by Kalagaan

VertExmotion

What is VertExmotion?

VertExmotion is a shader based softbody system coupled with a procedural animation system.

You can easily animate parts of your mesh like hair, cloths, fatness... within Unity editor !
All elements will move with a procedural way, so no need to add bones for everything !

Because it's shader based, it's really fast !
Because you don't have time to waste, it's super easy to use !

- Add a single component.
- Paint what you want to see moving !
- Add sensors and set motion properties
- Hit play and enjoy !

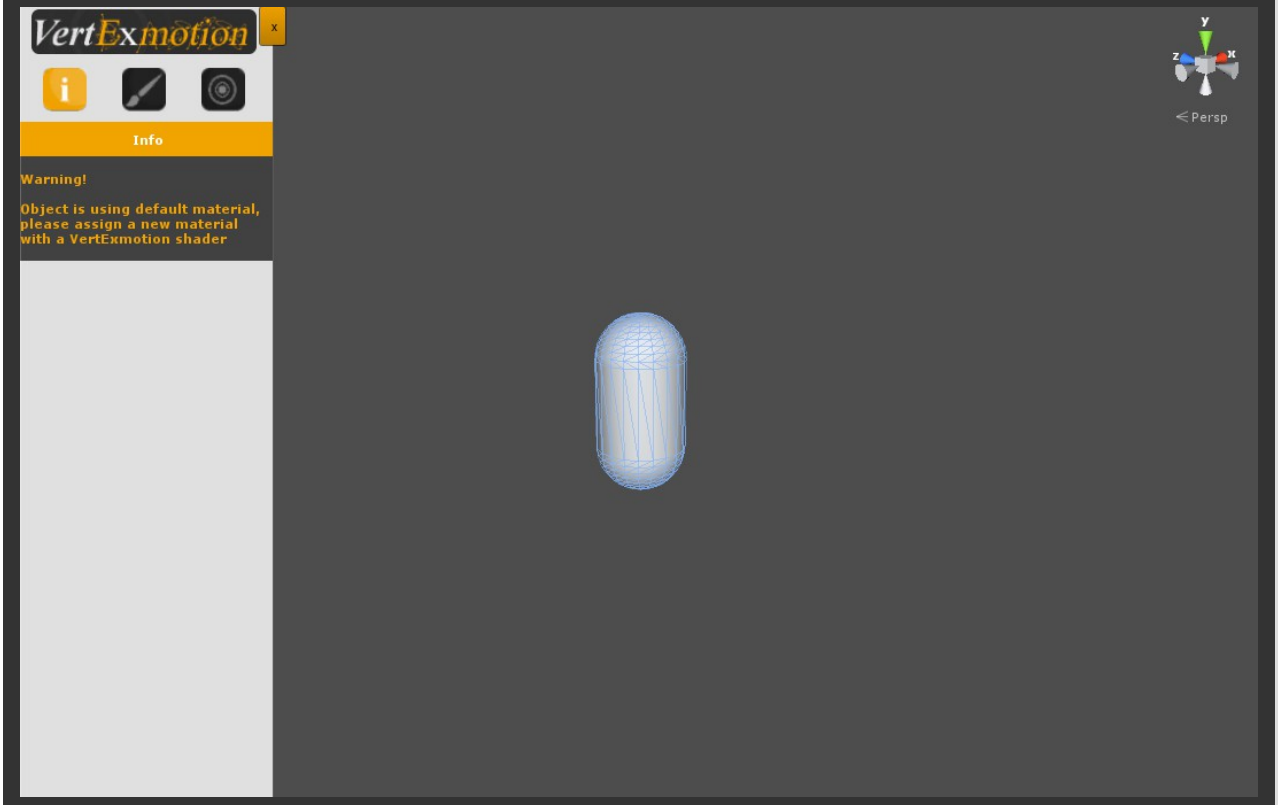
All parts will follow the mouvement of the mesh !
Compatible with more than 80 Unity builtin shaders.
Easy to include in your custom shaders.
Works with static mesh or skinned mesh.

Tested on PC/MAC/iOS/Android/Webplayer

How to use it ? (Tutorial)

The easiest way to learn is to follow a tutorial, let's go!

- First, select your mesh, in this case: the basic capsule.
- Add VertExmotion component (menu->Component->VertExmotion).
  VertExmotion panel appears.



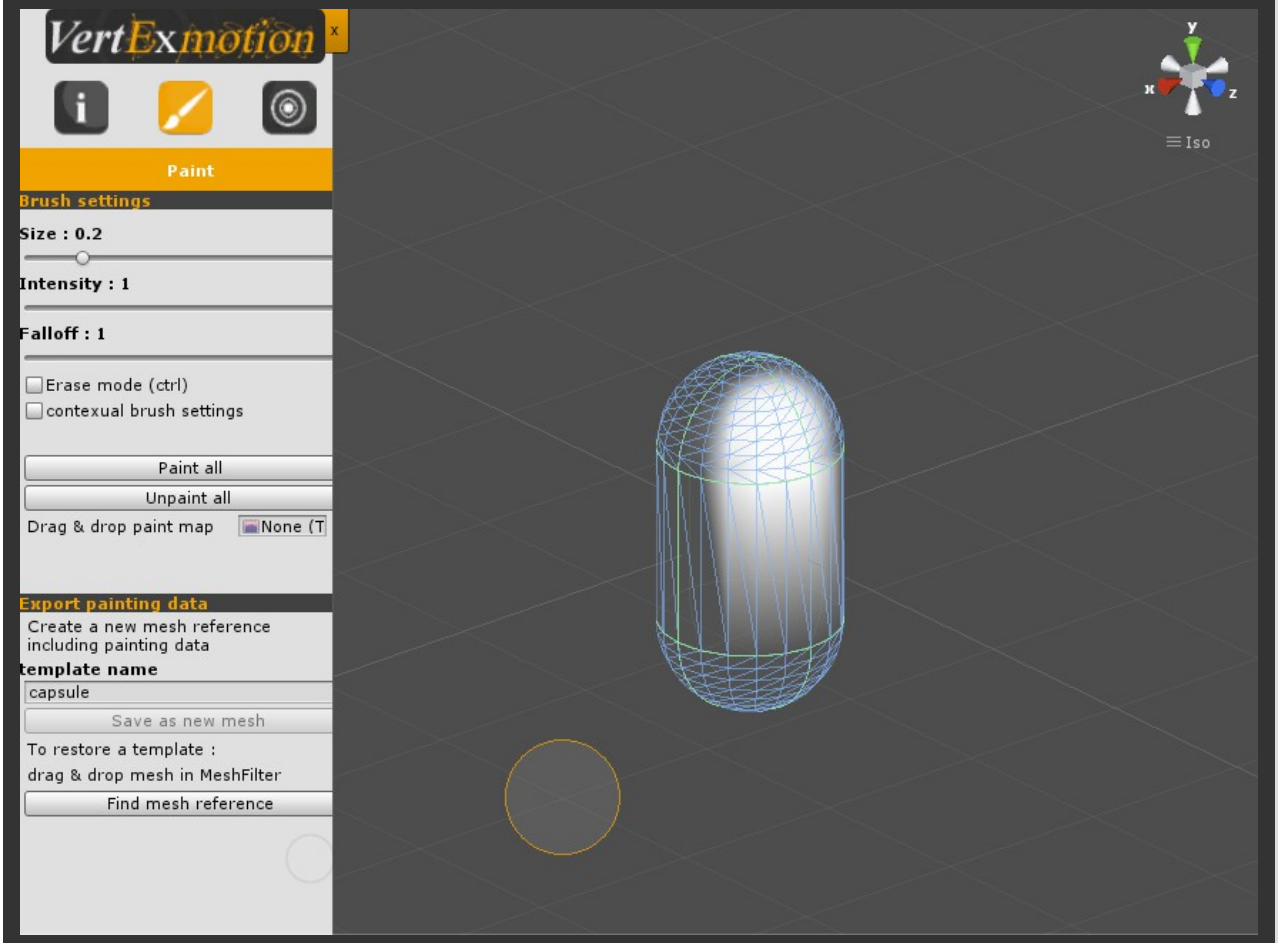
  This mesh use the default material, you have to create a new one.
- Drag & drop the material on the mesh.

