bitmap.PixelWidth)),
        Math.Max(0, Math.Min(pos.Y, bitmap.PixelHeight)));

    selection = new Rect(
        Math.Round(Math.Min(start.X, pos.X)),
        Math.Round(Math.Min(start.Y, pos.Y)),
        Math.Round(Math.Abs(start.X - pos.X)),
        Math.Round(Math.Abs(start.Y - pos.Y)));
}
```

6. F5 キーを押してアプリケーションを実行し、「画像の読み込み」ボタンをクリックして画像を読み込みます。
7. 画像の一部をマウスで選択し、「画像のクリッピング」'ボタンをクリックすると、選択された部分が切り取られます。

画像の反転

Bitmap は、画像を縦方向または横方向に反転できます。Bitmap を使用して反転したイメージを生成するには、**FlipRotator** クラスの **TransformOptions** プロパティを設定します。**TransformOptions** プロパティは、**TransformOptions** 列挙値によって設定できます。

以下の画像は、横方向に反転した画像を示しています。

以下のコードは、ボタンのクリック時に画像を上下または左右に反転させる方法を示しています。この例では、「クイックスタート」セクションで作成したサンプルを使用します。

- **Visual Basic**

```
Private Sub ApplyTransform(t As BaseTransform)
    Dim newBitmap = bitmap.Transform(t)
    bitmap.Dispose()
    bitmap = newBitmap
    UpdateImage()
End Sub

'ボタンクリック時に画像を縦方向に反転させるイベント
Private Sub Button_Click_1(sender As Object, e As RoutedEventArgs)
    ApplyTransform(New FlipRotator(TransformOptions.FlipVertical))
End Sub

'ボタンクリック時に画像を横方向に反転させるイベント
Private Sub Button_Click_2(sender As Object, e As RoutedEventArgs)
    ApplyTransform(New FlipRotator(TransformOptions.FlipHorizontal))
End Sub
```

- **C#**

```
void ApplyTransform(BaseTransform t)
{
    var newBitmap = bitmap.Transform(t);
    bitmap.Dispose();
    bitmap = newBitmap;
    UpdateImage();
}
```

```
//ボタンクリック時に画像を縦方向に反転させるイベント
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    ApplyTransform(new FlipRotator(TransformOptions.FlipVertical));
}

//ボタンクリック時に画像を横方向に反転させるイベント
private void Button_Click_2(object sender, RoutedEventArgs e)
{
    ApplyTransform(new FlipRotator(TransformOptions.FlipHorizontal));
}
```

画像の回転

Bitmap では、画像を時計回りに90度、180度、270度に回転できます。Bitmap を使用してイメージを回転するには、**FlipRotator** クラスの **TransformOptions** プロパティを設定します。TransformOption プロパティは、**TransformOptions** 列挙値によって設定できます。

以下の画像は、時計回りに180度回転した画像を示しています。

以下のコードは、ボタンのクリック時に時計回りと反時計回りの方向に画像を回転させる方法を示しています。この例では、「クイックスタート」セクションで作成したサンプルを使用します。

- **Visual Basic**

```
Private Sub ApplyTransform(t As BaseTransform)
    Dim newBitmap = bitmap.Transform(t)
    bitmap.Dispose()
    bitmap = newBitmap
    UpdateImage()
```

```
End Sub

'ボタンをクリック時に時計回りに画像を回転させるイベント
Private Sub Button_Click_1(sender As Object, e As RoutedEventArgs)
    ApplyTransform(New FlipRotator(TransformOptions.Rotate180))
End Sub

