Visage Technologies
FACE TRACKING & ANIMATION

Animation & AR Modeling Guide

version 3.0

Contents

1. Introduction

This guide is written for artists and developers working on preparing and importing their own 3D art assets in applications using visage|SDK based on the included samples. It is divided into sections detailing the modeling process for face animation and augmented reality applications based on the Facial Animation and Virtual Try-on samples. The process consists of preparing the assets in an authoring tool and importing the assets into the application.

The main sections of this guide are written in a generic way so they can be applied regardless of which authoring tool is used. The sections that deal with importing the assets into final applications are specific to the runtime engines for which integration with visage|SDK is already provided – other engines can be integrated and supported independently.

2. Face animation modeling guide

This part of the guide explains how to prepare 3D models for face tracking driven animation in applications based on visage|SDK facial features tracking. Action Units and head transform provided by visage|SDK can be used for driving blendshape and bone animation as shown in the facial animation sample for Windows, iOS and Android. The guide shows how to prepare models for such applications.

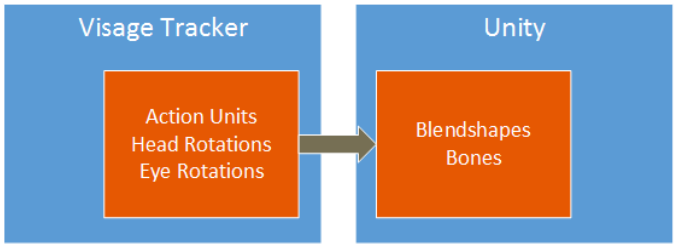

Fig. 1 – Action Units are used to drive blendshape animation in Unity

2.1. Creating blendshapes

Blendshapes are a method of defining animation often used for animating facial expressions. They are also known as morph targets or just morphs. A blendshape is defined using two meshes – base and target mesh. These two meshes need to have the same number of vertices and the same topology. Consult your authoring software manual for details on creating blendshapes.

A blendshape animation will morph the base mesh into the target mesh based on the weight value ranging from [0-1]. A value of 0 means that the base will stay the same and the value of 1 means the vertex positions in the base mesh will match the vertex positions in the target mesh. Values between 0-1 will linearly interpolate vertex positions between the base and the target mesh. This allows creating animations of specific expressions such as opening the mouth or raising an eyebrow which can be triggered by visage|SDK's Action Units and played simultaneously.

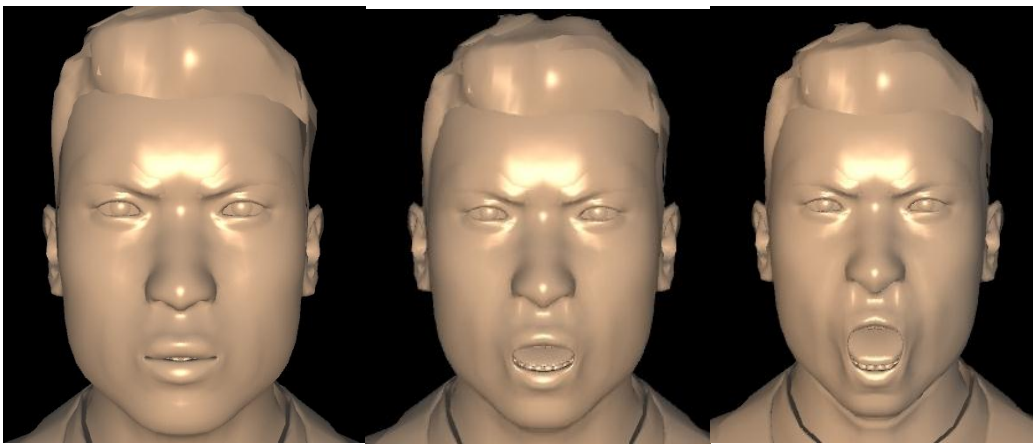

Fig. 2 - Blendshape animation with the base mesh on the left, mesh with blendshape amount 0.5 in the middle and the target mesh on the right.

A separate blendshape should be created for every mesh you want to animate with an action unit. For example if you wish to animate a character's face, teeth and tongue using the au_jaw_drop action unit you should create three blendshapes; one on the face mesh, one on the teeth mesh and one on the tongue mesh.

Animation can be displayed differently in the authoring tool and the final application depending on whether the application supports blendshape normalization groups – it is best to either disable normalization in the authoring tool or create a separate normalization group for each blendshape. If both blendshapes and bone animation are

used it is worth noting that blendshape deformations should come before bone deformations in the deformation chain.