  VertExmotion is a shader based system, the material must use one of the compatible shader.

- Press 'Fix material' button or choose a VertExmotion shader in the material list.
  The shader is replaced by a compatible one.
  Now some help appears in the info panel.
  Time to paint !

Paint Settings

- Press the brush icon.
- Set up size, intensity and falloff with sliders.
- Paint on the mesh



- Press ctrl to switch to erase mode.
  White vertices will be ready for motion.
  Black vertices will be static like a standard mesh.
  Intensity will affect motion.

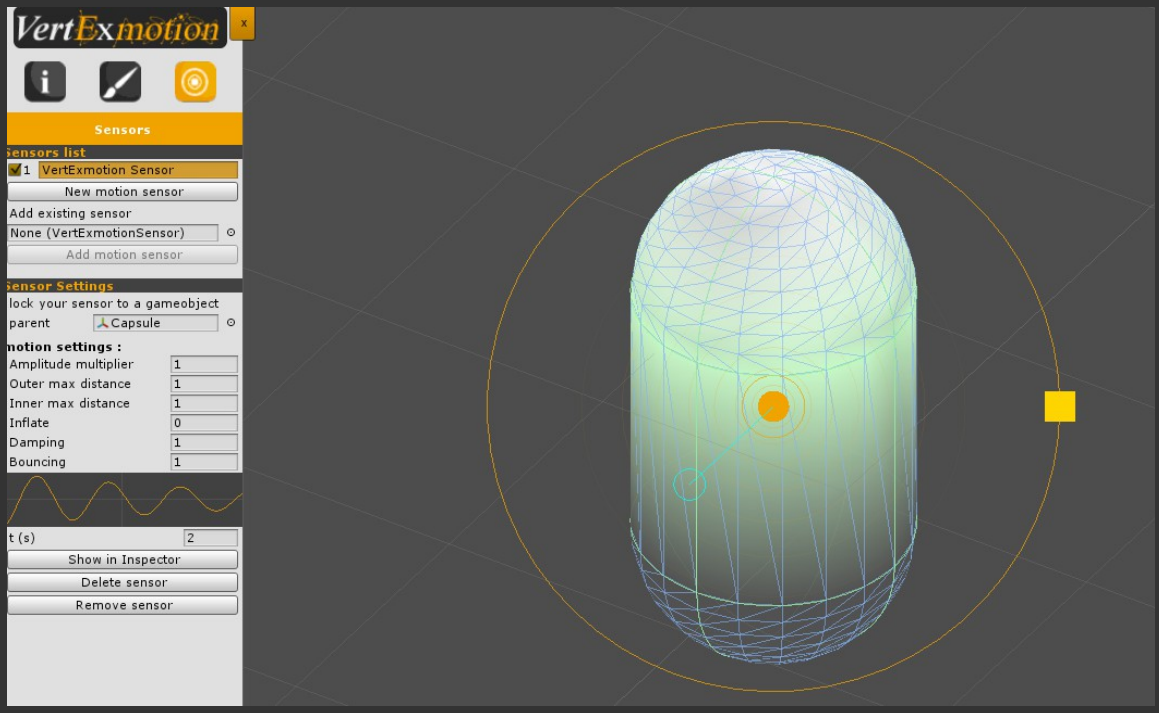
You can export painting data as a new mesh. This is very usefull, because you can share painting information between different prefabs, or save different painting templates.
Exporting as a new mesh will also enable mesh sharing between the instances and optimize memory.

- Enter a template name : 'capsule'
- Click 'Save as new Mesh'
  Now, the new mesh reference is saved in another prefab.
  Painting data are linked to this asset, so you don't need t save it again.

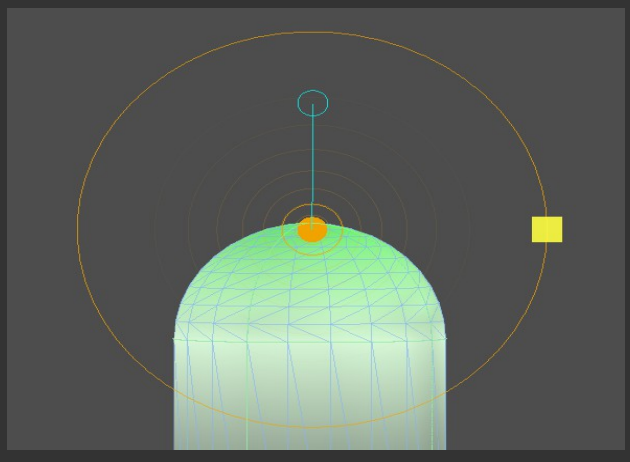
Note : To import a reference mesh, drag&drop it from the project window to the import field.

Sensors Settings

- Press sensor icon.
- Press 'New motion sensor'
  A sensor defines how mesh parts will move.



- Drag the sensor on the top of the mesh.
  Green vertices are in sensor range.
- Try to change it by draging the yellow square handler.

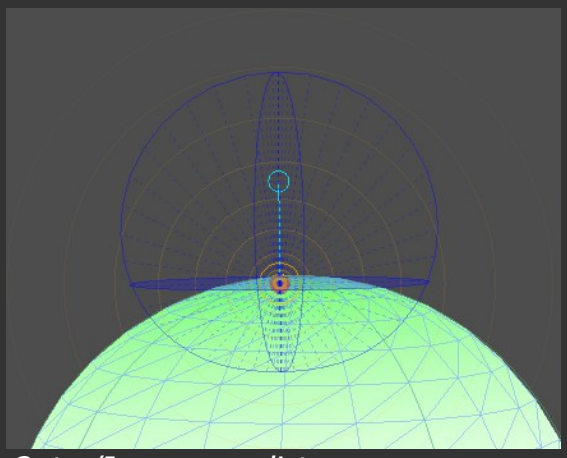


  Blue line is the sensor orientation
- Keep it out of the mesh by draging blue circle handler.
- Hit the play button.
- Move your mesh in sceneview.
  That's it! You have made your first jelly capsule!

