Microsoft Research Face SDK

Microsoft Research Face SDK には、複数の Microsoft Research チームが開発した最新の顔テクノロジが統合されています。現在のベータ リリースでは、顔画像を処理するための最先端のアルゴリズム (顔検出、顔配列、顔追跡、cartoon (似顔絵) の生成など) を提供しています。開発者はこれらのテクノロジを使用して、Windows Phone での興味深いシナリオを試すことができます。

このセクションの内容

内容

プログラミング ガイド

Microsoft Research Face SDK Beta for Windows Phone では、Windows Phone 向け顔関連アプリの開発に役立つさまざまなマネージ API を提供しています。

このセクションの内容

参照

その他の技術情報

[Microsoft Research Face SDK]

概要

Welcome to Microsoft Research Face Software Development Kit (SDK) Beta for Windows Phone をご利用いただき、ありがとうございます。この SDK に同梱されている内容は、次のとおりです。

- 顔関連アルゴリズムを使用するための技術ドキュメント
- マネージ コードでプログラミングを行うための参照 API とドキュメント
- Windows Phone 向け顔関連アプリを開発するためのベスト プラクティスを示したサンプル

Face SDK で実装できる機能には、たとえば次のような機能があります。

- 画像内の顔を検出する。
- 顔の特徴点の位置を特定する。
- ビデオ ストリーム内でリアルタイムの顔を追跡する。
- 1 つの画像からパーソナライズされた漫画の似顔絵を作成する。

顔検出

画像を指定すると、このモジュールでは指定した画像からすべての顔を検出して、顔の位置を示す枠のリストを返します。検出アルゴリズムによって、照明の変化に加え、さまざまな顔の角度が自動的に処理されます。

図 1: 顔検出の結果

概要

この SDK では 3 つの顔検出アルゴリズムを実装しており、どれも Harr 特徴に基づいています。どのアルゴリズムでも、開発者はいくつかのパラメーター (画像領域、ステップ サイズ、ポスト フィルターを適用するかどうか、色情報を使うかどうかなど) を指定して、検出結果を制御できます。3 つの顔検出アルゴリズムの主な違いは、次のとおりです。

- Harr 検出機能は高速で、正面顔を対象としています。
- マルチビュー検出機能は正面顔と横顔の両方の検出に優れていますが、処理に時間がかかります。
- ピラミッド構造を使用したマルチビュー検出機能は、マルチビュー検出機能と似ていますが、メモリ使用量を抑えることができます。

解説

状況によっては、検出アルゴリズムで顔を検出できない場合があります。

顔配列

このモジュールでは、顔の構成要素（目、眉、口、鼻など）の位置を特定して、特定した構成要素の中心または輪郭を返すことを試みます。

概要

この SDK では、Active Shape Model (ASM) 手法とニューラル ネットワーク ベース手法の 2 つの顔配列アルゴリズムを実装しています。

ASM

ASM モデルでは、87 個の点を使用して顔の形を記述します。これらの点は、8 つのグループに分割できます。

- ～ 7: 左目
- 8 ～ 15: 右目
- 16 ～ 25: 左眉
- 26 ～ 35: 右眉
- 36 ～ 47: 鼻
- 48 ～ 59: 口
- 60 ～ 67: 口の内側
- 68 ～ 86: 輪郭

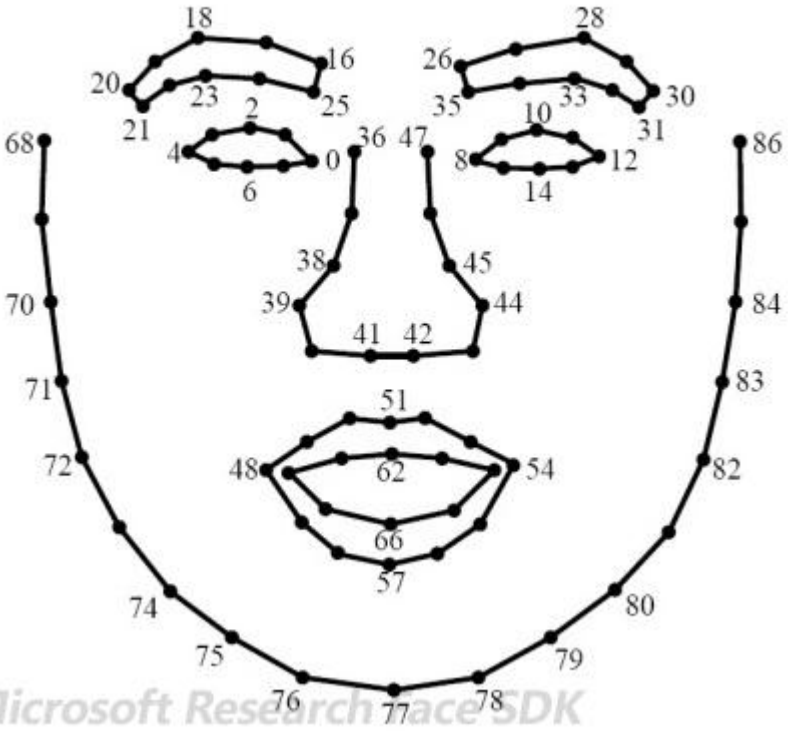


図 1: ASM モデルの位相

図 2: ASM の配列結果

ニューラル ネットワーク

このアルゴリズムでは、顔の構成要素の位置を特定します。特定結果には、左目、右目、鼻、左口角、右口角の順に 5 点が含まれています。

図 3: ニューラル ネットワーク ベースの配列結果

顔追跡

このモジュールでは、ライブ ビデオ ストリーム内でリアルタイムの顔の位置を特定します。ユーザーは、頭を動かすことで (左右に振るなどして) Windows Phone を操作できます。

参照

その他の技術情報

Microsoft Research Face SDK プログラミング ガイド

似顔絵

このモジュールでは、1 つの画像から漫画の似顔絵を作成します。ユーザーは、さまざまなスタイルを選択し、さまざまなテンプレートを適用することで、似顔絵をカスタマイズできます。

図 1: 漫画の似顔絵

以下のセクションでは、似顔絵モジュールの詳細について説明します。

- 似顔絵のパイプライン
- 似顔絵のスタイル
- 似顔絵のテンプレート

似顔絵のパイプライン

似顔絵を生成する手順は、次のとおりです。

- 顔写真を読み込む。
- 顔を検出し、顔の特徴点の位置を特定する。
- スケッチを生成する。
- スケッチにスタイルを設定する。
- スケッチにテンプレートを適用する。
- スケッチを描画する。

注: 生成される似顔絵は、照明や顔の向きによって異なります。均一な照明で撮影した、解像度が高い正面顔の画像が適しています。

似顔絵のスタイル

似顔絵モジュールには、複数のスタイルが用意されています。各スタイルは、似顔絵のさまざまなプロパティ (顔の形、肌色、目の細部、描画スタイルなど) を定義しています。次の図は、サポートされているスタイルを示しています。

図 2: 似顔絵のスタイル

似顔絵のテンプレート

テンプレートは、似顔絵の顔に適用すると似顔絵の表現を向上できる、追加のコンポーネントです。たとえば、ユーザーは、似顔絵に合わせるさまざまな眼鏡、髪型、体などを選択できます。テンプレートは、さらに静的とアニメーションの 2 種類に分類できます。静的テンプレー

トのフレームは 1 つだけですが、アニメーション テンプレートには複数のフレームがあるので、生成される似顔絵 (gif ファイル) をアニメーション表示できます。

静的テンプレートには、背景、ひげ、体、イヤリング、表情、眼鏡、髪型、ペットなどがあります。アニメーション テンプレートには、体や表情などがあります。

図 3: 似顔絵のテンプレート

肌色の分類

肌色の分類アルゴリズムでは、あるピクセルが人間の肌色に一致しているかどうか判定します。

このアルゴリズムは、入力されたカラー画像を事前に処理するために使用します。たとえば、肌と考えられる領域を識別してから顔検出機能でこのような領域を対象に顔を検索すると、検出速度が大幅に向上します。ただし、色が正しく分類されないと、再現率が低下することもあります。