'ボタンクリック時に反時計回りに画像を回転させるイベント
Private Sub Button_Click_2(sender As Object, e As RoutedEventArgs)
    ApplyTransform(New FlipRotator(TransformOptions.Rotate270))
End Sub
```

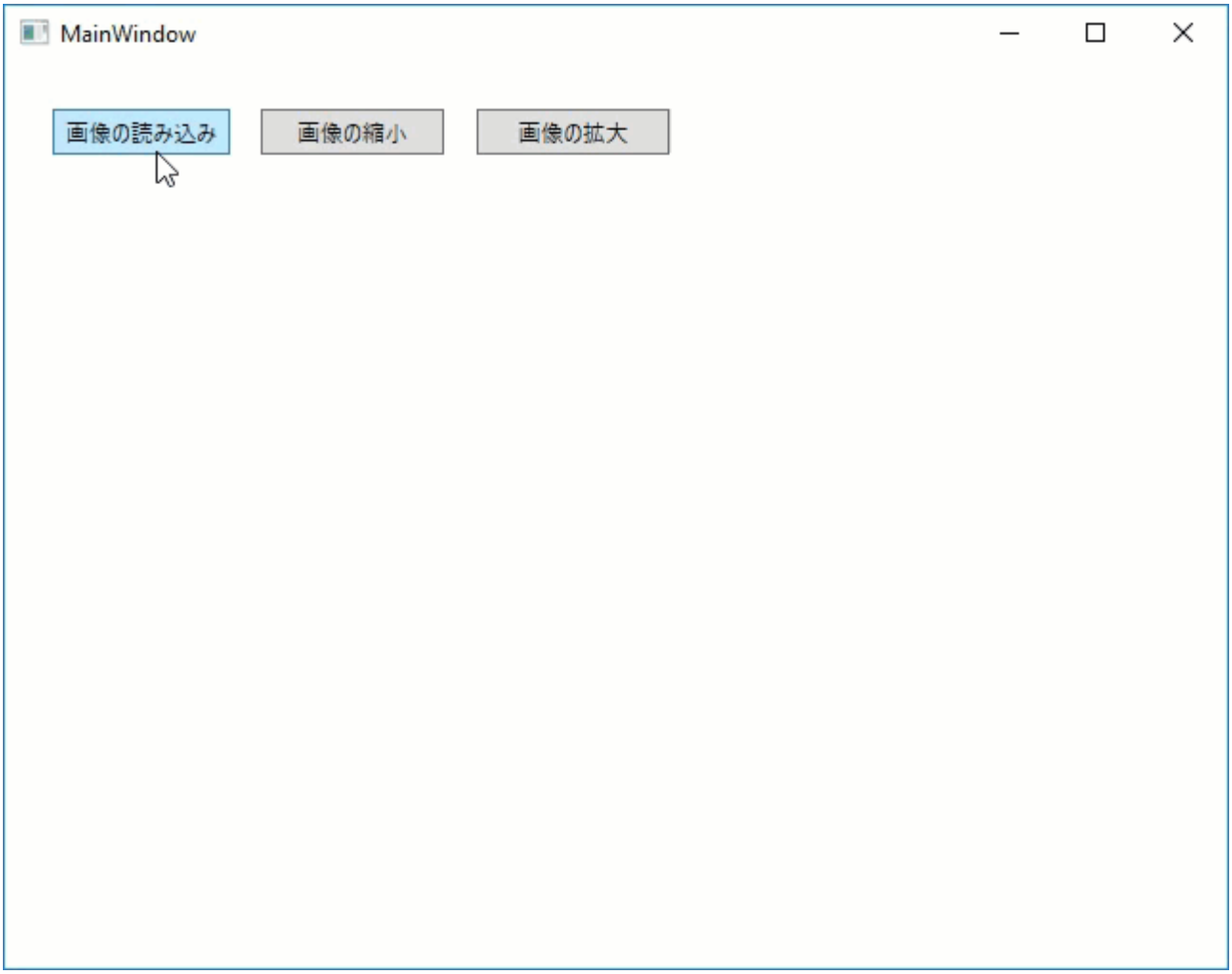
- **C#**

```
void ApplyTransform(BaseTransform t)
{
    var newBitmap = bitmap.Transform(t);
    bitmap.Dispose();
    bitmap = newBitmap;
    UpdateImage();
}

//ボタンをクリック時に画像を時計回りに回転させるイベント
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    ApplyTransform(new FlipRotator(TransformOptions.Rotate180));
}

//ボタンクリック時に画像を反時計回りに回転させるイベント
private void Button_Click_2(object sender, RoutedEventArgs e)
{
    ApplyTransform(new FlipRotator(TransformOptions.Rotate270));
}
```

画像の拡大/縮小

画像の拡大/縮小（スケーリング）は、画像のリサイズ（サイズの増減）に伴う画像処理の重要な要件です。Bitmap では、**Scaler** クラスの **InterpolationMode** プロパティを使用して、画像の拡大/縮小を行うことができます。

以下の画像は、拡大/縮小機能を示しています。



以下のコードは、ボタンクリック時の画像の拡大/縮小を示しています。この例では、「クイックスタート」セクションで作成したサンプルを使用します。

- **Visual Basic**