- Now, have a look to these sensor settings

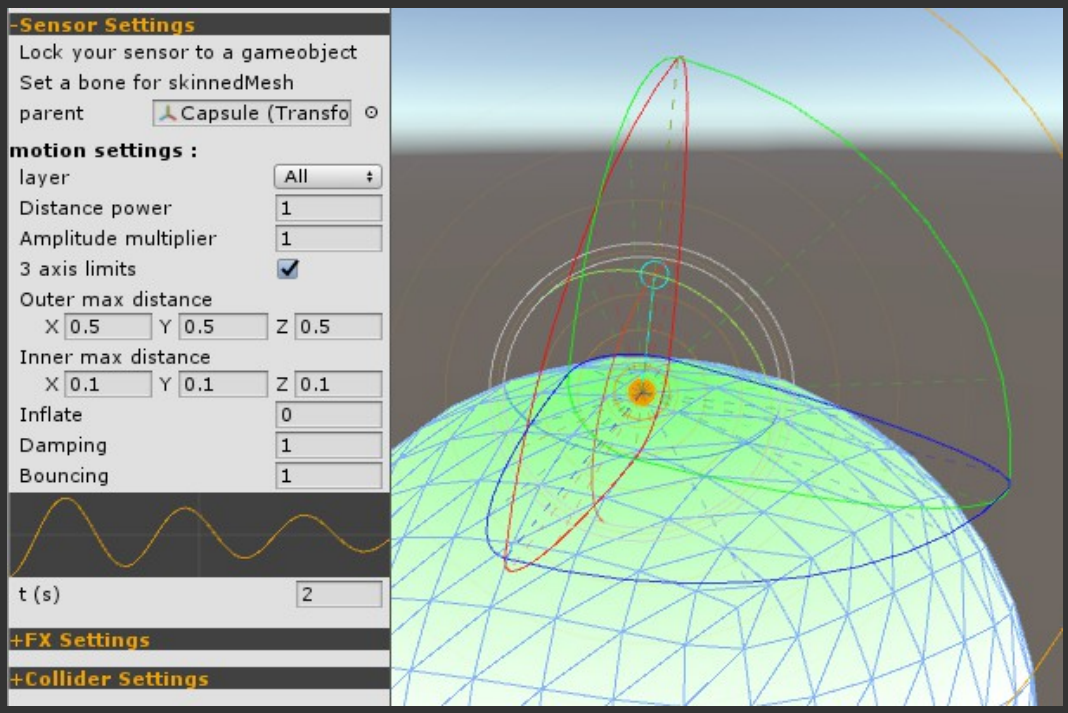
The blue circles define the limits of the vertices displacement for the current sensor. The standard mode is based on the Z axis of the sensor transform.

The 'outer max distance' parameter set the limit according to the sensor direction, and the 'inner max distance' parameter set the opposite side limit.



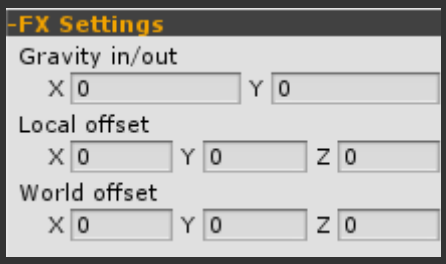
Outer/Inner max distances

The '3 axis limits' mode is more accurate, you can set the Outer/Inner limits for each axis.



- Try to change settings and move your object.

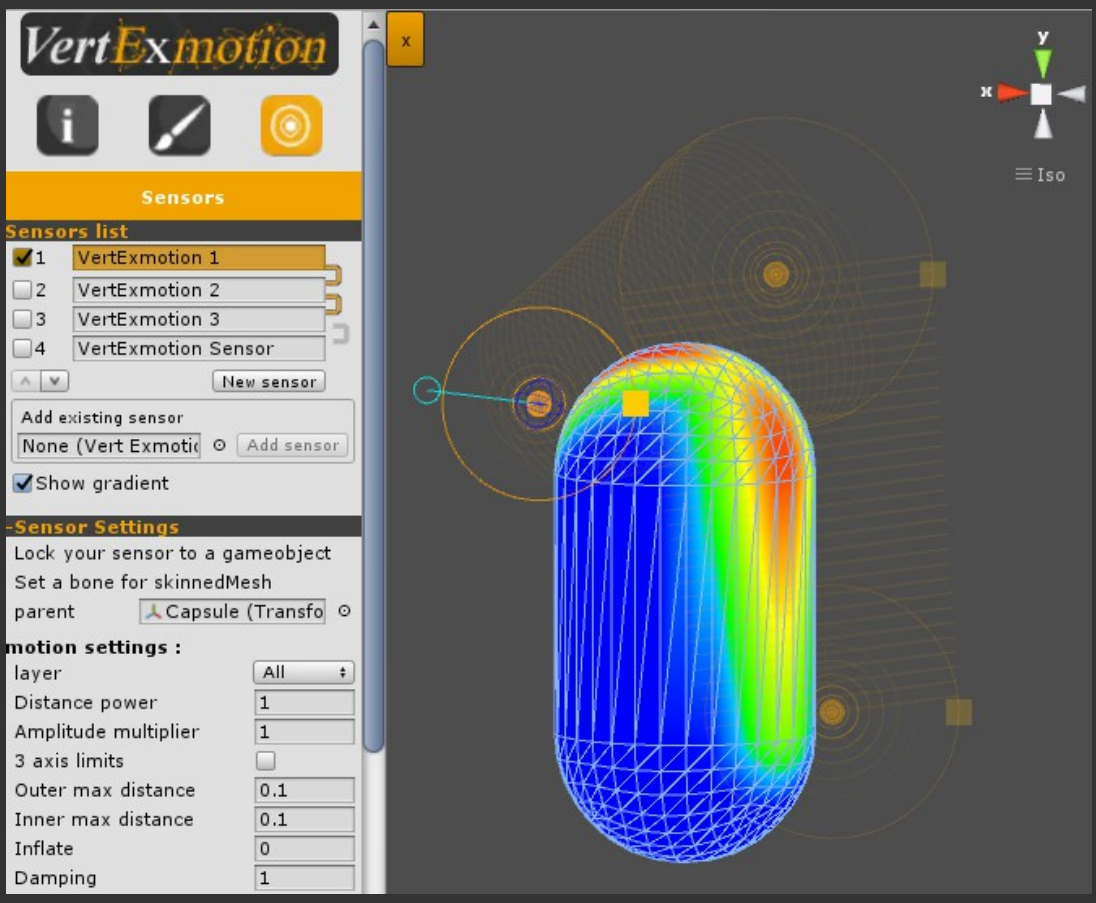
Parent : the parent of sensor transform (set nearest bone for SkinnedMeshRenderer).
Layer : the layer of the sensor (default : all)
Distance power : The sensor attraction power (default : 1)
Amplitude multiplier : amplify or reduce the motion amplitude.
Outer max distance : max vertex displacement in the sensor direction
Inner max distance : max vertex displacement in the opposite of sensor direction.
Inflate : inflate vertices from sensor position.
Damping : increase to stabilise motion.
Bouncing : increase to amplify bounce.
t(s) : change curve time for visualisation only.



Gravity in/out : gravity (Physics.gravity) applied on vertices.
Local offset : translation offset in sensor space.
World offset : translation offset in world space.

Sensor's Link

Each sensors can be linked to the next one in the list. The link behavior is activated/deactivated by pressing the link icon .



