Included in this document are the requirements for Coding4Fun project submissions. Text in black and blue should not be modified. Text in orange provides direction.

\_\_\_\_Before you begin building a project \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Project/sample guidelines:

* Code uses consistent code conventions, clear function/method/variable names, and a sensible public API.
* The code is clearly commented, documenting intentions and edge cases.
* The project has a name that is easy to remember, gives some idea what the project does, and does not conflict with an existing project or infringe on any [trademarks](http://www.uspto.gov/).
* Provided to Microsoft using private GitHub.
* If necessary, the process for releasing a new version is [clearly documented](https://github.com/github/linguist#releasing).
* There are no sensitive materials in the revision history, issues, or pull requests.
* The project uses publicly-accessible continuous integration that integrates with the status API.
* It has a CONTRIBUTING.md if necessary.

To help us determine if further code review is required, please answer the following:

* Provide code line count (You can use Visual Studio to determine the line count. This limit includes automatically generated code.): **155**
* Does the code does contain any third-party code or content that is authored by any party other than Microsoft: **No**
* Does the code use an Apache, Mozilla, or copyleft license? **No**

Project asset requirements:

Assets will be used to promote projects on dev.windows.com/projects

* Images for landing page
  + Should include images of project in use such as screen shots of app, photos of hardware during buildout and in action, etc.
  + Max upload: 4MB. Min. size: 280W x 160H
* Video (if included in scope)
  + MP4 format
  + Follow Windows Brand guidelines
  + Determine where music will come from and if procuring licensed music is required.

Resources:

* [What’s new to Windows 10?](https://dev.windows.com/en-us/getstarted/whats-new-windows-10)

\_\_\_\_Project documentation template \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

FaceTheremin

To inspire you to experiment more with the new media APIs in UWP, we decided to make something very simple, but at the same time highly visual and entertaining.

You’ve probably seen people play Guitar Hero or similar games at parties countless times. It’s a lot of fun! If you’re a real geek and developer, you can make a similar entertainment system for a party in just few hours. All you need is Windows 10 PC with a web camera and Visual Studio.

We’re going to use face detection on a live video stream from a camera to trigger musical events. The easiest way to do this is to consider the video frame as a grid of cells where each cell corresponds to a unique sound. If our app detects a face in a given cell, it should trigger a sound. How?

With the new FaceTracker API you can track a collection of faces at the same time. We are going to use this API as our “facial” musical input.

There are multiple options for making sounds available to developers, including real-time tone generation, but the simplest way is to have a collection of readymade MP3 or WAV audio files. The new Audio Graph API is very easy to use, supports compressed audio formats and uses Windows 10’s low-latency audio pipeline.

Combining these new APIs together, building a FaceTheremin is very easy.

FaceTheremin

Project overview

To inspire you to experiment more with the new media APIs in UWP, we decided to make something very simple, but at the same time highly visual and entertaining.

You’ve probably seen people play Guitar Hero or similar games at parties countless times. It’s a lot of fun! If you’re a real geek and developer, you can make a similar entertainment system for a party in just few hours. All you need is Windows 10 PC with a web camera and Visual Studio.

We’re going to use face detection on a live video stream from a camera to trigger musical events. The easiest way to do this is to consider the video frame as a grid of cells where each cell corresponds to a unique sound. If our app detects a face in a given cell, it should trigger a sound. How?

With the new FaceTracker API you can track a collection of faces at the same time. We are going to use this API as our facial musical input.

There are multiple options for making sounds available to developers, including real-time tone generation, but the simplest way is to have a collection of readymade MP3 or WAV audio files. The new Audio Graph API is very easy to use, supports compressed audio formats and uses Windows 10’s low-latency audio pipeline.

Combining these new APIs together, building a FaceTheremin is very easy.

Getting Started

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
* Any web camera supported by Windows 10, including embedded laptop camera
* Set of audio samples in MP3 format, or use the ones provided in the project solution

Sample features:

**Note:** Features in this app are subject to change.

FaceTracker API:

<https://msdn.microsoft.com/en-us/library/windows/apps/windows.media.faceanalysis.facetracker.aspx>

Audio Graph API:

<https://msdn.microsoft.com/en-us/library/windows/apps/mt203787.aspx>

Running the sample:

Copy to be provided by Microsoft.

Code at a glance:

Let’s look at the actual app making mechanics!

Create a new Blank App (Universal Windows) project, add CaptureElement for streaming video from a webcam and TextBlock for status of the app. By the way, it’s more natural if the video image is inverted horizontally, so you can see yourself 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>

In order to render a live video preview from the webcam, we should use MediaCapture 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();

Once we can see ourselves, it’s time to detect faces. We need some kind of loop that runs a face detection function every N milliseconds. Timer is the easiest way to do this, especially in a “toy” app like 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);

});

}

Now, with faces detected, it’s time to play some sounds. We chose three different instruments: drums, piano chords and some synth tones. Each type of sound has two columns with 8 variations each. AudioGraph connects several input nodes (they can be generated, played from files or MIDI) to one or more output nodes (in our case speakers). First of all, we need to 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;

}

}

We have a separate StorageFile and AudioFileInputNode for each sound. After initialization, we can add an output node and store sounds in 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 know the coordinates of the detected faces, so we know in which cells they are and can finally start audio playback:

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

{

audioFileInputNode.Reset();

audioFileInputNode.Start();

}

That’s it! You can invite your friends to help you make a new hit or at least have tons of fun!

Extend the project:

* Selecting and loading multiple audio packs;
* Visual editor for assigning custom sounds to each cell;
* Possibility to record and share performance;

Additional Resources:

There are massive improvements in UWP for 3rd party developers working with media APIs such as camera and audio playback:

<https://msdn.microsoft.com/en-us/library/windows/apps/mt203788.aspx>

If you haven’t yet had time to checkout the related //BUILD 2015 sessions, you definitely should. Here are just a few:

**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>

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

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