Action units can be configured using the tracker configuration file. Consult the Visage Tracker Configuration Manual for more info on configuring action units. It is recommended to create blendshapes for animating at least these action units if using the prepackaged asymmetric configuration files:

- au_jaw_drop
- au_lip_stretcher_left
- au_lip_stretcher_right
- au_left_outer_brow_raiser
- au_left_inner_brow_raiser
- au_right_outer_brow_raiser
- au_right_inner_brow_raiser

2.1.1. Blendshape suggestions

The “Jones” model included in the Facial Animation Unity Demo contains the blendshapes for the aforementioned action units, along with:

- au_jaw_x_push
- au_jaw_z_push
- au_reye_closed
- au_leye_closed
- au_nose_wrinkler

The images below may serve as a visual aid in creating your own blendshapes. Note the omission of symmetrical counterparts – for example, au_lip_stretcher_right is simply a mirrored au_lip_stretcher_left.

Fig. 3 – neutral expression

Fig. 4 - au_left_inner_brow_raiser

Fig. 5 - au_left_outer_brow_raiser

Fig. 6 – au_jaw_drop

Fig. 7 – au_jaw_x_push

Fig. 8 – au_jaw_z_push

Fig. 9 – au_ lip_stretcher_left

Fig. 10 – au_nose_wrinkler

Fig. 11 - au_reye_closed

2.2. Binding configuration

A configuration system is in place in the facial animation example for creating simple action unit to blendshape animation bindings using a binding configuration. To use the binding configuration assign it to the Binding Configurations property of the Visage Tracker object in the sample. Below are instructions for creating a binding configuration. An example binding configuration file is included in the facial animation sample (kenchi.bind.txt).

Lines in the binding configuration beginning with a '#' are comments.

Each non-comment line is a semicolon delimited list of values in the following format: au_name; blendshape_identifier; min_limit; max_limit; inverted [optional]; weight [optional]; filter_window [optional]; filter_amount [optional].

Optional parameters may be ommited, but only the last ones. For example, it is not possible to omit 'inverted' and 'weight' values and not omit 'filter_window' and 'filter_amount' values.

Last two parameters (filter_window and filter_amount) control the smoothing filter. This is similiar to the smoothing filter used in the tracker, but these can be defined for each action unit separately. Stronger filtering will stabilize the animation but can also introduce delay.

Value descriptions:

au_name	The name of the action unit used to drive the specified blendshape. Example: "au_jaw_drop".
blendshape_identifier	Platform-specific blendshape identifier. Example for Unity-based implementations: "tina_face:1". The part before the colon is the game object name with the SkinnedMeshRenderer component you wish to animate, the part after is the blendshape index.
min_limit	Lower action unit limit. Consult the action unit limits defined in the tracker configuration file for appropriate values. Example: "-0.16".
max_limit	Upper action unit limit. Consult the action unit limits defined in the tracker configuration file for appropriate values. Example: "0.51".
inverted	Inverts the action unit values. Valid values are "0" and "1". Value of "1" means the action unit will be inverted. This is useful when a blendshape works in the direction opposite of the action unit. For example, action unit is "au_left_inner_brow_raiser" and the blendshape works by lowering the inner brow. Default value is "0".
weight	Action unit weight. Action units are multiplied with this value. Default value is "1.0".
filter_window	Action unit history size used to filter values. Valid value range are integers from "1" to "16". Default value is "6".
filter_amount	Strenght of the filter. Valid value range is from "0.0" to "1.0". Default value is "0.3".

2.3. Importing the model into the final application

This section covers importing models into face animation applications built in Unity. Visage|SDK includes a sample implementation of a face animation application built in Unity – the Facial Animation Unity Demo, and these instructions are compatible with that application. It is possible to build similar applications in other 3D engines using visage|SDK.

2.3.1. Unity (used in visage|SDK for Windows, iOS and Android)

Unity supports blendshape animations from version 4.3.

Export the model from an authoring tool in a FBX file with included blendshapes, copy it to the project's Assets folder and enable blendshapes in the model's inspector window. For a list of supported formats consult the Unity's how-to guide on importing here.

Note that Unity does not support blendshape weights, existing weights should be baked into the blendshape target before exporting. Unity doesn't support normalization groups either so the final result can be different than in the authoring software. Blendshape animations can be previewed in Unity by adding the model in the scene and changing the blendshape values.