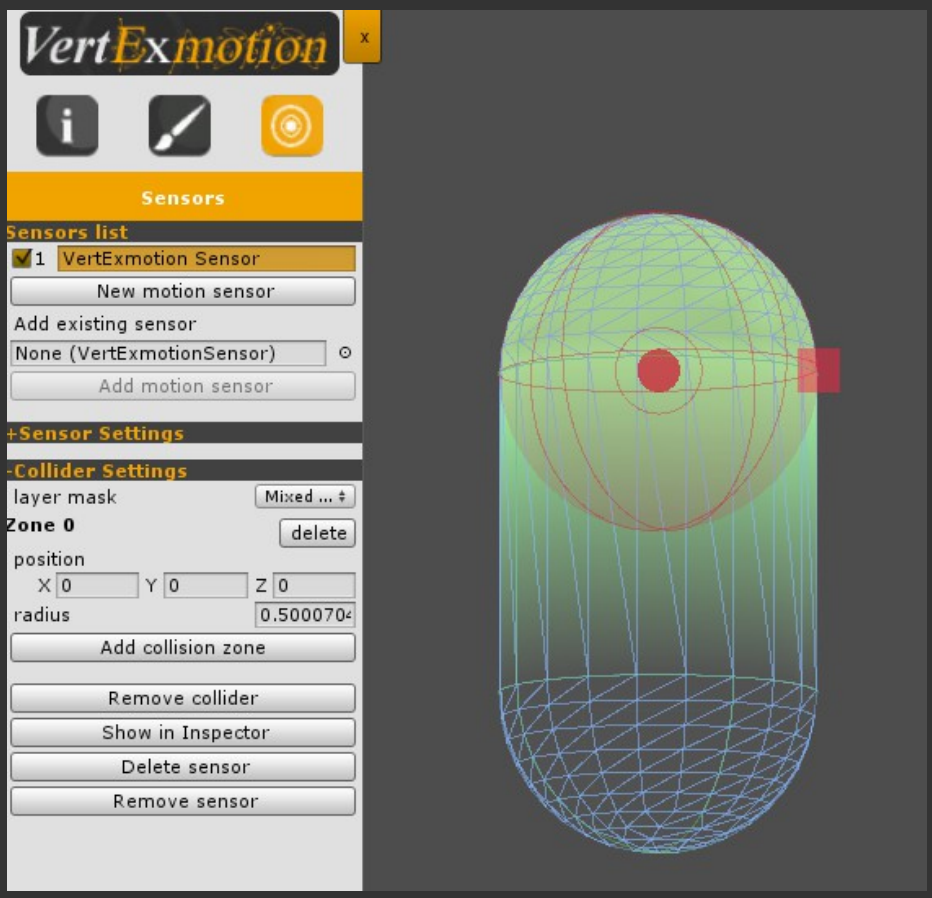
When two sensors are linked, all the vertices included in the range of the capsule (dual-sphere) will receive the motion data. The deformation informations are merged according to distance of the two sensors.

The sensor order can be modified with the up and down buttons .

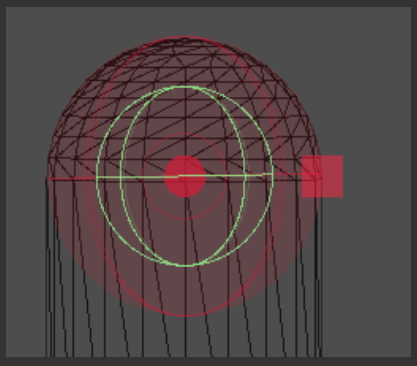
How to setup the collision system ?

Collision Zone

Each sensor can interact with physics by adding some collision zones.



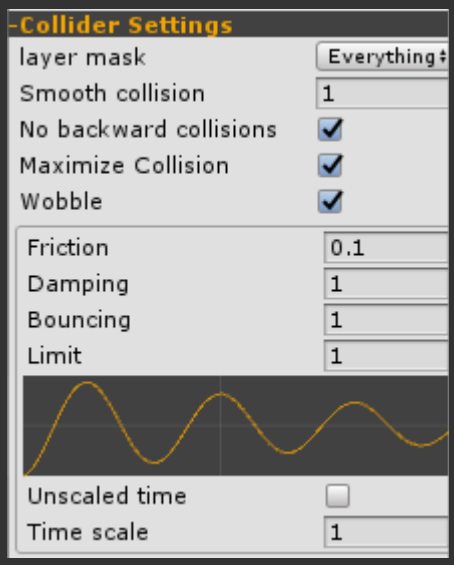
- Click on 'collider settings'
- Press 'Add Collider'
- Set the physics layer mask to collide with.
- Add a new collider zone.
- Change the position and the radius.
- Add other collision zones to suit the mesh surface.



If the mesh has already a physic collider (SphereCollider, BoxCollider...), collision zones must be larger than collider area.

Collider Settings

The collider settings define how the deformations will be applied when a collision is detected.



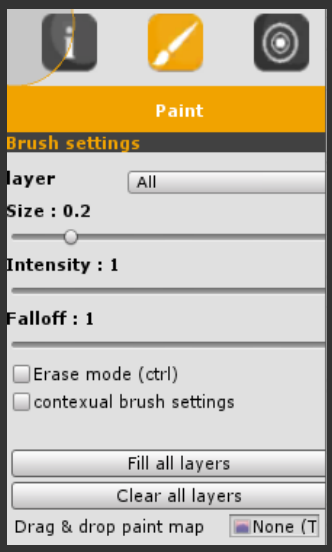
- **Layer mask** : Only the physics colliders in the defined layers will be activated.
- **Smooth collision** : Smooth the motion system for a better animation.
- **No backward collisions**: Disable collisions according to the sensor orientation.
- **Maximize collision** : Use the maximum area of the collision zone. Disable this parameter for using only the half sphere for detecting collisions.
- **Wobble** : Add a procedural animation after collisions.

Wobble parameters :

- **Friction** : increase it to add a friction factor when a collision is detected.
- **Damping** : increase it to stabilise the motion.
- **Bouncing** : increase it to amplify the bounce.
- **Limit** : the maximum bouncing value.

How to setup the sensor's layers

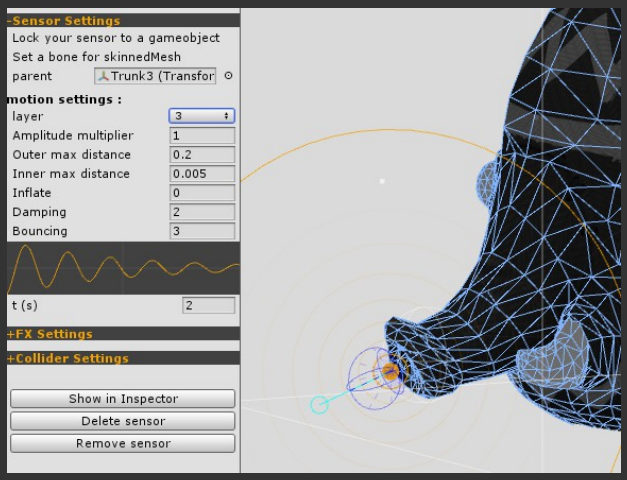
There's 3 layers of painting data on a mesh, this provide a way to avoid some sensors to overlap on unwanted mesh parts.