```
Private Sub ApplyTransform(t As BaseTransform)
    Dim newBitmap = bitmap.Transform(t)
    bitmap.Dispose()
    bitmap = newBitmap
    UpdateImage()
End Sub

'ボタンをクリックする時に画像の拡大を行うイベント
Private Sub Button_Click_1(sender As Object, e As RoutedEventArgs)
    Dim px As Integer = CInt(bitmap.PixelWidth * 0.625F + 0.5F)
    Dim py As Integer = CInt(bitmap.PixelHeight * 0.625F + 0.5F)
    If px > 0 AndAlso py > 0 Then
        ApplyTransform(New Scaler(px, py, InterpolationMode.HighQualityCubic))
    End If
End Sub

'ボタンをクリックする時に画像の縮小を行うイベント
Private Sub Button_Click_2(sender As Object, e As RoutedEventArgs)
    Dim px As Integer = CInt(bitmap.PixelWidth * 1.6F + 0.5F)
    Dim py As Integer = CInt(bitmap.PixelHeight * 1.6F + 0.5F)
    ApplyTransform(New Scaler(px, py, InterpolationMode.HighQualityCubic))
End Sub
```

- **C#**

```
void ApplyTransform(BaseTransform t)
{
    var newBitmap = bitmap.Transform(t);
    bitmap.Dispose();
    bitmap = newBitmap;
    UpdateImage();
}

//ボタンをクリックする時に画像の拡大を行うイベント
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    int px = (int)(bitmap.PixelWidth * 0.625f + 0.5f);
    int py = (int)(bitmap.PixelHeight * 0.625f + 0.5f);
    if (px > 0 && py > 0)
    {
        ApplyTransform(new Scaler(px, py, InterpolationMode.HighQualityCubic));
    }
}

//ボタンをクリックする時に画像の縮小を行うイベント
private void Button_Click_2(object sender, RoutedEventArgs e)
{
    int px = (int)(bitmap.PixelWidth * 1.6f + 0.5f);
    int py = (int)(bitmap.PixelHeight * 1.6f + 0.5f);
    ApplyTransform(new Scaler(px, py, InterpolationMode.HighQualityCubic));
}
```

Bitmap の操作

「Bitmap の操作」セクションは、ユーザーの皆様が Bitmap コントロールの基礎と機能および一般的な使用方法を理解していることを前提としています。次のセクションでは、Bitmap で提供されている補助機能について説明します。

Direct2D エフェクトの適用
: Direct2D エフェクトをコードで適用する方法を説明します。

Direct2D エフェクトの適用

Direct2D は、Microsoft によって設計された 2D グラフィック API で、画像を操作するための広範な組み込みおよびカスタムのエフェクトが提供されています。この API を使用すると、ビットマップ、2D ジオメトリ、テキストの高品質で高速なレンダリングが可能です。

Bitmap では、Direct2D のエフェクトを使用したり、画像にエフェクトを適用することができます。Bitmap を使用して適用できる画像エフェクトを次に一覧します。

- ブラー(ガウス)
- シャープネス
- 水平スミア
- シャドウ
- ディスプレイスメントマップ
- エンボス
- エッジ検出
- セピア

これらのエフェクトから 1 つを選んで画像に適用してみましょう。次の図は、Bitmap で Direct2D を使用する例として、組み込み 2D エフェクトの 1 つ、シャドウを示しています。

コードで、Bitmap が Direct2D ビットマップに変換されます。次に Direct2D を使用して画像を操作し、Direct3D API との相互運

用によって組み込みエフェクト、シャドウが適用されます。すべての操作が完了したら、画像が Direct2D ビットマップから C1Bitmap にロードし直されます。

画像にシャドウエフェクトを適用するには、**C1.Util.DX.Direct2D.Effects** 名前空間のメンバクラスである **Shadow**、**AffineTransform2D**、**Composite** のプロパティを使用します。

以下の手順は、2D シャドウエフェクトを画像に適用する方法を示します。この例では、「クイックスタート」で作成したサンプルを使用します。

1. 次の名前空間を追加します。
   - **Visual Basic**

```
Imports D2D = C1.Util.DX.Direct2D
Imports D3D = C1.Util.DX.Direct3D11
Imports DW = C1.Util.DX.DirectWrite
Imports DXGI = C1.Util.DX.DXGI
Imports C1.Util.DX
```

   - **C#**

