jcoola face-tracker

Introduction

jcoola face-tracker is a light-weight and optimized plugin for Unity game engine enabling games to be furnished with dynamic user-interface interactivity, accelerating the gaming experience making it a great solution for interactive 3D/2D games; throughout the experience you would have fun playing game using your face as an input providing the horizontal and vertical and the depth axis. Unity game developers are now able to incorporate this input-engine into their interactive and production games. The plugin product is available for Windows and Android platforms. The core implementation uses C language and OpenCV.

Specifications

Platforms: Windows, Android

Version: 1.5

API Documentation

Version: 1.5

Public Properties

ProcessRate

Signature:

```
public int ProcessRate;
```

Description:

ProcesRate is the number of times that the tracker is calculated per app fps. The value more than zero overrides the default value.

HighThreadPriority

Signature:

```
public bool HighThreadPriority;
```

Description:

Escalates the priority of the face-tracker thread to a higher one.

OverrideCascade

Signature:

```
public bool OverrideCascade;
```

Description:

face-tracker uses its default haar-cascade xml descriptors. Developer can use his own version of the descriptor by overriding the default cascade. When this property is set to true, the new cascade xml can be assigned to the face-tracker using the public property 'CascadeXML'. Note that this property is the design-only property and should be set only before runtime. To set this property, simply set this property to true in the property inspector.

CascadeXML

Signature:

```
public TextAsset CascadeXML;
```

Description:

When 'OverrideCascade' property is set to true, the new cascade xml can be assigned to this property to override the default cascade xml used by the Face-tracker. Note that this property is the design-only property and should be set only before runtime. To assign the cascade file to this property, simply drag and drop the file into this property in the property inspector.

IsActive

Signature:

```
public bool IsActive { get; private set; }
```

Description:

It is a read-only property and used only to check if the Face-tracker engine is active or not. face-tracker deactivates when it is stopped.

The following properties are provided for advanced developers who have good knowledge in OpenCV. Typically the default values of these properties are recommended for most typical usages of the plugin, but you can tweak them as desired with prior knowledge on them. For more information on these advanced properties please consult the online documentation of OpenCV.

CV_SearchScaleFactor

Signature:

```
public float CV_SearchScaleFactor;
```

Description:

The parameter determines how many different sizes of faces to look for; typically it would be 1.1 for good detection, or 1.2 for faster detection that does not find the face as often.

CV_MinNeighbors

Signature:

```
public int CV_MinNeighbors;
```

Description:

This parameter determines how sure the detector should be that it has detected a face, typically a value of 3 but you can set it higher if you want more reliable faces, even if many faces are not detected.

CV_Flags

Signature:

```
public int CV_Flags;
```

Description:

This parameter allows you to specify whether to look for all faces (default) or only look for the largest face (CASCADE_FIND_BIGGEST_OBJECT). If you only look for the largest face, it should run faster. There are several other parameters you can add to make the detection about one percent or two percent faster, such as CASCADE_DO_ROUGH_SEARCH or CASCADE_SCALE_IMAGE.

```
public enum CV_FLAG {

	CV_HAAR_DO_CANNY_PRUNING =	1,

	CASCADE_SCALE_IMAGE =		2,

	CASCADE_FIND_BIGGEST_OBJECT =	4,

	CASCADE_DO_ROUGH_SEARCH =	8

};
```

CV_MinFeatureSizeX

Signature:

```
public int CV_MinFeatureSizeX;
```

CV_MinFeatureSizeY

Signature:

```
public int CV_MinFeatureSizeY;
```

Description:

This parameter determines the minimum face size that we care about, typically 20 x 20 or 30 x 30 pixels but this depends on your use case and image size. If you are performing face detection on a webcam or smartphone where the face will always be very close to the camera, you could enlarge this to 80 x 80 to have much faster detections, or if you want to detect far away faces, such as on a beach with friends, then leave this as 20 x 20.

Public Methods

Init

Signature:

```
public bool Init(WebCamTexture webCam)
```

Description:

Sets up the face-tracker using a camera reference. Returns Boolean TRUE if the initialization routine is completed successfully.

StartTracker

Signature:

```
public void StartTracker(void)
```

Description:

Starts the Tracker and sets the default operation settings.

StopTracker

Signature:

```
public void StopTracker(void)
```

Description:

Stops the Tracker.

DisposeTracker

Signature:

```
public void DisposeTracker(void)
```

Description:

Is used to dispose and free up the used memory.

FrameWidth

Signature:

```
public int FrameWidth(void)
```

Description:

The horizontal resolution that the face-tracker is operating on.

FrameHeight

Signature:

```
public int FrameHeight(void)
```

Description:

The vertical resolution that the face-tracker is operating on.

FacePosition

Signature:

```
public Vector3 FacePosition(void)
```

Description:

The horizontal (x) and vertical (y) axis and the depth axis (z) can be extracted by mapping the returned x, y and z relative to the current frame resolution and measurement. The return type is Vector3 and as a result it contains the X and Y and Z components. X and Y represents the location where the face was detected,

Where:

0 < X < FrameWidth()

And

0 < Y < FrameHeight()

And the Z component is the Depth/Zoom axis.