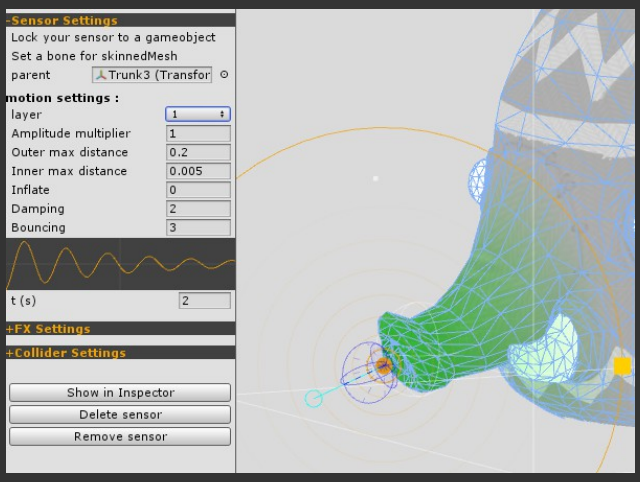


You can paint on an individual layer by selecting the layer id in the dropbox, or paint on all the layers at the same time by selecting 'All'.

When the painting information are done, each sensor can be assign to one layer, or to all the layers.

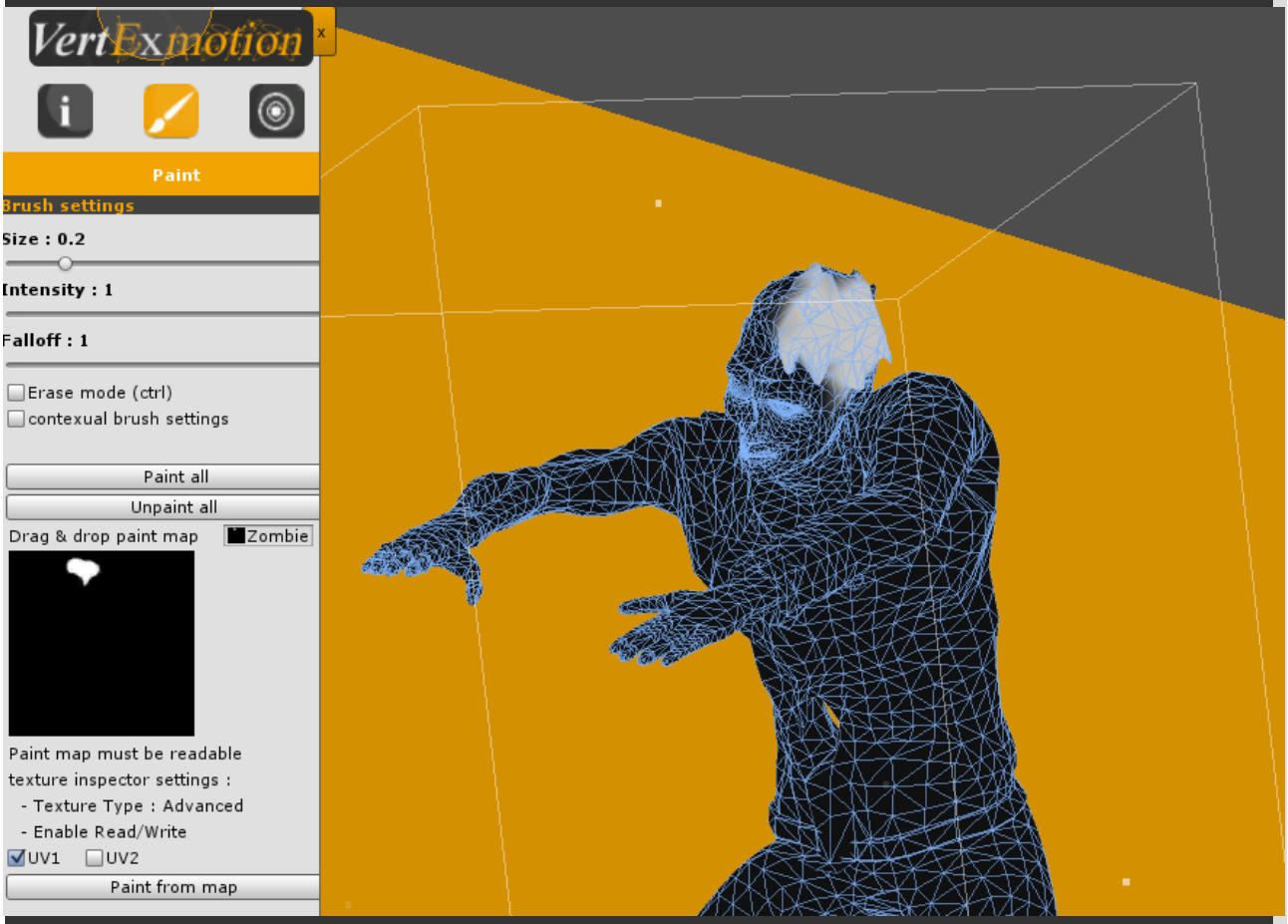
In the following case, the paint informations are not available on the 3rd layers, the sensor should be assign to '1' or 'All'.





How to import paint data from a map

Sometime painting on a mesh is very difficult, importing a texture for painting data is a real time saver.

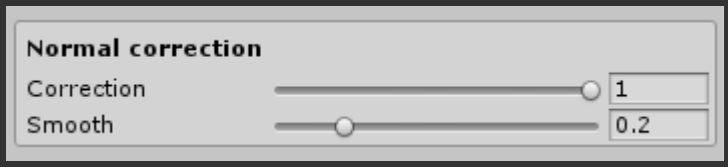


Here a tutorial :

- *Create a copy of the diffuse map.*
- *Open it in your favorite software (photoshop, gimp...)*
- *create a black layer with 50% alpha.*
- *paint a white mask for softbody parts.*
- *Set layer to 100% alpha (only mask is visible)*
- *Save it in the unity project*
- *Select image file in Project panel.*
- *In the inspector panel set 'Texture type' to '**Advanced**'*
- *Check '**Read/Write** Enabled'*
- *Select VertExmotion object*
- *Open Paint panel*
- *Drag & drop paint map*
- *Select you UV channel (UV1 by default)*
- *Click 'Paint from map'*

Normal correction

When the vertices move, the normal could change according to the new face orientation. To enable the normal correction you have to set the correction parameter to 1, if there's some visual artefacts, you can decrease the value to fix them.
The smooth parameter define a threshold for smoothing normals according to the distance from the sensor position.



How to share sensors between mesh ?

If you want to share a sensor between different meshes, you can add an existing sensor instead of creating a new one. The Vertexmotion components will share sensors settings. This is usefull if you want to synchronize the body deformation with clothes.

How to include VertExmotion in my custom shader ?

