Getting Started

At the end of this project you will have a simple Windows 10 app that uses face detection on a live video stream from a camera to trigger musical events. Users will be able to make music by moving their faces to different positions on the screen. This is accomplished by creating a grid of cells and assigning a unique sound to each cell. When a face is detected within a cell, the sound is triggered. The FaceTracker API makes it possible for multiple people to join as it is able to track several faces at one time.

Showcasing new APIs for image processing and low-latency audio.

Estimated time commitment: 2 hours

Technical requirements:

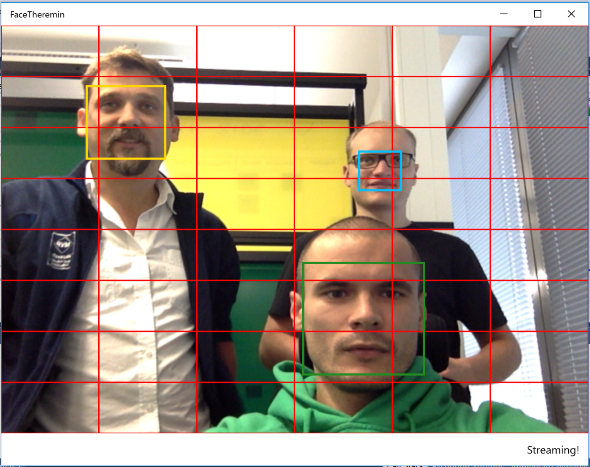
* [Visual Studio 2015 and Windows developer tooling](https://dev.windows.com/en-us/downloads)
* Ensure you are using [Windows 10](https://www.microsoft.com/en-us/windows/windows-10-upgrade) or better
* Set of audio samples in MP3 format, or use the ones provided in the project solution
* Hardware:
  + A device that supports video capture.

Sample features:

Note: Features in this app are subject to change.

* [FaceTracker](https://msdn.microsoft.com/en-us/library/windows/apps/windows.media.faceanalysis.facetracker.aspx) – Detects faces in VideoFrame objects and tracks faces across subsequent video frames.
  + Part of [Windows.Media.FaceAnalysis](file:///C:\Users\v-rehodg\Desktop\Windows.Media.FaceAnalysis) namespace which provides APIs for face detection in bitmaps or video frames.
* [Audio Graphs class](https://msdn.microsoft.com/en-us/library/windows/apps/mt203787.aspx) - parent of all nodes that make up the graph.
* [MediaCapture class](https://msdn.microsoft.com/en-us/library/windows/apps/windows.media.capture.mediacapture.aspx) – Provides functionality for capturing photos, audio, and videos from a capture device, such as a webcam.

Code at a glance:



We use [CaptureElement](https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.controls.captureelement.aspx) for streaming video from a webcam and [TextBlock](https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.controls.textblock.aspx) for status of the app.

**Tip:** You may want to invert the image horizontally so that you see a mirror image of yourself.

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid RenderTransformOrigin="0.5,0.5">

<Grid.RenderTransform>

<ScaleTransform ScaleX="-1"/>

</Grid.RenderTransform>

<CaptureElement x:Name="StreamingElement"

HorizontalAlignment="Stretch"

VerticalAlignment="Stretch"

Stretch="Fill">

</CaptureElement>

</Grid>

<RelativePanel Grid.Row="1" Margin="12">

<TextBlock Text="{x:Bind StatusText, Mode=OneWay}"

RelativePanel.AlignRightWithPanel="True"

RelativePanel.AlignVerticalCenterWithPanel="True"/>

</RelativePanel>

</Grid>

To render a live video preview, use [MediaCapture](https://msdn.microsoft.com/en-us/library/windows/apps/windows.media.capture.mediacapture.aspx) with StreamingCaptureModeVideo mode and set it as a source for the CaptureElement.

var settings = new MediaCaptureInitializationSettings {StreamingCaptureMode = StreamingCaptureMode.Video};

\_mediaCapture = new MediaCapture();

await \_mediaCapture.InitializeAsync(settings);

\_videoProperties = \_mediaCapture.VideoDeviceController.GetMediaStreamProperties(MediaStreamType.VideoPreview) as VideoEncodingProperties;

StreamingElement.Source = \_mediaCapture;

await \_mediaCapture.StartPreviewAsync();