図 1: 元の画像

図 2: 肌の領域

"方法" トピック

この SDK には、マネージ コードで作成した多数のサンプルが付属しています。サンプルでは、SDK の API を使用して、顔を検出し、特徴点の位置を特定し、ライブ ビデオ ストリーム内の顔を追跡する方法を示しています。

新しい API について学習する場合は、ユーザー タスクごとに分解したサンプルを参照すると容易に学習できることがよくあります。ユーザー タスクとは、開発者が確認する必要があるプログラミング タスクを示す、それ以上分解できないレベルのタスクです。このようなサンプルを "方法" トピックと呼んでいます。

"方法" トピックは、各タスクの達成に必要な作業を確認しやすくすることを目的としています。パフォーマンスとわかりやすさのどちらを優先するか検討した結果、"方法" トピックではできる限り簡単な方法で機能を説明しています。このため、"方法" トピックは機能を手早く実現する手段として優れていますが、優れたパフォーマンスを得る方法としては優れているとは限りません。また、"方法" トピックでは、サンプル コードを読みやすくするために、多くの箇所でエラー チェックを省略しています。

次の "方法" トピックは、SDK のサンプルに基づいたユーザー タスクを表しています。

顔検出と顔配列の "方法" トピック

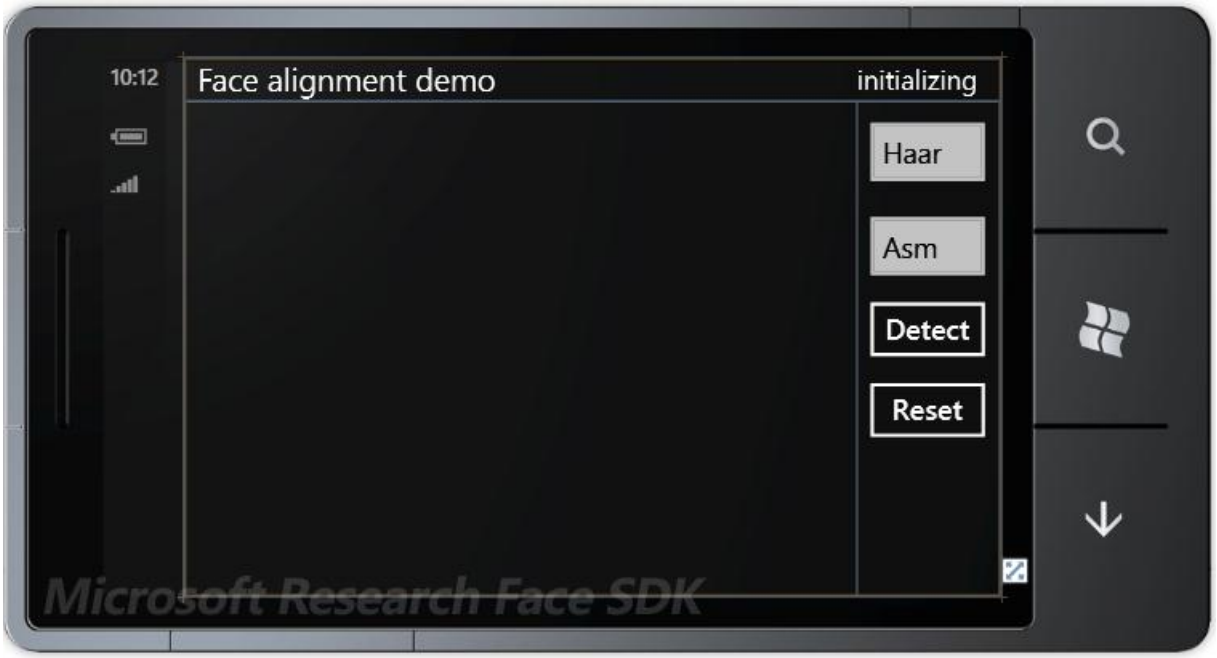
概要

この "方法" トピックでは、FaceAlignmentDemo のユーザー タスクを説明します。このサンプルでは、サンプル写真から顔を検出し、顔ごとに顔の特徴点の位置を特定します。

コーディング方法

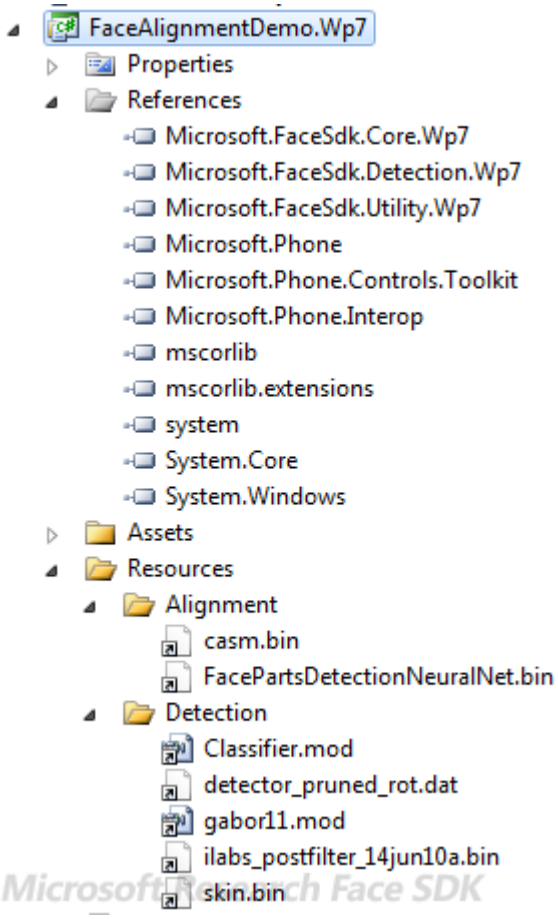
Windows Phone アプリ プロジェクトを作成する

`Image` コントロールと、いくつかの `Button` と `TextBlock` を含む、単純な Windows Phone アプリ プロジェクトを作成できます。次に、GUI のレイアウトを示します。



次に、Microsoft.FaceSdk.Core.Wp7.dll、Microsoft.faceSdk.Detection.Wp7.dll、および Microsoft.FaceSdk.Utility.Wp7.dll を参照します。

プロジェクト フォルダーの下に Resources\Alignment フォルダーと Resources\Detection フォルダーを作成し、関連するリソース ファイルを追加するか、それらのリソース ファイルにリンクし、ファイルの "ビルド アクション" プロパティを "コンテンツ" に設定します。



画像を読み込む

C#

```
var si = Application.GetResourceStream(new Uri("Assets/sample.jpg", UriKind.Relative));
var source = new BitmapImage();
source.CreateOptions = BitmapCreateOptions.None;
source.SetSource(si.Stream);

// まず WriteableBitmap オブジェクトを作成してから、Image<byte> に変換します。
// Face SDK は Image<byte> オブジェクトに対して機能します。
var wb = new WriteableBitmap(source);
var sdkImg = ImageConverter.SystemToSdk(wb);
```

顔検出機能を作成する

C#

```
IFaceDetector detector = FaceDetectorFactory.Create(type);
```

サポートされている検出機能の種類は、FaceDetectionType.Haar、FaceDetectionType.Multiview、および FaceDetectionType.MultiviewPyramid です。

顔を検出する

この顔検出機能は、カラー画像またはグレースケール画像のどちらに対しても機能します。最も簡単に顔を検出するには、次のコードを実行します。

C#

```
var faces = detector.Detect(sdkImg);
```

sdkImg がカラー画像 (RGB 形式または ARGB 形式) の場合、上記のコードはプログラム内部で次のように機能します。

- グレースケール画像を生成する。
- カラー画像上で肌の領域を検出する。
- グレースケール画像上で肌の領域から顔を検出する。

一方、グレースケール画像を入力した場合は、画像全体が顔検出の対象になります。次のコードに示すように、明示的にグレースケール画像を入力できます。

C#
```
var gray = new ImageGray(sdkImg);
var faces = detector.Detect(gray);
```

この検出機能では、開発者が検出動作を完全に制御できる高度な API も提供しています。

C#
```
var faces = detector.Detect(gray, Size.Empty, gray.Size, 0.1f, true, gray.Region, sdkImg);
```