First, copy your shader in another file.
*Change the name of the shader to '**VertExmotion/shadername**' for editor compatibility.*

- *For surface shader you have to modify these lines in your shader :*

```
#pragma surface surf Lambert alpha vertex:vert addshadow
#include "Assets/VertExmotion/Shaders/VertExmotion.cginc"
void vert (inout appdata_full v) {VertExmotion( v );}
```

- *If your shader has already a vertex function, add theses lines :*

```
#include "Assets/VertExmotion/Shaders/VertExmotion.cginc"
void vert (inout appdata_full v) {
        VertExmotion( v );
        //original shader code
}
```

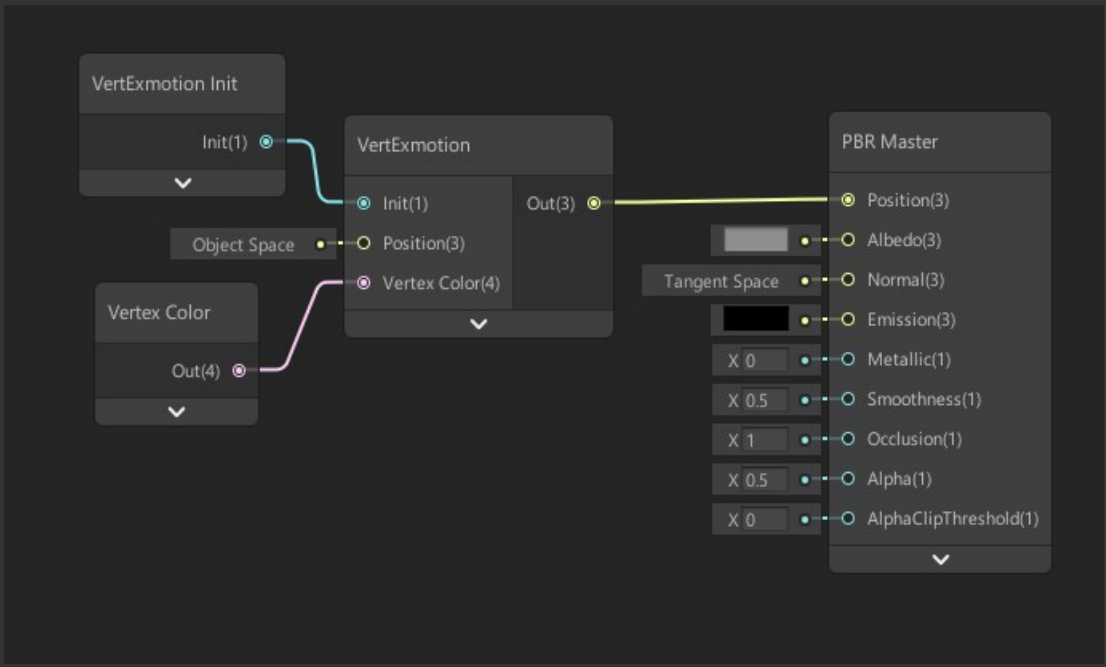
- *If the vertex function don't use appdata_full add theses lines :*

```
#include "Assets/VertExmotion/Shaders/VertExmotion.cginc"
void vert (inout appdata v) {
        v.vertex = VertExmotion( v.vertex, v.color );
        //original shader code
}
```

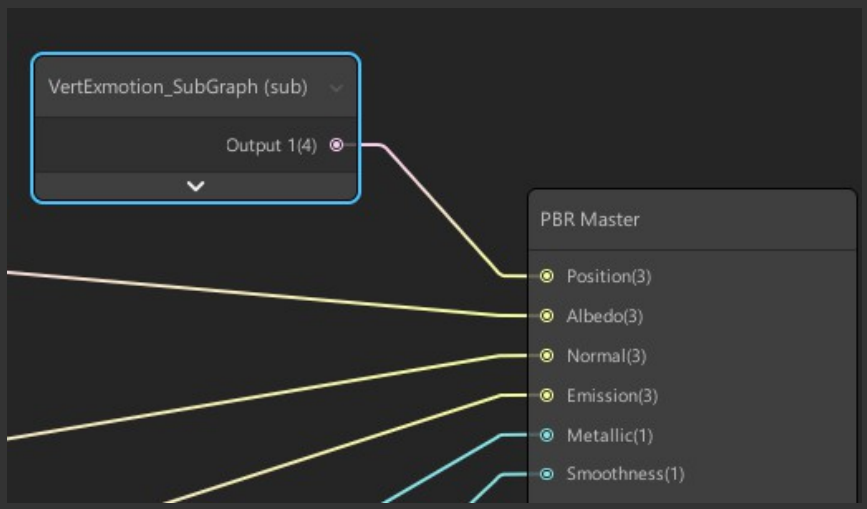
How to include VertExmotion in Unity Shader Graph

The VertExmotion package include built-in nodes for Unity Shader Graph.
Unpack the file '**VertExmotion/Addon/VertExmotion_ShaderGraph_X_Nodes**'.

ShaderGraph 3.X :
- Add the nodes '**VertExmotion**' and '**VertExmotion Init**'

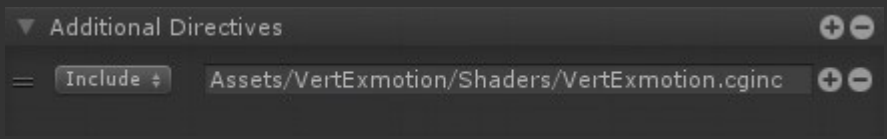


ShaderGraph 4.X :
- Drag & drop the VertExmotion Subraph (VertExmotion/Editor/ShaderGraph/)

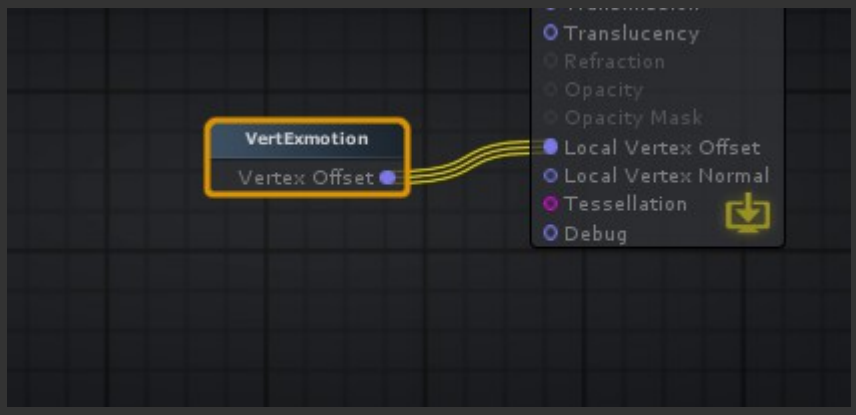


How to include VertExmotion in Amplify Shader Editor

The VertExmotion package include built-in nodes for ASE.
- Unpack the file '**VertExmotion/Addon/VertExmotion_AmplifyShaderEditorNodes**'.
- Open your shader within the ASE window.
- Set the **Vertex Output to** '**Relative**' in the general section.
- Add the VertExmotion include file in the **'Additional Directives' section**.