```
using D2D = C1.Util.DX.Direct2D;
using D3D = C1.Util.DX.Direct3D11;
using DW = C1.Util.DX.DirectWrite;
using DXGI = C1.Util.DX.DXGI;
using C1.Util.DX;
```

2. 次のクラスオブジェクトを作成します。
   - **Visual Basic**

```
Private bitmap As C1Bitmap

' 装置独立リソース
Private d2dFactory As D2D.Factory2
Private dwFactory As DW.Factory

' 装置リソース
Private dxgiDevice As DXGI.Device
Private d2dContext As D2D.DeviceContext1

' Direct2Dの組み込み効果
Private shadow As D2D.Effects.Shadow
Private affineTransform As D2D.Effects.AffineTransform2D
Private composite As D2D.Effects.Composite
```

   - **C#**

```
C1Bitmap bitmap;

// 装置独立リソース
D2D.Factory2 d2dFactory;
DW.Factory dwFactory;

// 装置リソース
DXGI.Device dxgiDevice;
D2D.DeviceContext1 d2dContext;

// Direct2Dの組み込み効果
D2D.Effects.Shadow shadow;
D2D.Effects.AffineTransform2D affineTransform;
D2D.Effects.Composite composite;
```

3. 次の整数定数と列挙を宣言します。
   - **Visual Basic**

```
Const marginLT As Integer = 20
Const marginRB As Integer = 36

Public Enum ImageEffect
    Original
    Shadow
End Enum
```

   - **C#**

```
const int marginLT = 20;
```

```
const int marginRB = 36;

public enum ImageEffect
{
    Original,
    Shadow
}
```

4. ストリームを使用して画像を C1Bitmap にロードします。詳細については、「クイックスタート」を参照してください。
5. 次のコードを追加して、リソースと画像ソースを作成し、画像ソースを画像と関連付けます。
    - **Visual Basic**

```
' Direct2DおよびDirectWriteファクトリを作成します
d2dFactory = D2D.Factory2.Create(D2D.FactoryType.SingleThreaded)
dwFactory = DW.Factory.Create(DW.FactoryType.[Shared])

' GPUリソースを作成します
CreateDeviceResources()
```

    - **C#**

```
// Direct2DおよびDirectWriteファクトリを作成します
d2dFactory = D2D.Factory2.Create(D2D.FactoryType.SingleThreaded);
dwFactory = DW.Factory.Create(DW.FactoryType.Shared);

// GPUリソースを作成します
CreateDeviceResources();
```

6. 次のコードを追加して、2D シャドウエフェクトを適用します。
    - **Visual Basic**

```
Private Sub Button_Click(sender As Object, e As RoutedEventArgs)

    UpdateImageSource(ImageEffect.Shadow)
End Sub

Private Sub CreateDeviceResources()
    Dim actualLevel As D3D.FeatureLevel
    Dim d3dContext As D3D.DeviceContext = Nothing
    Dim d3dDevice = New D3D.Device(IntPtr.Zero)
    Dim result = HResult.Ok
    For i As Integer = 0 To 1
        ' ハードウェアが利用できない場合はWARPを使用します
        Dim dt = If(i = 0, D3D.DriverType.Hardware, D3D.DriverType.Warp)
        result = D3D.D3D11.CreateDevice(Nothing, dt, IntPtr.Zero, _
                                        D3D.DeviceCreationFlags.BgraSupport Or _
                                        D3D.DeviceCreationFlags.SingleThreaded, _
                                        Nothing, 0, _
            D3D.D3D11.SdkVersion, d3dDevice, actualLevel, d3dContext)
        If result.Code <> CInt(&H887A0004UI) Then
            ' DXGI_ERROR_UNSUPPORTED
            Exit For
        End If
    Next
    result.CheckError()
    d3dContext.Dispose()

    ' DXGI装置を格納します(アプリケーションが中断されているときにトリミングするため)
    dxgiDevice = d3dDevice.QueryInterface(Of DXGI.Device)()
    d3dDevice.Dispose()

    ' RenderTargetを作成します(Direct2D描画用のDeviceContext)
    Dim d2dDevice = D2D.Device1.Create(d2dFactory, dxgiDevice)
    Dim rt = D2D.DeviceContext1.Create(d2dDevice, D2D.DeviceContextOptions.None)
    d2dDevice.Dispose()
    rt.SetUnitMode(D2D.UnitMode.Pixels)
    d2dContext = rt