上記のコードでは、sdkImg はカラー画像である必要があります。

この例では、開発者は 6 つのパラメーターを指定できます。パラメーターの詳細は次のとおりです。

- minSize: 検出される顔の最小サイズ。検出可能な顔の最小サイズは IFaceDetector.MinResolution で指定します。このサンプルでは便宜上、空のサイズ (0, 0) を指定しています。
- maxSize: 検出される顔の最大サイズ。検出可能な顔の最大サイズは IFaceDetector.MaxResolution で指定します。このサンプルでは便宜上、画像サイズを指定しています。
- stepRatio: 隣り合うスライディング ウィンドウ (1 つのスライディング ウィンドウは 1 つの検索領域を定義します) の間隔を、縦横両方向にピクセル単位で指定します。たとえば、ウィンドウ サイズが 40x40 でステップ比率が 0.1 の場合、間隔は 4 ピクセルです。比率を小さくすると再現率が向上する可能性がありますが、かかる時間が長くなります。
- usePostFilter: 未処理の検出結果にポスト フィルターを適用するかどうかを示します。
- region: 顔を検索するローカル領域を定義します。
- colorImage: 指定すると、顔検出機能では、指定の色情報に基づいて顔を検索します。たとえば、肌と考えられる領域を識別してからその領域内だけを対象に顔を検索すると、検出速度を大幅に向上できます。

顔配列機能を作成する

C#
```
IFaceAlignmentor alignmentor = FaceAlignmentorFactory.Create(type);
```

サポートされている配列機能の種類は、FaceAlignmentType.Asm と FaceAlignmentType.NeuralNetwork です。

顔の特徴点の位置を特定する

C#
```
var points = alignmentor.Align(gray, faceRect);
```

ここでの faceRect は顔の領域を指しています。通常は、この SDK の顔検出機能からの出力です。他の種類の値を入力する場合は、配列結果は顔の領域サイズの影響を受けやすいため、領域が小さすぎたり大きすぎたりしないようにする必要があります。

結果を視覚化する

C#
```
// 描画機能は、ARGB 画像をキャンバスとして受け取ります。
using (var render = Renderer.Create(new ImageARGB(sdkImg)))
{
  float radius = 3;
  float penWidth = 1.0f;
  foreach(var rc in detectResults)
    render.DrawRectangle(Color.Blue, penWidth, (RectangleF)rc);
  foreach (var shape in alignmentResults)
    render.DrawPoints(shape, radius, Color.Red);

  // Image<byte> を WriteableBitmap に変換し、Image コントロールのコンテンツを更新します。
  imgControl.Source = ImageConverter.SdkToSystem(render.Image);
}
```

顔追跡の "方法" トピック

概要

この "方法" トピックでは、FaceTrackDemo のユーザー タスクを説明します。このサンプルでは、Windows Phone のカメラで撮影したライブ ビデオ内の顔を追跡します。

コーディング方法

Windows Phone アプリ プロジェクトを作成する

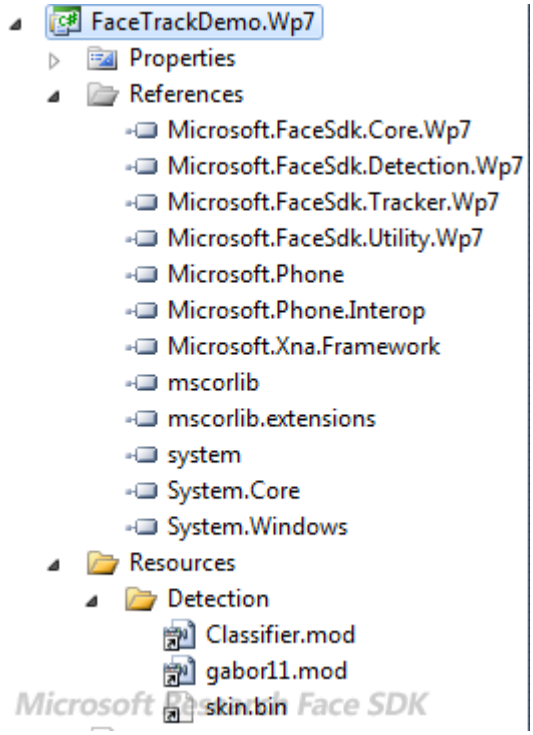
単純な Windows Phone アプリ プロジェクトを作成できます。このプロジェクトには、ライブ ビデオを表示する `VideoBrush` コントロールと、顔の位置を示す四角形が含まれています。

XAML

```
<Rectangle Width="640" Height="480" HorizontalAlignment="Left">
  <Rectangle.Fill>
    <VideoBrush x:Name="viewfinderBrush"/>
  </Rectangle.Fill>
</Rectangle>

<Rectangle x:Name="faceRect"
  StrokeThickness="2"
  Stroke="Red"
  Visibility="Collapsed"
  HorizontalAlignment="Left"
  VerticalAlignment="Top"/>
```

次に、Microsoft.FaceSdk.Core.Wp7.dll、Microsoft.FaceSdk.Detection.Wp7.dll、Microsoft.FaceSdk.Tracker.Wp7.dll、および Microsoft.FaceSdk.Utility.Wp7.dll を参照します。

プロジェクト フォルダーの下に Resources\Detection フォルダーを作成し、関連するリソース ファイルを追加するか、それらのリソース ファイルにリンクし、ファイルの "ビルド アクション" プロパティを "コンテンツ" に設定します。



Windows Phone のカメラからビデオ ストリームを取得する

Windows Phone OS 7.1 以降では、プログラムを使用してデバイスのカメラにアクセスできます。まず `OnNavigatedTo` メソッドでカメラを初期化します。

C#

```
// デバイスのカメラを利用できるかどうか確認します。
if ((PhotoCamera.IsCameraTypeSupported(CameraType.Primary) == true) ||
(PhotoCamera.IsCameraTypeSupported(CameraType.FrontFacing) == true))
{
  // 既定のカメラを初期化します。
  cam = new Microsoft.Devices.PhotoCamera();

  // PhotoCamera オブジェクトが初期化されるとイベントが発生します。
  cam.Initialized += cam_Initialized;

  // VideoBrush のソースをカメラに設定します。
  viewfinderBrush.SetSource(cam);
}
```

これで、ビデオ ストリームをアプリで表示できるようになりました。

ビデオ フレームにアクセスする

上記のコードでは、ビデオ ストリームを取得し、ウィンドウ (VideoBrush) に直接表示しました。

ビデオ フレームにアクセスするには、スレッドを作成して、デバイスのフレームを照会します (これはポンプのように機能します)。各フレームの処理時間を予測できないうえ、ビデオの取得処理を妨げることは望ましくないので、この場合は別のスレッドが必要です。ユーザーがボタンをクリックしたらスレッドを作成します。

C#

```
private Thread ARGBFramesThread;
// ユーザー スレッドにおけるフレームの取得/処理動作を同期するためのメンバー変数
private static ManualResetEvent pauseFramesEvent = new ManualResetEvent(true);
private bool pumpARGBFrames;
```