To detect faces, establish a loop that runs a face detection function every N milliseconds. Timer is an easy way to accomplish this. For every “tick”, MediaCapture provides us with a VideoFrame which we can pass to the FaceTracker to get a list of DetectedFaces.

using (var previewFrame = new VideoFrame(inputPixelFormat, (int)\_videoProperties.Width, (int)\_videoProperties.Height))

{

await \_mediaCapture.GetPreviewFrameAsync(previewFrame);

IList<DetectedFace> faces;

var faces = await \_faceTracker.ProcessNextFrameAsync(previewFrame);

Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () => {

ProcessFaceCells(previewFrameSize, faces);

});

}

You’ll need 48 audio files to use in this project. There are multiple options for audio file production available to developers, including real-time tone generation; however, the simplest method is to have a collection of readymade MP3 or WAV files.

Previously, MediaElement was a good option for hassle-free playback of pre-recorded media in Windows Runtime. Or, if you needed really low-latency playback, you could use the XAudio2 API which is C++ based and more complex.

Now, there is a new [Audio Graph API](https://msdn.microsoft.com/en-us/library/windows/apps/mt203787.aspx) available that is much easier to use, supports compressed audio formats, and uses the Windows 10 low-latency audio pipeline.

In this example, three types of sounds were selected: drums, piano chords, and synth tones. Each sound type is assigned to two columns with 8 variations each – 48 audio files in all. AudioGraph connects several input nodes, which can be generated, played from files, or MIDI, to one or more output nodes like your system speaker.



Next, initialize AudioGraph and create an output node.

var settings = new AudioGraphSettings(AudioRenderCategory.Media);

var result = await AudioGraph.CreateAsync(settings);

if (result.Status == AudioGraphCreationStatus.Success)

{

\_audio = result.Graph;

var outputResult = await \_audio.CreateDeviceOutputNodeAsync();

AudioDeviceOutputNode audioDeviceOutputNode = null;

if (outputResult.Status == AudioDeviceNodeCreationStatus.Success)

{

audioDeviceOutputNode = outputResult.DeviceOutputNode;

}

}

Each sound has a separate [StorageFile](https://msdn.microsoft.com/en-us/library/windows/apps/windows.storage.storagefile.aspx) and [AudioFileInputNode](https://msdn.microsoft.com/en-us/library/windows/apps/windows.media.audio.audiofileinputnode.aspx). After initialization, we can add an output node and store sounds in a two-dimension array.

var instruments = new[] {"snd\_2", "snd\_1", "synth\_2", "synth\_1", "drum\_2", "drum\_1"};

var storageFiles = new StorageFile[CellsRowsCount, CellsColumnsCount];

for (var i = 0; i < instruments.Length; i++)

{

await LoadStorageFiles(storageFiles, i, instruments[i]);

}

for (var y = 0; y < CellsRowsCount; y++)

{

for (var x = 0; x < CellsColumnsCount; x++)

{

var inputResult = await \_audio.CreateFileInputNodeAsync(storageFiles[y, x]);

if (inputResult.Status == AudioFileNodeCreationStatus.Success)

{

var audioFileInputNode = inputResult.FileInputNode;

audioFileInputNode.Stop();

audioFileInputNode.AddOutgoingConnection(audioDeviceOutputNode);

\_audioFileInputNodes[y, x] = audioFileInputNode;

}

}

}

We now know the coordinates of the detected faces and can start audio playback.

foreach (var audioFileInputNode in newCells.Select(x => \_audioFileInputNodes[x.Y, x.X]))

{

audioFileInputNode.Reset();

audioFileInputNode.Start();

}

Additional Resources:

For a complete and comprehensive example of face tracking, take a look at CameraFaceDetection from the official UWP sampler repository.

<https://github.com/Microsoft/Windows-universal-samples/tree/master/Samples/CameraFaceDetection>

A Studio in the Palm of Your Hand: Developing Audio and Video Creation Apps for Windows 10

<https://channel9.msdn.com/Events/Build/2015/3-634>

Camera: Developing Powerful Camera Apps

<https://channel9.msdn.com/Events/Build/2015/2-730>

Developing Audio and Video Apps

<https://channel9.msdn.com/Events/Build/2015/3-747>