    ' 組み込みの効果を作成します
    shadow = D2D.Effects.Shadow.Create(rt)
```

```
        affineTransform = D2D.Effects.AffineTransform2D.Create(rt)
        composite = D2D.Effects.Composite.Create(rt)
    End Sub

    Private Sub UpdateImageSource(imageEffect__1 As ImageEffect)
        ' ソース画像の範囲外のピクセルを変更する可能性のある交換があるため、
        ' これらのピクセルを表示するためのマージンが必要です
        Dim targetOffset = New Point2F(marginLT, marginLT)
        Dim w As Integer = bitmap.PixelWidth + marginLT + marginRB
        Dim h As Integer = bitmap.PixelHeight + marginLT + marginRB

        ' レンダー対象オブジェクト
        Dim rt = d2dContext

        ' 対象Direct2Dビットマップを作成します
        Dim bpTarget = New  _
            D2D.BitmapProperties1(New D2D.PixelFormat(DXGI.Format.B8G8R8A8_UNorm, _
                                  D2D.AlphaMode.Premultiplied), _
                    CSng(bitmap.DpiX), CSng(bitmap.DpiY), D2D.BitmapOptions.Target Or _
                D2D.BitmapOptions.CannotDraw)
        Dim targetBmp = D2D.Bitmap1.Create(rt, New Size2L(w, h), bpTarget)

        ' 対象ビットマップをレンダー対象に関連付けます
        rt.SetTarget(targetBmp)

        ' 描画を開始します
        rt.BeginDraw()

        ' 対象ビットマップをクリアします
        rt.Clear(Nothing)

        ' C1Bitmap画像をDirect2D画像に変換します
        Dim d2dBitmap = bitmap.ToD2DBitmap1(rt, D2D.BitmapOptions.None)

        Select Case imageEffect__1
            Case ImageEffect.Original
                rt.DrawImage(d2dBitmap, targetOffset)
                Exit Select
            Case ImageEffect.Shadow
                rt.DrawImage(ApplyShadow(d2dBitmap), targetOffset)
                Exit Select
        End Select
        d2dBitmap.Dispose()

        If Not rt.EndDraw(True) Then
            targetBmp.Dispose()

            ' 旧GPU装置が削除された場合、装置リソースを再作成してみてください
            DiscardDeviceResources()
            CreateDeviceResources()
            Return
        End If

        '  対象ビットマップをデタッチします
        rt.SetTarget(Nothing)

        '  一時的なC1Bitmapオブジェクトを作成します
        Dim outBitmap = New C1Bitmap(bitmap.ImagingFactory)

        ' Direct2D対象ビットマップからC1Bitmapに画像をインポートします
        outBitmap.Import(targetBmp, rt, New RectL(w, h))
        targetBmp.Dispose()

        ' C1BitmapをWriteableBitmapに変換し、イメージソースとして使用します
        image.Source = outBitmap.ToWriteableBitmap()
```

```
    outBitmap.Dispose()

End Sub

Private Sub DiscardDeviceResources()
    shadow.Dispose()
    affineTransform.Dispose()
    composite.Dispose()
    dxgiDevice.Dispose()
    d2dContext.Dispose()
End Sub