3. AR modeling guide

This part of the guide explains how to prepare 3D models to be used in augmented reality (AR) applications based on visage|SDK head tracking. As an example, the guide shows how to prepare eyeglasses for Virtual Try-on application. The same principles may be applied to other objects to be used in AR applications.

3.1. Size and positioning

Real-life size, in meter units.
X positioning: center of glasses at X=0
Y positioning: visual axes at Y = 0
Z positioning: eyeball front edge at Z = 0 (therefore lenses at approx. Z = 0.02 - 0.025 m depending on glasses model)

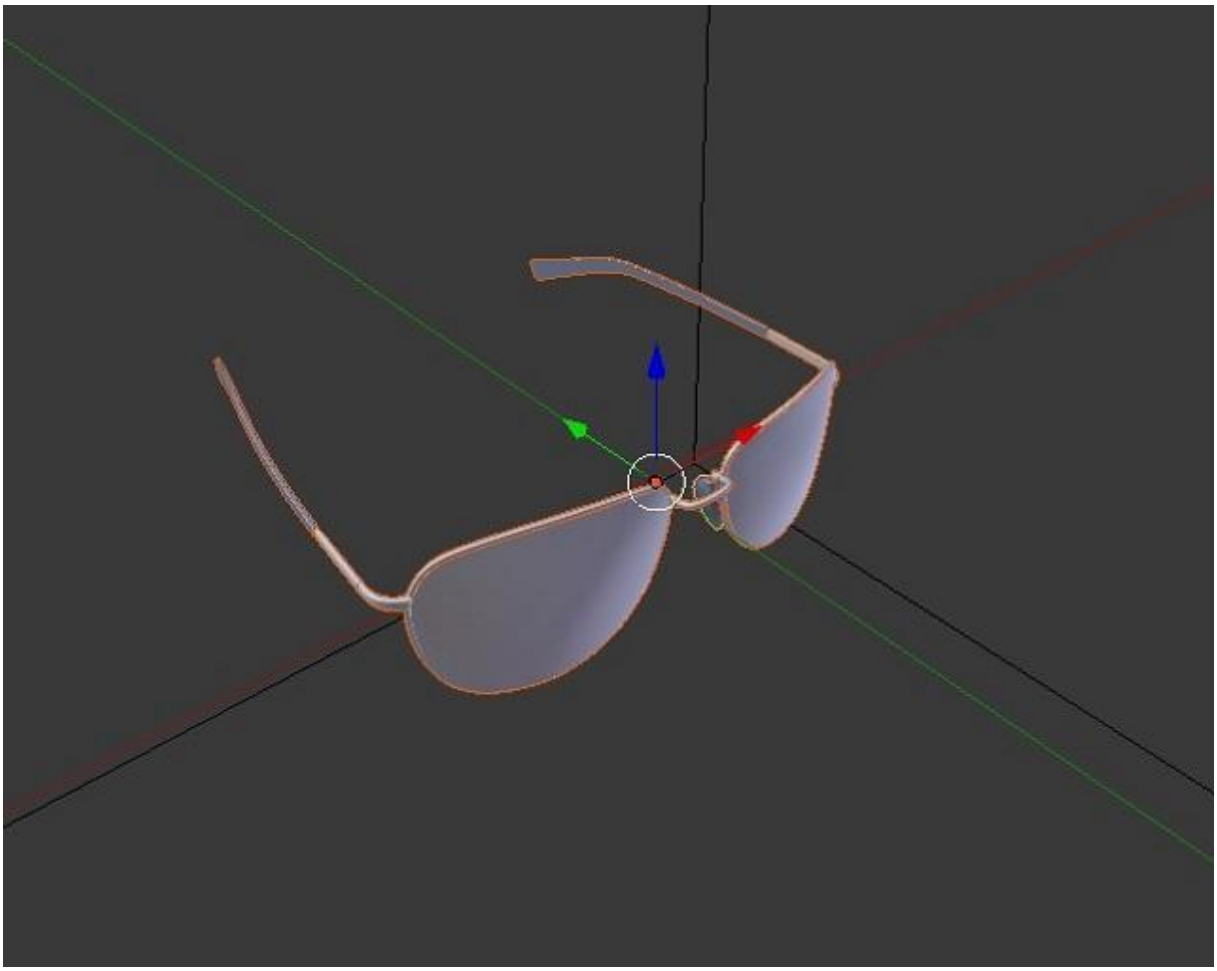

Fig. 12 - Size and positioning of the glasses in 3D authoring tool.