C#

```
// フレーム ポンプを開始します。
void TrackOn_Clicked(object sender, RoutedEventArgs e)
{
  if (ARGBFramesThread != null && ARGBFramesThread.IsAlive)
  {
    MessageBox.Show("Please stop tracking first!");
  }
  else
  {
    pumpARGBFrames = true;
    ARGBFramesThread = new System.Threading.Thread(PumpARGBFrames);
    ARGBFramesThread.Start();
  }
}

// フレーム ポンプを停止します。
void TrackOff_Clicked(object sender, RoutedEventArgs e)
{
  pumpARGBFrames = false;
}

// フレーム ポンプ スレッドのメソッド
void PumpARGBFrames()
{
  // 取得用バッファーを作成します。
  var input = new Microsoft.FaceSdk.Image.ImageGray((int)cam.PreviewResolution.Width,
(int)cam.PreviewResolution.Height);

  PhotoCamera phCam = (PhotoCamera)cam;

  while (pumpARGBFrames)
  {
    // 他のポンプ スレッドを待機します。
    pauseFramesEvent.WaitOne();
```

```
    // 以降の操作のために、現在の輝度フレームをバッファーにコピーします。
    // このサンプルでは、輝度データのみを顔の追跡に使用します。
    phCam.GetPreviewBufferY(input.Buffer);

    // フレームを処理します。
    // ...

    // イベントをリセットします。
    pauseFramesEvent.Reset();
  }
}
```

顔追跡機能を作成する

C#

```
MOTFaceTracker tracker = new MOTFaceTracker();
```

顔を追跡する

フレームの準備ができたら、ポンプ スレッド内で顔を追跡します。カラー画像をグレースケールに変換するときの計算コストを抑えるには、デバイスのグレースケール フレームを直接照会します (実際にはドライバーで変換します。これは手動で変換するよりも時間がかかりません)。顔の追跡中は、マーカーのサイズと位置を更新します。

C#

```
Stopwatch sw = Stopwatch.StartNew();
// グレースケール フレーム内の顔を追跡します。
var rc = tracker.Track(input);
sw.Stop();

var ellapsed = sw.ElapsedMilliseconds;

// UI を更新します。
Deployment.Current.Dispatcher.BeginInvoke(()=>
{
  txtDebug.Text = string.Format("track time: {0} ms, {1}", ellapsed, !rc.IsEmpty ? "Yes" : "No");

  faceRect.Margin = new Thickness(rc.Left, rc.Top, 0, 0);
  faceRect.Width = rc.Width;
  faceRect.Height = rc.Height;
  faceRect.Visibility = System.Windows.Visibility.Visible;

  pauseFramesEvent.Set();
});
```

似顔絵の "方法" トピック

概要

この "方法" トピックでは、CartoonDemo のユーザー タスクを説明します。このサンプルでは、ランダムにスタイルやテンプレートを適用することで、漫画の似顔絵を作成します。

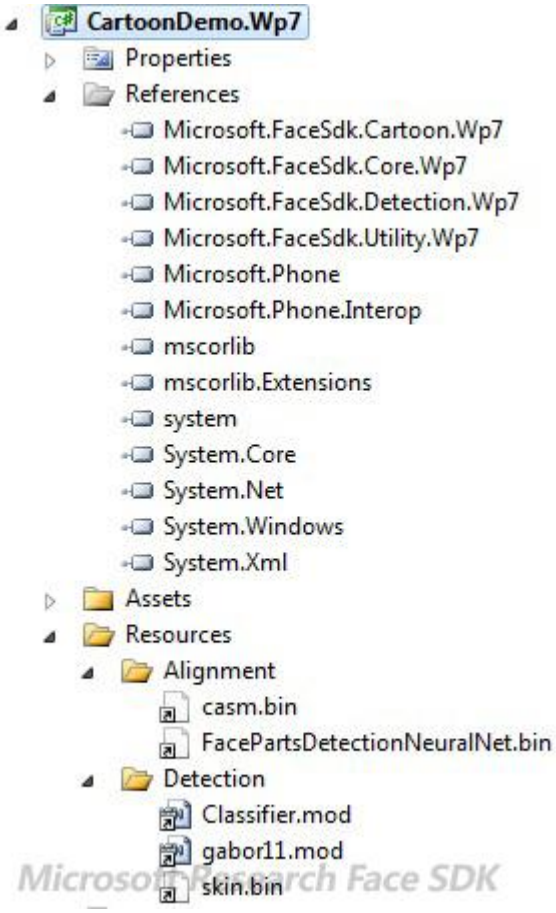
コーディング方法

Windows Phone アプリ プロジェクトを作成する

`Image` コントロール、1 つの `Button`、およびいくつかの `TextBlock` を含む、単純な Windows Phone アプリ プロジェクトを作成できます。

次に、Microsoft.FaceSdk.Core.Wp7.dll、Microsoft.FaceSdk.Detection.Wp7.dll、Microsoft.FaceSdk.Cartoon.Wp7.dll、および Microsoft.FaceSdk.Utility.Wp7.dll を参照します。

プロジェクト フォルダーの下に Resources\Alignment フォルダーと Resources\Detection フォルダーを作成します。関連するリソース ファイルを追加するか、それらのリソース ファイルにリンクし、ファイルの "ビルド アクション" を "コンテンツ" に設定します。



似顔絵エンジンを作成する

C#

```
CtnEngine engine = CtnEngine.Create();

CtnSys sys = CtnEngine.CtnSystem;

// サポートされている似顔絵スタイルを照会します。
var filters = (from fc in sys.Filters
  where fc.Id == CtnGuid.GUID_CATEGORY_SKETCHGENERATION
  from f in fc.Filters
  select f).ToArray();
```

CtnSys オブジェクトには、似顔絵モジュール全体の構成が格納されています。このオブジェクトから、サポートされているスタイルやテンプレートを照会できます。

スケッチを作成する

C#

```
ICtnSketch sketch = CtnEngine.CreateSketch();

var img = ImageConverter.SystemToSdk(new WriteableBitmap(bmpSrc));
sketch.Image = img;
```

ICtnSketch オブジェクトでは、漫画の似顔絵のプロパティをすべて定義します。上記のコードでは、空のスケッチを作成しています。この空のスケッチを初期化して、他のパイプラインで処理します。

顔を検出する

C#

```
IFaceDetection detector = engine.GetFaceDetector();
IFaceAlignment alignmentor = engine.GetFaceAlignmentor();

// 顔を検出します。
var rect = detector.Detect(sketch);
```

```
// 顔をトリミングします。
detector.FinalizeRect(sketch, rect);

// 顔の特徴点の位置を特定します。
alignmentor.LoadRawSketch(sketch);
sketch = alignmentor.GetSketch();
```

このアルゴリズムでは、実際の人物における顔の形に基づいて似顔絵を生成します。最初の段階では、顔の特徴点を検出します。配列結果の精度は、実際の人物に似た漫画の似顔絵を生成するうえで非常に重要です。通常は、解像度が高い正面顔の画像を使用すると精度が向上します。

スタイルを適用する

C#

```
int imageWidth = 512;
int imageHeight = 512;
Random rng = new Random(DateTime.Now.Millisecond);

// ランダムにスタイルを取得します。
int generationId = rng.Next(0, filters.Length);
Guid generatorId = filters[generationId].Id;

// プログラム内部では、スタイルの作成機能で UIElement オブジェクトである WriteableBitmap が使用されるので、
// スレッドをまたいだ UI オブジェクトへのアクセスに関する問題を防ぐために、このメソッドは UI スレッド内で呼び出してください。
ISketchGeneration generator = engine.GetSketchGenerator(generatorId);

// スケッチのサイズを指定します。
SdkRect rcBound = new SdkRect(0, imageWidth, 0, imageHeight);
generator.Generate(sketch, rcBound);
```

スタイル ジェネレーターは UI スレッド内に作成されることに注意してください。また、似顔絵のスタイル設定には数十秒かかるので、開発者は UI スレッドの応答を考慮してください。

顔を誇張する

C#

```
IFaceExaggeration exaggerator = engine.GetFaceExaggerator();
exaggerator.Exaggerate(sketch, ExaggerationType.AutoAddConstr, 0.0f);
```

誇張段階の処理は省略できます。誇張すると、似顔絵の顔の形を変形できます (たとえば、口や鼻の位置の調整、顔の拡大/縮小など)。誇張しない場合は、より写実的な似顔絵になります。