Private Function ApplyShadow(bitmap As D2D.Bitmap1) As D2D.Effect
    shadow.SetInput(0, bitmap)
    shadow.BlurStandardDeviation = 5.0F
    affineTransform.SetInputEffect(0, shadow)
    affineTransform.TransformMatrix = Matrix3x2.Translation(20.0F, 20.0F)
    composite.SetInputEffect(0, affineTransform)
    composite.SetInput(1, bitmap)
    Return composite
End Function
```

- **C#**

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    UpdateImageSource(ImageEffect.Shadow);
}

private void CreateDeviceResources()
{
    D3D.FeatureLevel actualLevel;
    D3D.DeviceContext d3dContext = null;
    var d3dDevice = new D3D.Device(IntPtr.Zero);
    var result = HResult.Ok;
    for (int i = 0; i <= 1; i++)
    {
        // ハードウェアが利用できない場合はWARPを使用します
        var dt = i == 0 ? D3D.DriverType.Hardware : D3D.DriverType.Warp;
        result = D3D.D3D11.CreateDevice(null, dt,
            IntPtr.Zero, D3D.DeviceCreationFlags.BgraSupport |
            D3D.DeviceCreationFlags.SingleThreaded,
            null, 0, D3D.D3D11.SdkVersion,
            d3dDevice, out actualLevel,
            out d3dContext);
        if (result.Code != unchecked((int)0x887A0004)) // DXGI_ERROR_UNSUPPORTED
        {
            break;
        }
    }
    result.CheckError();
    d3dContext.Dispose();

    // DXGI装置を格納します（アプリケーションが中断されているときにトリミングするため）
    dxgiDevice = d3dDevice.QueryInterface<DXGI.Device>();
    d3dDevice.Dispose();

    // RenderTargetを作成します（Direct2D描画用のDeviceContext）
    var d2dDevice = D2D.Device1.Create(d2dFactory, dxgiDevice);
    var rt = D2D.DeviceContext1.Create
        (d2dDevice, D2D.DeviceContextOptions.None);
    d2dDevice.Dispose();
    rt.SetUnitMode(D2D.UnitMode.Pixels);
    d2dContext = rt;

    // 組み込みの効果を作成します
    shadow = D2D.Effects.Shadow.Create(rt);
```

```
        affineTransform = D2D.Effects.AffineTransform2D.Create(rt);
        composite = D2D.Effects.Composite.Create(rt);
    }

    void UpdateImageSource(ImageEffect imageEffect)
    {
        // ソース画像の範囲外のピクセルを変更する可能性のある交換があるため、
        // これらのピクセルを表示するためのマージンが必要です
        var targetOffset = new Point2F(marginLT, marginLT);
        int w = bitmap.PixelWidth + marginLT + marginRB;
        int h = bitmap.PixelHeight + marginLT + marginRB;

        // レンダー対象オブジェクト
        var rt = d2dContext;

        // 対象Direct2Dビットマップを作成します
        var bpTarget = new D2D.BitmapProperties1(
            new D2D.PixelFormat(DXGI.Format.B8G8R8A8_UNorm,
            D2D.AlphaMode.Premultiplied),
            (float)bitmap.DpiX,
            (float)bitmap.DpiY, D2D.BitmapOptions.Target |
            D2D.BitmapOptions.CannotDraw);
        var targetBmp =
            D2D.Bitmap1.Create(rt, new Size2L(w, h), bpTarget);

        // 対象ビットマップをレンダー対象に関連付けます
        rt.SetTarget(targetBmp);

        // 描画を開始します
        rt.BeginDraw();

        // 対象ビットマップをクリアします
        rt.Clear(null);

        // C1Bitmap画像をDirect2D画像に変換します
        var d2dBitmap =
            bitmap.ToD2DBitmap1(rt, D2D.BitmapOptions.None);

        switch (imageEffect)
        {
            case ImageEffect.Original:
                rt.DrawImage(d2dBitmap,
                    targetOffset);
                break;
            case ImageEffect.Shadow:
                rt.DrawImage(ApplyShadow(d2dBitmap),
                    targetOffset);
                break;
        }
        d2dBitmap.Dispose();

        if (!rt.EndDraw(true))
        {
            targetBmp.Dispose();

            // 旧GPU装置が削除された場合、装置リソースを再作成してみてください
            DiscardDeviceResources();
            CreateDeviceResources();
            return;
        }

        // 対象ビットマップをデタッチします
        rt.SetTarget(null);

        // 一時的なC1Bitmapオブジェクトを作成します
```

```
    var outBitmap =
        new C1Bitmap(bitmap.ImagingFactory);

    // Direct2D対象ビットマップからC1Bitmapに画像をインポートします
    outBitmap.Import(targetBmp, rt, new RectL(w, h));
    targetBmp.Dispose();

    // C1BitmapをWriteableBitmapに変換し、イメージソースとして使用します
    image.Source = outBitmap.ToWriteableBitmap();
    outBitmap.Dispose();

}

private void DiscardDeviceResources()
{
    shadow.Dispose();
    affineTransform.Dispose();
    composite.Dispose();
    dxgiDevice.Dispose();
    d2dContext.Dispose();
}

D2D.Effect ApplyShadow(D2D.Bitmap1 bitmap)
{
    shadow.SetInput(0, bitmap);
    shadow.BlurStandardDeviation = 5f;
    affineTransform.SetInputEffect(0, shadow);
    affineTransform.TransformMatrix = Matrix3x2.Translation(20f, 20f);
    composite.SetInputEffect(0, affineTransform);
    composite.SetInput(1, bitmap);
    return composite;
}
```