3.2. Occlusion

If occlusions are not handled, glasses would always appear on top of the face like this:

Fig. 13 - No occlusion

In reality, parts of the glasses are occluded by the head, nose or ears. To achieve the same effect in augmented reality, an occlusion mask is used. This is an object shaped roughly like a human head and placed inside the glasses. The material of this object is called *occluder_mat*. At runtime, the occlusion mask covers the object behind it but is not itself visible, achieving the desired effect. The next two images show the occlusion mesh highlighted as wireframe (left), and the final effect (right).

Fig 14 - Occlusion final(wireframe)

Fig 15 - Occlusion final

3.2.1. Importing the occlusion mask

Occlusion mask is available: models/occlusion_mask.obj. When imported into the scene, the mask should automatically be correctly sized and positioned, as follows:
Real-life size, in meter units.
X positioning: point between the eye centers at X=0
Y positioning: eye center's at Y = 0
Z positioning: eyeball front edge at Z = 0
The occlusion mask must not be moved or scaled.

3.2.2. Adjusting the occlusion mask

Small adjustments to the mask can be made so it better fits the specific glasses model.

Nose adjustment

Nose can be widened in order to fit the glasses' nose pads as close as possible. Following images illustrate the correct fitting of the nose:

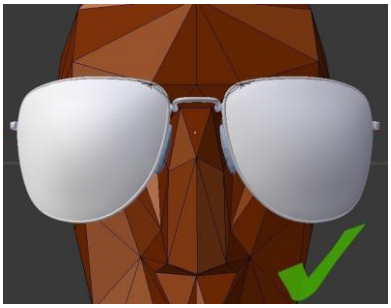

Fig 16 - Correct nose adjustments

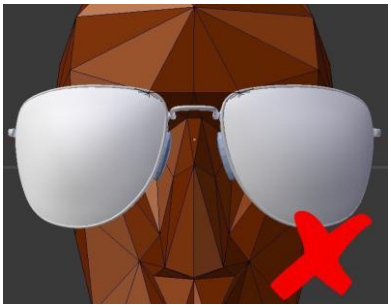

Fig 17 - Nose too narrow

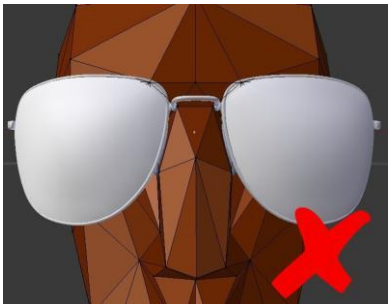

Fig 18 - Nose too wide

Head sides

If the occlusion mask is too wide or too narrow, it can be adjusted by moving the sides of the head or adjusting them so that the temples of the glasses fit closely to the mask.