テンプレートを適用する

C#

```
var templateFilters = engine.GetCartoonTemplates(generatorId);
foreach (var tf in templateFilters)
{
  if (tf.Count == 0) continue;

  if (tf.TemplateId == CtnGuid.GUID_TEMPLATETYPE_ANIMATEBODYCLOTH ||
    tf.TemplateId == CtnGuid.GUID_TEMPLATETYPE_ANIMATEBODYCLOTH_P ||
    tf.TemplateId == CtnGuid.GUID_TEMPLATETYPE_ANIMATEEXPRESSION ||
    tf.TemplateId == CtnGuid.GUID_TEMPLATETYPE_MULTIPART_ANIMATION)
    continue;

  int index = rng.Next(0, tf.Count);
  tf.Apply(sketch, index);
}
```

テンプレートはスタイルに関連付けられています。言い換えると、一部のテンプレートは、特定のスタイルだけで利用できます。上記のコードでは、各テンプレート カテゴリのテンプレートを 1 つランダムに適用しています。詳細については、「似顔絵」を参照してください。

似顔絵を描画する

C#

```
float ratio, tx, ty;
sketch.FitToRect(SdkRect.FromLTWH(0, 0, imageWidth, imageHeight), out ratio, out tx, out ty);
var sketchImage = sketch.Render(imageWidth, imageHeight);

var wb = ImageConverter.SdkToSystem(sketchImage);
imgControl.Source = wb;
```

似顔絵を描画するキャンバスのサイズは、指定可能です。ICtnSketch オブジェクトでは、似顔絵をキャンバスのサイズに適合し (似顔絵とキャンバスの縮尺を統一し)、変換パラメーターを返します。

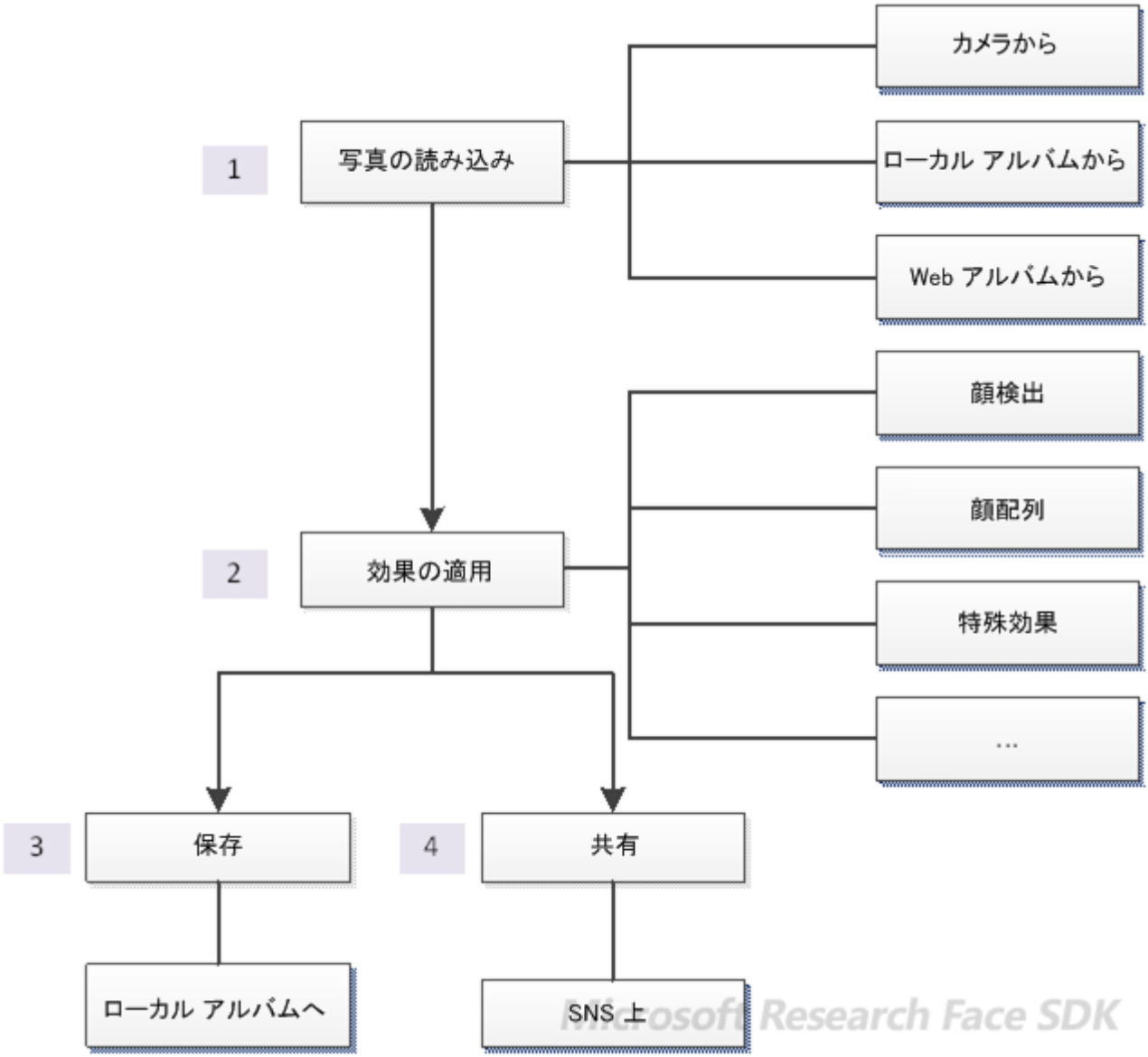
Face Party の "方法" トピック

概要

この "方法" トピックでは、FaceParty のユーザー タスクを説明します。FaceParty は、Microsoft Research Face SDK を利用した Face Party (英語) アプリ シリーズのサンプル アプリです。このアプリの目的は、開発者が Face Party アプリの構造を理解して独自の顔関連アプリを迅速に開発できるようにすることです。

アプリのワークフロー

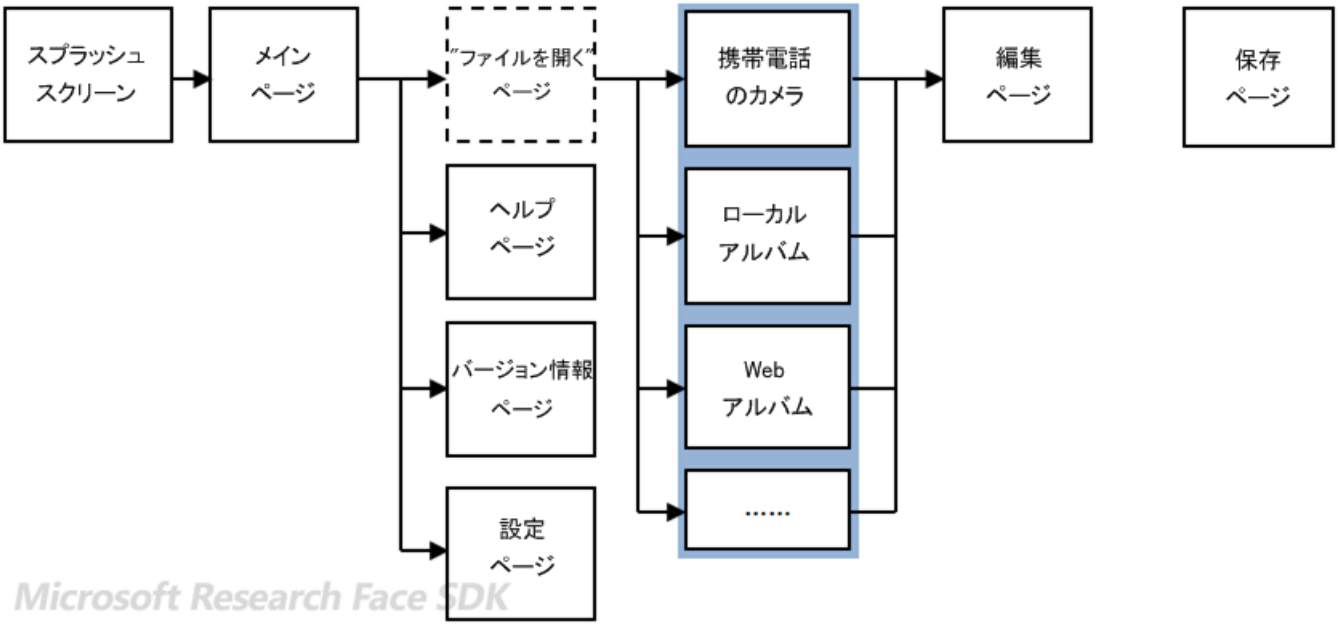
一般的な Face Party アプリは、次の 4 段階で実行されます。



1. 写真の読み込み: アプリを起動すると、ユーザーは 1 枚以上の写真を開くよう要求されます。写真の取得方法には、ローカル アルバム、携帯電話のカメラ、またはインターネットからの 3 種類があります。
2. 効果の適用: この段階は、開発者が独自の処理を試すことができる部分の中核です。
3. 保存: 写真の編集結果をローカル アルバムに保存します。
4. 共有: SNS への投稿など、写真を共有する手段も用意します。