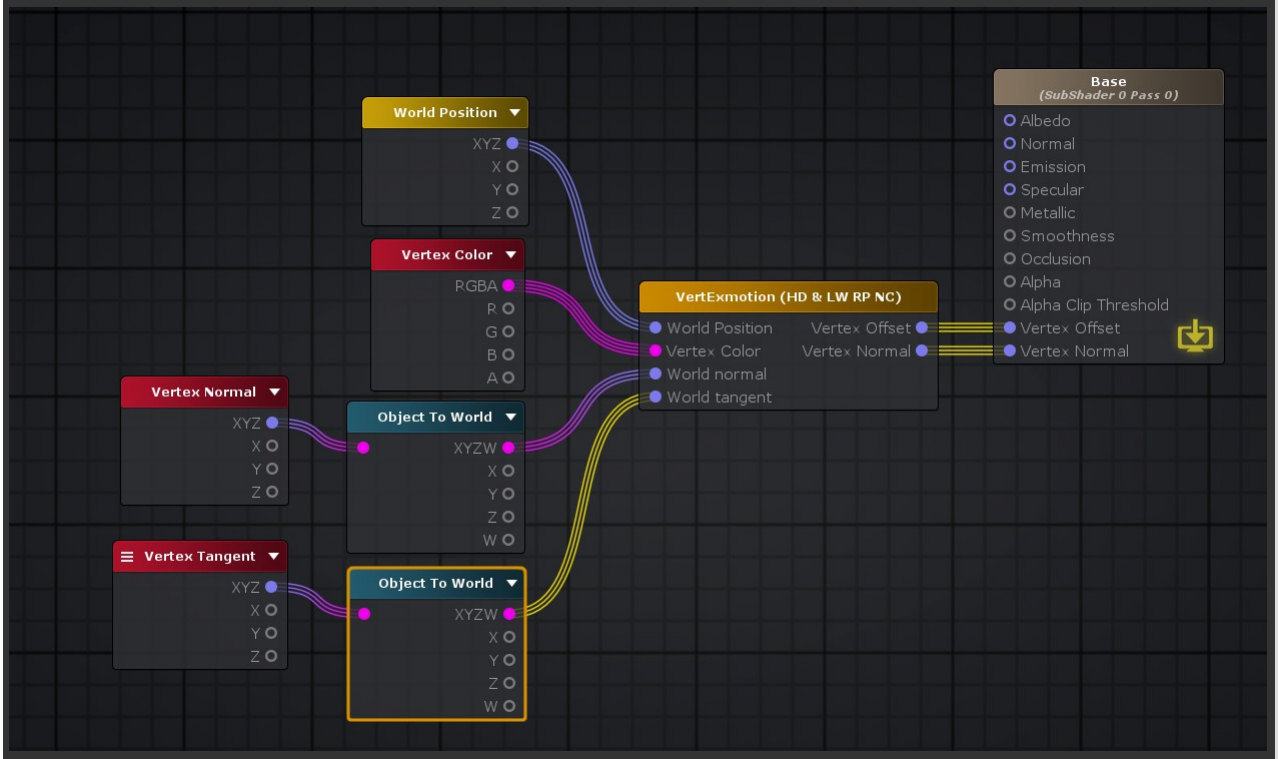
- The new node category 'VertExmotion' is available.
- Drag & drop the VertExmotion node.
- Connect it to the **'Local Vertex Offset' input**.



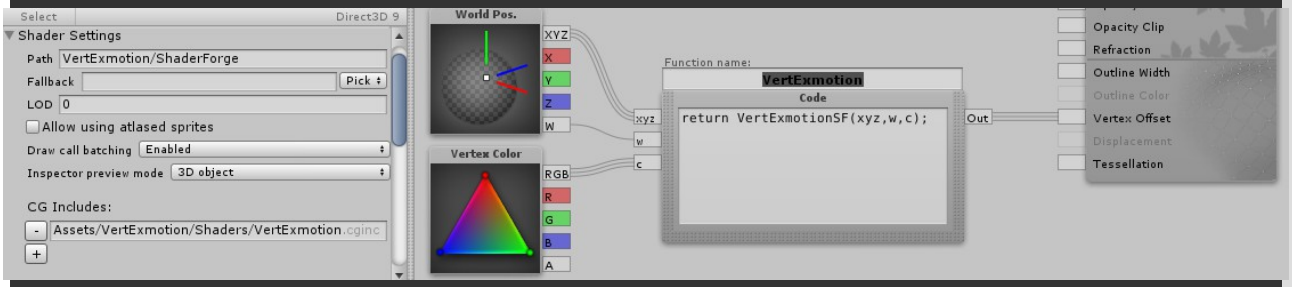
If you're using **tessellation** or other nodes that modify the 'local Vertex Offset', you can use the advanced version.

Official ASE documentation :
http://wiki.amplify.pt/index.php?title=Unity_Products:Amplify_Shader_Editor/Manual#VertExmotion

The normal correction node for LWRP and HDRP require the world normal and the world tangent.



How to include VertExmotion in Shader forge



- Open the shader in shaderForge editor.
- in shader Settings, set the path to 'VertExmotion/ShaderName' (ex : 'VertExmotion/ShaderForge/test1')
- Add a new CG include line : 'Assets/VertExmotion/Shaders/VertExmotion.cginc'
- Add a code node 'VertExmotion'
- copy the code :

```
return VertExmotionSF( xyz, w, c );
```

- set output to 'Float3'
- add an input 'Float3' : **xyz**
- add an input 'Float' : **w**
- add an input 'Float3' : **c**

- add a World pos node
- plug the XYZ output to **xyz**
- plug the W putput to **w**

- add a Vertex Color node
- plug the RGB output to **c**
- plug the VertExmotion node to 'Vertex Offset'

How to include VertExmotion to Alloy

Alloy 1.3.6 includes an official support to VertExmotion.
- Open the file 'Alloy/Shaders/Config.cginc'

- Uncomment :

```
#define A_USE_VERTEX_MOTION
#include "Assets/VertExmotion/Shaders/VertExmotion.cginc"
```

- Reimport the 'Alloy/Shaders' folder

How to convert a complex shader

Some shader are more complex than others, the method to convert them is always the same :
- Create a copy of the shader file
- Add the path 'VertExmotion/' to the name
- Find the vertex program or create it if not found.

The vertex program can be found in the surface shader by looking at the line
*#pragma surface ... vert:**VertExFunctionName** ...*

If the shader is a vertex/fragment program, look at the line
*#pragma vertex **VertExFunctionName***

In the complex shaders, the vertex function could be included in a .cginc file,
in this case, you'll have to :
- duplicate the cginc file (ex : shaderCore.cginc -> shadercore-VM.cginc)
- add the VertExmotion.cginc at the top of the file
- include the VertExmotion function in all the vertex program
- rename the include file in the shader ('#include "shaderCore-VM'.cginc")

- Add the line '#include "Assets/VertExmotion/Shaders/VertExmotion.cginc" ' at the top of the vertex function or at the top of the cginc file.

- Insert the VertExmotion function vertex program.

Do this for all the vertex program of the shader.

Tutorials for complex shaders

Preintegrated Skin Shader

- Find the vertex program used in the shader,
look at this line :

```
#pragma vertex pss_vert
...
#include "./PreIntegratedSkinShaderCore.cginc"
```

- The Vertex program is located in 'PreIntegratesSkinShaderCore.cginc' (pss_vert)
Duplicate the file to 'PreIntegratesSkinShaderCore-VM.cginc', and add the vertExmotion function

```
PSS_V2F pss_vert(PSS_VIN v) {
PSS_V2F o;
VertExmotion(v);
UNITY_INITIALIZE_OUTPUT(PSS_V2F,o);
```

-Add the include file at the top of the cginc file.

```
#include "Assets/VertExmotion/Shaders/VertExmotion.cginc"
```

Note : if you want to move VertExmotion folder location, you can copy 'VertExmotion.cginc' in the shader folder and change the line to : '#include "VertExmotion.cginc"'

- Duplicate the shader file to 'PreIntegratedSkinShaderStandard-VM.shader and add "VertExmotion/" at the begining of the name

- Change the input structure in 'PreIntegratedSkinShaderStandard.shader',
VertExmotion function require the 'appdata_full' structure

```
//#define PSS_VIN appdata_tan
#define PSS_VIN appdata_full
```

- Change all the '#include "./PreIntegratedSkinShaderCore.cginc"'
--> '#include "./PreIntegratedSkinShaderCore-VM.cginc"'