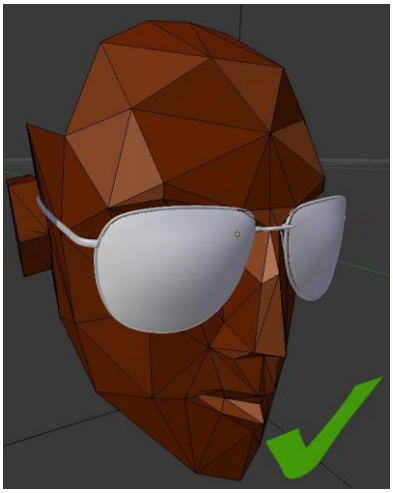

Fig 19 - Correct head sides adjustments

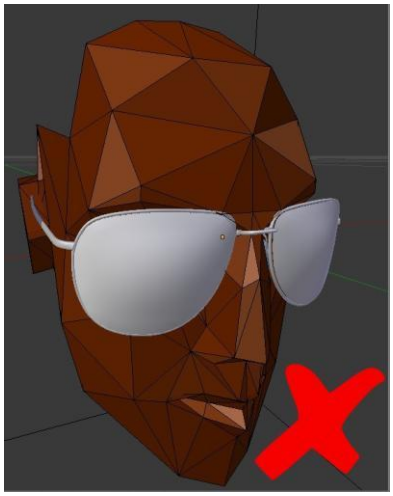

Fig 20 - Head sides too narrow

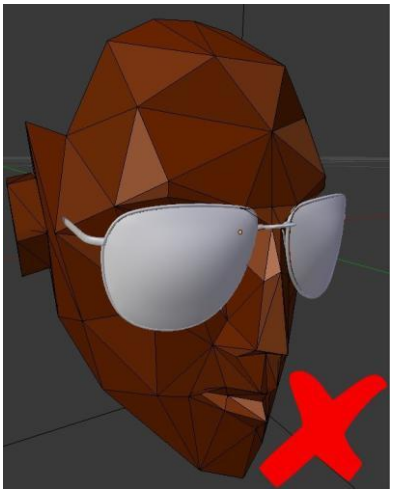

Fig 21 - Head sides too wide

Ears

The ears of the occlusion mask should fully cover the back part of the temples.

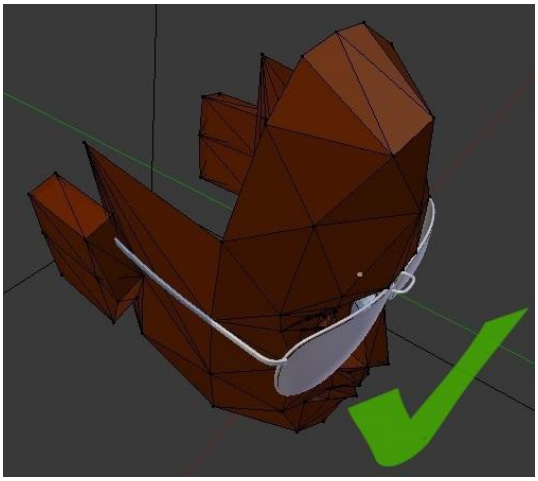

Fig 22 - Correct ears adjustments

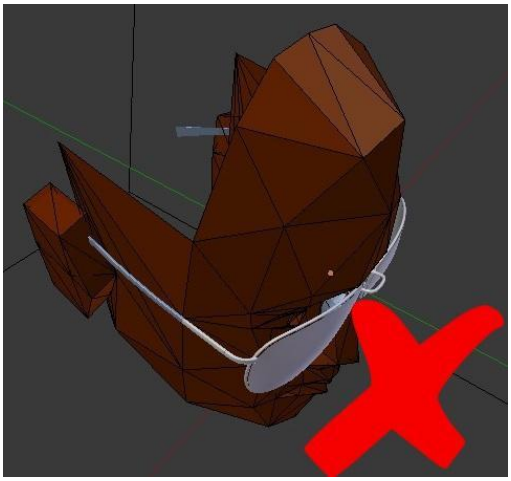

Fig 23 - Temples not completely occluded

3.3. Importing the model into the final AR application

This section covers issues of file formats and conversions. The two subsections cover the two 3D platforms typically used to build final AR applications using visage|SDK - Three.js and Unity.

3.3.1. Three.js (used in visage|SDK for HTML5)

Models need to be in OBJ format with a corresponding MTL material file. Most 3D authoring tools provide a way to export an object to OBJ and MTL.

Resulting OBJ file must contain both the glasses and the occlusion mask. Occlusion mask mesh should be grouped under *occluder_mat* name. For example, in Blender this is done by naming the material of the mesh *occluder_mat* and exporting to OBJ with setting *Material Groups* checked.

Editing the MTL file

Materials are defined in the MTL file, usually generated from the materials in the authoring tool. After exporting advanced users can do fine tuning by editing the file. For syntax reference see here. Textures used in the MTL file should be placed in the path relative to the MTL folder. Supported texture formats include JPG, TGA and PNG.

3.3.2. Unity (used in visage|SDK for Windows, iOS, Android)

Models are typically built in a 3D authoring tool such as 3ds Max or Blender, then imported into Unity3D. See a tutorial here for a list of supported formats and details regarding the import process. After importing process is complete **check that the size and positioning of the model and the occlusion mask are consistent with the instructions in sections** 3.1 **and 3.2.1.**

Unity package contains an occlusion shader that needs to be assigned to the *occluder_mask* object. Occlusion shader is available in the unity scene in: Materials/. Unity package for following versions can be found:

- Windows - Samples\OpenGL\data\VisageTrackerUnity\VisageTrackerUnity.unitypackage
- Android - Samples\Android\VisageTrackerUnityDemo\VisageTrackerUnityDemo.unitypackage
- iOS - Samples\iOS\VisageTrackerUnityDemo\VisageTrackerUnityDemo.unitypackage

Note that most materials will need to be recreated because Unity uses its own shaders.