ページ構成

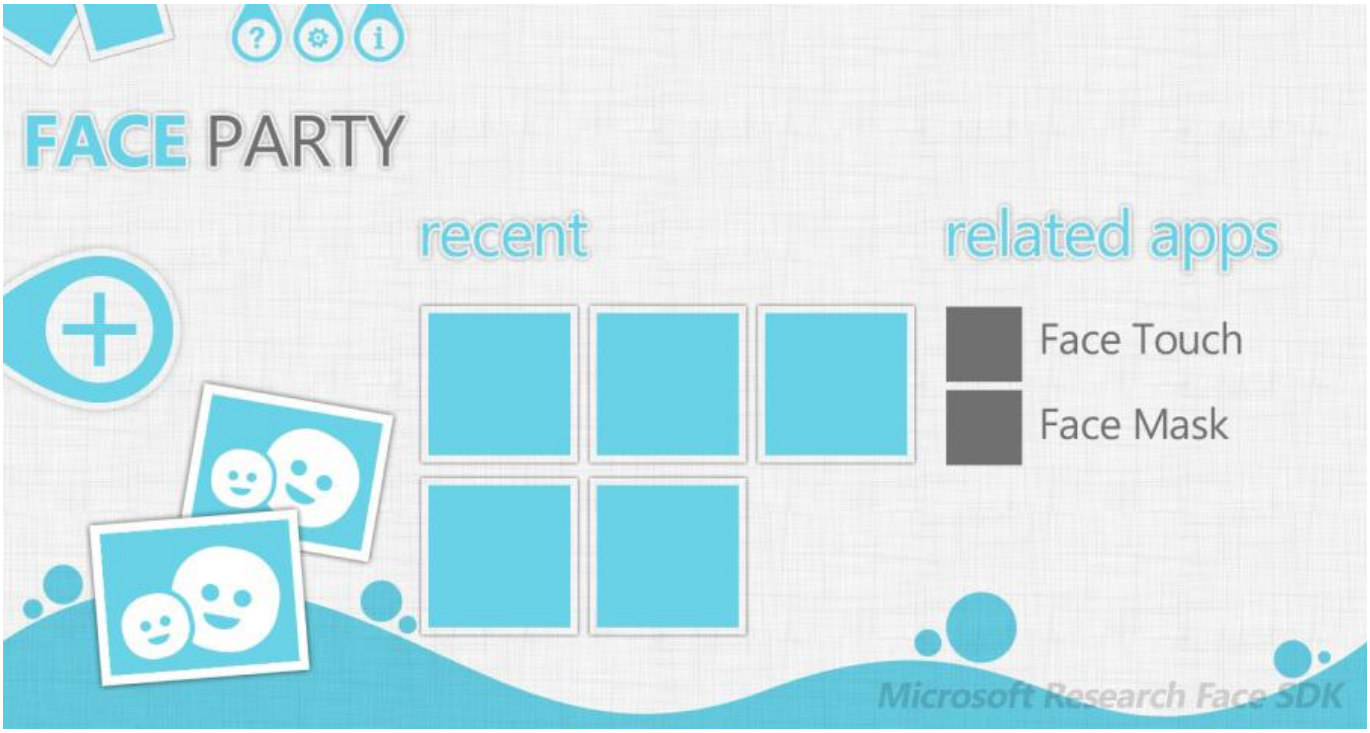
次の図は、一般的な Face Party アプリのページ構成例を示しています。独自のアプリを開発する際は、この対話の流れに従うことをお勧めします。以降のセクションでは、各ページの詳細について説明します。



注: この図の "ファイルを開く" ページとは、写真を読み込むときのリダイレクト先ページを表す、仮想的なページです。リダイレクト先のページは、システム既定の場合もあれば、サードパーティ製の場合もあります。

コーディング方法

メイン ページ



メイン ページは、アプリの入り口です。このページでは、Panorama コントロールを使用してアプリの情報 (タイトル、最近編集した写真の一覧、関連アプリなど) を表示します。次のコードは、Panorama コントロールのレイアウト構造を示しています。開発者は、このレイアウトを変更せずコンテンツ (画像、テキストなど) だけを更新することをお勧めします。

XAML

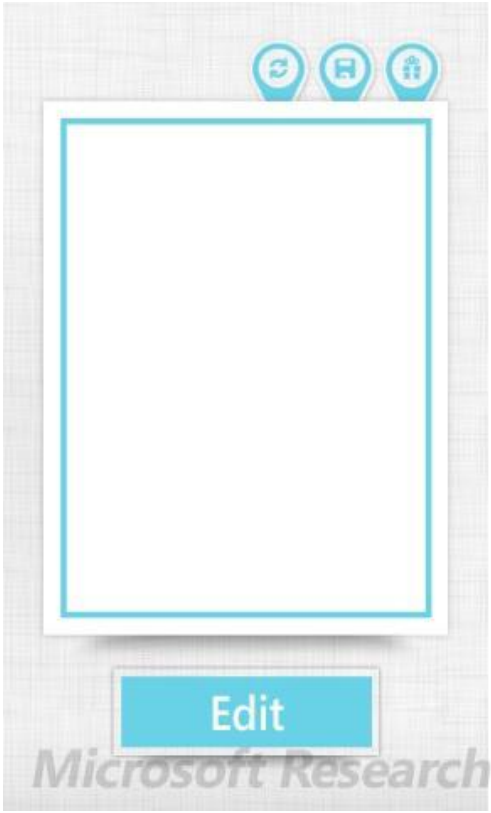
```
<controls:Panorama x:Name="MainPanorama" TitleTemplate="{StaticResource TitleTemplate}">
  <controls:Panorama.Background>
    <ImageBrush ImageSource="/Assets/Images/mainbg.jpg" />
  </controls:Panorama.Background>
  <controls:PanoramaItem x:Name="AddPhotoItem">
    <Grid>
      <Grid.RowDefinitions>
        <RowDefinition Height="250"/>
        <RowDefinition/>
        <RowDefinition Height="35"></RowDefinition>
      </Grid.RowDefinitions>
      <uictrl:StateImageButton Grid.Row="0" x:Name="AddButton" VerticalAlignment="Top"
HorizontalAlignment="Left"
        Width="262" Height="196" Margin="-80 20 0 0"
        NormalImage="/Assets/Images/add.png"
        PressedImage="/Assets/Images/addtap.png" Click="AddButton_Click"/>
      <Image Grid.Row="1"  Source="/Assets/Images/hpsample.png" Width="351"
VerticalAlignment="Top" HorizontalAlignment="right"/>
      <TextBlock Grid.Row="2" Margin="5 0" Text="Microsoft Research" FontSize="21"
Foreground="#FF707070"></TextBlock>
    </Grid>
  </controls:PanoramaItem>
  <controls:PanoramaItem x:Name="RecentListItem"
    HeaderTemplate="{StaticResource RecentHeader}" Orientation="Horizontal">
    <uictrl:PhotoListPanel x:Name="RecentList" CanDelete="True" Orientation="Vertical"
MaxHeight="500"  HorizontalAlignment="Left" VerticalAlignment="Top">
      <uictrl:PhotoListPanel.EmptyContent>
        <ContentPresenter>
          <StackPanel Orientation="Horizontal" VerticalAlignment="Top" HorizontalAlignment="Left">
            <TextBlock Text="Tap the" FontSize="30" Margin="10" Foreground="Gray" />
            <Image Source="/Assets/Images/addtip.png" Width="32"/>
            <TextBlock Text="to open a picture." FontSize="30" Margin="10" Foreground="Gray" />
          </StackPanel>
        </ContentPresenter>
      </uictrl:PhotoListPanel.EmptyContent>
    </uictrl:PhotoListPanel>
    </controls:PanoramaItem>
      <controls:PanoramaItem HeaderTemplate="{StaticResource RelatedAppsHeader}">
    <uictrl:RelatedAppList FilterAppName="Face Party"></uictrl:RelatedAppList>
  </controls:PanoramaItem>
</controls:Panorama>
```