- Modify the shadow pass at the end of the shader file.
Use a VertexMotion pass to enable shadow deformation

```
// Shadow pass.
//SubShader { UsePass "VertexLit/SHADOWCASTER" }
SubShader{ UsePass "VertExmotion/Standard/SHADOWCASTER" }
```

UBER Shader

- Rename "UBER_StandardCore.cginc" to "UBER_StandardCore-original.cginc"
- Duplicate "UBER_StandardCore-original.cginc" to "UBER_StandardCore-VM.cginc"
- Create a new file "UBER_StandardCore.cginc" and copy :

```
#define VERTEXMOTION

#ifdef VERTEXMOTION
    #include "assets/VertExmotion/Shaders/VertExmotion.cginc"
    #define VERTEXMOTION_FUNC(v) v.vertex = VertExmotion(v.vertex,v.color);
    #include "UBER_StandardCore-VM.cginc"
#else
    #include "UBER_StandardCore-original.cginc"
#endif
```

- Edit "UBER_StandardCore-VM.cginc"
- Find the functions '**vertForwardBase**', '**vertForwardAdd**', '**vertDeferred**' and '**vert_meta**'
- Add 'VERTEXMOTION_FUNC(v)' at the beginning of each function

```
VertexOutputForwardBase vertForwardBase (VertexInput v)
{
    VERTEXMOTION_FUNC(v)
```

- Rename "UBER_StandardShadow_Tessellation.cginc"
  to "UBER_StandardShadow_Tessellation-original.cginc"
- Duplicate "UBER_StandardShadow_Tessellation-original.cginc"
  to "UBER_StandardShadow_Tessellation-VM.cginc"
- Create a new file "UBER_StandardShadow_Tessellation.cginc" and copy :

```
#define VERTEXMOTION

#ifdef VERTEXMOTION
    #include "assets/VertExmotion/Shaders/VertExmotion.cginc"
    #define VERTEXMOTION_FUNC(v) v.vertex = VertExmotion(v.vertex,v.color);
    #include "UBER_StandardShadow_Tessellation-VM.cginc"
#else
    #include "UBER_StandardShadow_Tessellation-original.cginc"
#endif
```

- Edit "UBER_StandardShadow_Tessellation-VM.cginc"
- Find the functions '**vertShadowCaster**' and '**vertPierceable**'
- Add 'VERTEXMOTION_FUNC(v)' at the beginning of each function

```
void vertShadowCaster (VertexInput v,
            #ifdef UNITY_STANDARD_USE_SHADOW_OUTPUT_STRUCT
            out VertexOutputShadowCaster o,
            #endif
            out float4 opos : SV_POSITION)
{
            VERTEXMOTION_FUNC(v)
```

*- Change the type '**appdata_base**' to **appdata_full**' in the* **'vertPierceable'** parameters

```
v2fPierceable vertPierceable(
            appdata_full v
            ) {
            VERTEXMOTION_FUNC(v);
```

- Reimport the 'shaders' folder to apply the modifications

Poiyomi Toon Shader

- Edit the file "_PoiyomiToonShader/Shaders/Includes/PoiVert.cginc"
- Add the line : #include "**Assets/VertExmotion/Shaders/VertExmotion.cginc**"
- Change the input structure from '**appdata**' to '**appdata_full**' (required for VertExmotion)
- Add the VertExmotion function : **VertExmotion(v);**
- Reimport the folder "_PoiyomiToonShader/Shaders"

```
#ifndef POIVERT
    #define POIVERT

    #include "Assets/VertExmotion/Shaders/VertExmotion.cginc"

    v2f vert(appdata_full v)
    {
        VertExmotion(v);

        v2f o;
        UNITY_SETUP_INSTANCE_ID(v);
        UNITY_INITIALIZE_VERTEX_OUTPUT_STEREO(o);
        UNITY_TRANSFER_INSTANCE_ID(v, o);
```

How to use the VertExmotion API

You can access to all the VertExmotion settings in a custom script. You'll be able to create custom FX for your specific needs.

For using VertExmotion in a custom script, you have to add the namespace : 'Kalagaan'.

Here a sample :

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Kalagaan;

public class VertExmotionDocSample : MonoBehaviour {

    public VertExmotion m_vtm;

    void Start()  {

        m_vtm = GetComponent<VertExmotion>();

        //Add a new sensor
        VertExmotionSensorBase newSensor1 = m_vtm.CreateSensor("My sensor 1");

        //add the sensor to the VertExmotion object
        //a sensor can be added to different VertExmotion object,
        //it's usefull for sharing deformations.
        m_vtm.AddSensor(newSensor1);


        //Create and add another sensor with a link FX
        //All vertices in the range of the capsule made by the 2 sensors will be deformed
        VertExmotionSensorBase newSensor2 = m_vtm.CreateSensor("My sensor 2 linked");
        m_vtm.AddSensor(newSensor2, true);

        //You can also call this function on existing sensors
        m_vtm.UnLink(newSensor2);
        m_vtm.Link(newSensor2);


        //Remove a sensor
        //The VertExmotion object won't be affected by the deformation
        m_vtm.RemoveSensor(newSensor2);


        //-----------------------------------
        //usefull functions

        //Reset all the deformations
        m_vtm.ResetMotion();

        //This function can be used for a teleportation behavior
        //the delta postion won't be applied for this frame in the procedural aniamtion
        m_vtm.IgnoreFrame();


  }
```

```
        void Update ()  {

        if (m_vtm == null)
            return;

        //'Sensors' is the list of the VertExmotion sensors
        for (int i = 0; i < m_vtm.Sensors.Count; ++i)
        {
                //you can change the radius
                m_vtm.Sensors[i].m_envelopRadius = 2f;

                //or any other parameters
                VertExmotionSensor.Parameter sensorParam = m_vtm.Sensors[i].m_params;

                sensorParam.inflate = 1f;
                sensorParam.power = 2f;
                sensorParam.scale = new Vector3(1f, 2f, 1f);

                // animation
                sensorParam.bouncing = 1f;
                sensorParam.damping = 1f;
                //...

                //Translation
                sensorParam.translation.amplitudeMultiplier = 1f;
                sensorParam.translation.localOffset = Vector3.up; //local transformation
                sensorParam.translation.worldOffset = Vector3.right; // world transformation
                //...

                //Rotation
                sensorParam.rotation.angle = Mathf.Sin(Time.time) * 360f;
                sensorParam.rotation.axis = Vector3.forward;

        }

  }

}
```

Have a look to the demos included in the package, some of them uses custom scripts.

The full API is available here : www.kalagaan.com/VertExmotion/API

Third party compatible assets

Global Snow (by Kronnect)
https://assetstore.unity.com/packages/slug/79795

Alloy (by RUST LTD)
https://assetstore.unity.com/packages/vfx/shaders/11978

Amplify Shader Editor
https://assetstore.unity.com/packages/tools/visual-scripting/68570

Toony Color pro 2 (by Jean MORENO)
https://assetstore.unity.com/packages/vfx/shaders/8105

Support

More Informations About VertExmotion

Unity forum : http://forum.unity3d.com/threads/vertexmotion-released.277294/

Youtube : https://www.youtube.com/channel/UCiaTcE4YXU0TZK0Xi-VM8vQ

Email for support : contact@kalagaan.com