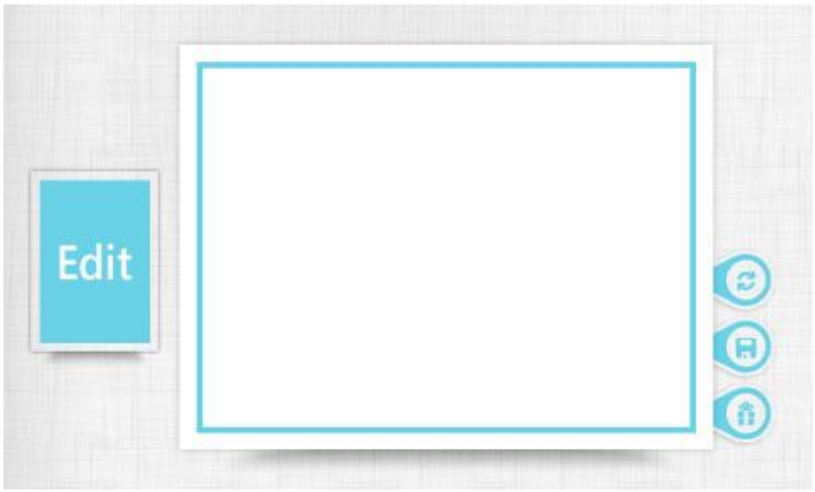
編集ページ

編集ページでは、顔画像を処理してその結果を表示します。このページには、次のような機能があります。

- 顔画像の処理 (顔検出、顔配列など)
- メタデータのローカル ストレージへの保存とローカル ストレージからの読み込み
- 編集後の画像の表示
- 編集、保存、共有などのユーザー操作のサポート

ページの向き





編集ページでは、横長と縦長の両方の向きをサポートしています。向きの変化を管理しやすいよう、XAML ファイルのリソースとして、UI 要素ごとに縦長、横長 (左側)、および横長 (右側) の 3 つのスタイルを定義しています。次のコードは、保存ボタンの 3 つのスタイルを示しています。

XAML

```
<Style x:Key="SaveButtonStyle" TargetType="uictrl:StateImageButton">
  <Setter Property="BorderThickness" Value="0"/>
  <Setter Property="Grid.Column" Value="1" />
  <Setter Property="Grid.RowSpan" Value="3" />
  <Setter Property="Width" Value="77"/>
  <Setter Property="Stretch" Value="Fill"/>
  <Setter Property="NormalImage" Value="/Assets/Images/save.png"/>
  <Setter Property="PressedImage" Value="/Assets/Images/savetap.png"/>
  <Setter Property="DisabledImage" Value="/Assets/Images/savedisabled.png"/>
</Style>

<Style x:Key="SaveButtonStyleLL" TargetType="uictrl:StateImageButton">
  <Setter Property="BorderThickness" Value="0"/>
  <Setter Property="Grid.Row" Value="1" />
  <Setter Property="Grid.ColumnSpan" Value="3" />
  <Setter Property="Width" Value="110"/>
  <Setter Property="Stretch" Value="Fill"/>
```

```
  <Setter Property="NormalImage" Value="/Assets/Images/saveLL.png"/>
  <Setter Property="PressedImage" Value="/Assets/Images/savetapLL.png"/>
  <Setter Property="DisabledImage" Value="/Assets/Images/savedisabledLL.png"/>
</Style>

<Style x:Key="SaveButtonStyleLR" TargetType="uictrl:StateImageButton">
  <Setter Property="Margin" Value="-35,0,0,0" />
  <Setter Property="BorderThickness" Value="0"/>
  <Setter Property="Grid.Row" Value="1" />
  <Setter Property="Grid.ColumnSpan" Value="3" />
  <Setter Property="Width" Value="110"/>
  <Setter Property="Stretch" Value="Fill"/>
  <Setter Property="NormalImage" Value="/Assets/Images/saveLR.png"/>
  <Setter Property="PressedImage" Value="/Assets/Images/savetapLR.png"/>
  <Setter Property="DisabledImage" Value="/Assets/Images/savedisabledLR.png"/>
</Style>
```

3 つのスタイルの名前が同じ文字列 (SaveButtonStyle など) で始まっていることに注意してください。続いて、コードでスタイルを登録します。

C#

```
static Dictionary<string, string> RegisteredStyles = new Dictionary<string, string>()
{
    // "<コントロール名>, <スタイル名>" の形式
    {"saveButton", "SaveButtonStyle"},
    ... ...
};
```

ページの向きが変化したら、ビジュアル ツリーに含まれる登録済みのコントロールを検索し、現在のページの向きに応じてスタイルを適用します。

C#

```
private void ApplyStyles(PageOrientation pageOrientation)
{
  string suffix = GetStyleSuffix(pageOrientation);
  ApplyStylesRec(this.LayoutRoot, suffix);
}

private void ApplyStylesRec(FrameworkElement element, string suffix)
{
  if (element == null)
    return;

  if (element.Style != null)
  {
    string oriStyle = FindOriginalStyle(element.Name);
    if (!string.IsNullOrEmpty(oriStyle))
    {
      string newStyle = oriStyle + suffix;
      if (Resources.Contains(newStyle))
        element.Style = (Style)Resources[newStyle];
  }

  // パネルに対して反復処理します。
  if (element is Panel)
  {
    Panel container = element as Panel;
    foreach (var item in container.Children)
      ApplyStylesRec(item as FrameworkElement, suffix);
  }
  else if (element is Border)
    ApplyStylesRec((element as Border).Child as FrameworkElement, suffix);
}

private string GetStyleSuffix(PageOrientation pageOrientation)
{
  if (pageOrientation == PageOrientation.LandscapeLeft)
```

```
    return "LL";
  else if (pageOrientation == PageOrientation.LandscapeRight)
    return "LR";
  else
    return null;
}

private string FindOriginalStyle(string elementName)
{
  if (string.IsNullOrEmpty(elementName))
    return null;
  if (RegisteredStyles.ContainsKey(elementName))
    return RegisteredStyles[elementName];
  else
    return null;
}
```

画像処理

編集ページでは、選択した写真に含まれている顔を検出します。検出速度を向上するには、大きい画像のサイズ縮小、顔の最小サイズの指定、色情報の使用による顔の検出などを行うことができます。ただし、このような対策を実施すると検出率が低下するおそれがあることに注意してください。

C#

```
public Face[] DetectFace(Image<byte>image)
{
  var smallImg = image;
  float scale = 1.0f;
  if (smallImg.Width > MaxImageDimension || smallImg.Height > MaxImageDimension)
  {
    smallImg = CropScale.ScaleImage(image, MaxImageDimension, out scale);
  }

  var gray = new ImageGray(smallImg);
  FaceRect[] faces = _detector.Detect(gray, new Size(MinFaceSize, MinFaceSize), gray.Size,
DetectorStep, true, null, smallImg);

  var alignedFaces = new List<Face>(faces.Length);
  float invScale = 1.0f / scale;
  foreach (var f in faces)
  {
    var shape = _alignmentor.Align(gray, f.Rect);
    shape.UpdateEach(p => p * invScale);
    alignedFaces.Add(new Face
    {
      Shape = shape,
      Region = (Rectangle)(f.Rect * invScale),
      Forehead = InterpolateForehead(shape)
    });
  }

  return alignedFaces.ToArray();
}
```

検出した顔それぞれに、マスクを重ねます。ユーザーはこのマスクを操作して、顔の位置、向き、およびサイズを調整したり、以降の処理対象から顔を除外したりすることができます。

ローカル ストレージ

最近処理した写真とメタデータは、ローカル ストレージに保存します。ローカル ストレージへのアクセスは Photo オブジェクトによって委任され、写真のストリームに対する読み書きは開発者が処理を指定する必要があります。次のコードは、メタデータ ストリームにアクセスする方法を示しています。

C#

```
private static FaceWrapper[] GetMetaData(Photo photo)
{
  FaceWrapper[] faces = null;
  try
  {
    photo.MetaStream.Position = 0;
    XmlSerializer serializer = new XmlSerializer(typeof(FaceWrapper[]));
    faces = serializer.Deserialize(photo.MetaStream) as FaceWrapper[];
  }
  catch
  { }

  return faces;
}

private static void SaveMetaData(Photo photo, FaceWrapper[] wrappers)
{
  if (wrappers != null)
  {
    photo.MetaStream = new MemoryStream();
    XmlSerializer serializer = new XmlSerializer(typeof(FaceWrapper[]));
    serializer.Serialize(photo.MetaStream, wrappers);
  }
}
```

共有ページ

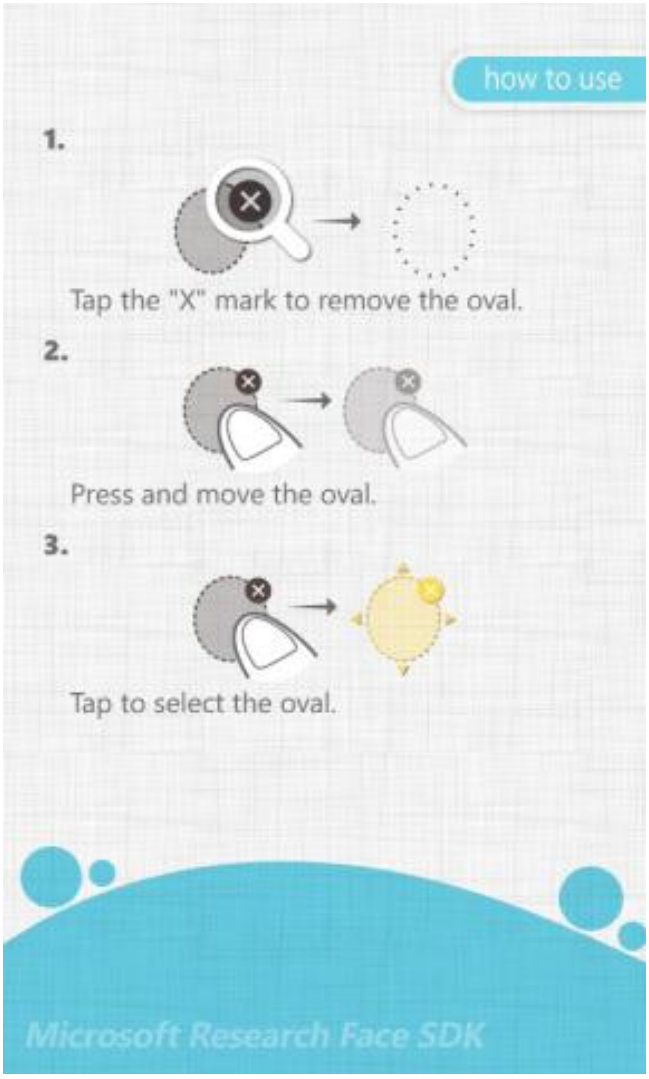
共有ページは、UIFramework ライブラリのコントロールとして実装されています。開発者は背景画像と配色をカスタマイズできます。

共有ページには、画像やテキストを人気のある複数のソーシャル ネットワーク プラットフォーム (Facebook、SkyDrive、Twitter (英語)、Weibo (中国語) など) にアップロードするためのプロトコルが実装されています。開発者は、サービスごとにアプリを作成して、Face Party アプリでのアプリ ID とアプリ シークレットを指定する必要があります (ID とシークレットの例については、$FaceParty\Common\Constant.cs を参照してください)。次のコードは、共有ページを作成する方法を示しています。

C#

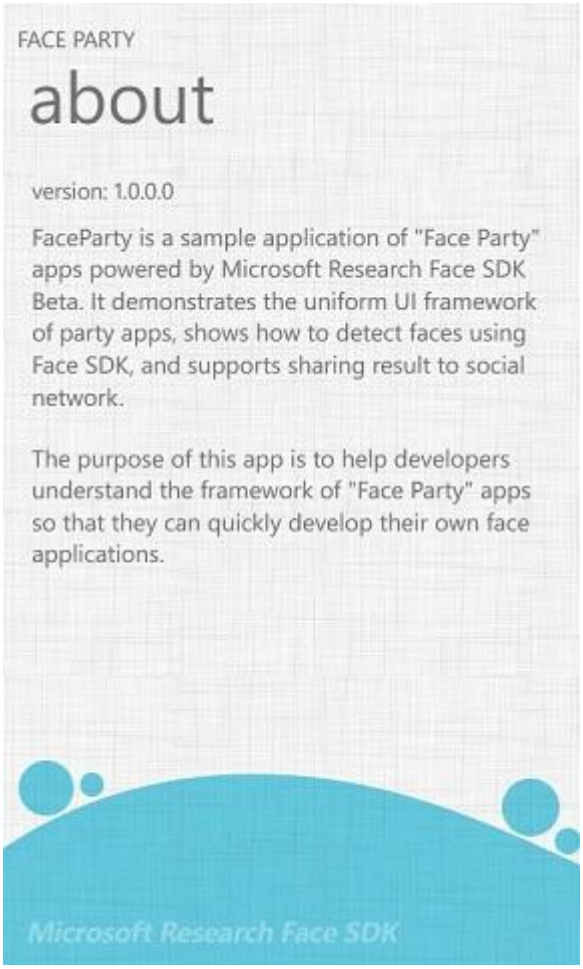
```
StringBuilder uriBuild = new StringBuilder("/UIFramework;component/Pages/PortraitSharePage.xaml");
uriBuild.AppendFormat("?FacebookAppId={0}", Constant.FacebookAppId);
uriBuild.AppendFormat("&FacebookAppSecret={0}", Constant.FacebookAppSecret);

uriBuild.AppendFormat("&TwitterAppCallBack={0}", Constant.TwitterAppCallBack);
uriBuild.AppendFormat("&TwitterAppSecret={0}", Constant.TwitterAppSecret);
uriBuild.AppendFormat("&TwitterAppKey={0}", Constant.TwipicAppKey);

uriBuild.AppendFormat("&SinaWeiboAppId={0}", Constant.SinaWeiboAppId);
uriBuild.AppendFormat("&SinaWeiboAppSecret={0}", Constant.SinaWeiboAppSecret);
uriBuild.AppendFormat("&SinaWeiboAppCallBack={0}", Constant.SinaWeiboAppCallBack);

uriBuild.AppendFormat("&WindowsLiveAppId={0}", Constant.WindowsLiveAppId);
uriBuild.AppendFormat("&WinsowsLiveAppSecret={0}", Constant.WinsowsLiveAppSecret);
NavigationService.Navigate(new Uri(uriBuild.ToString(), UriKind.Relative));
```

ヘルプ ページ



ヘルプ ページでは、操作方法やヒントを表示します。

バージョン情報ページ



バージョン情報ページでは、アプリについて簡単に説明し、使用条件やフィードバックへのリンクを表示します。

参照

その他の技術情報

Microsoft Research Face SDK の "方法" トピック