VertExmotion

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.
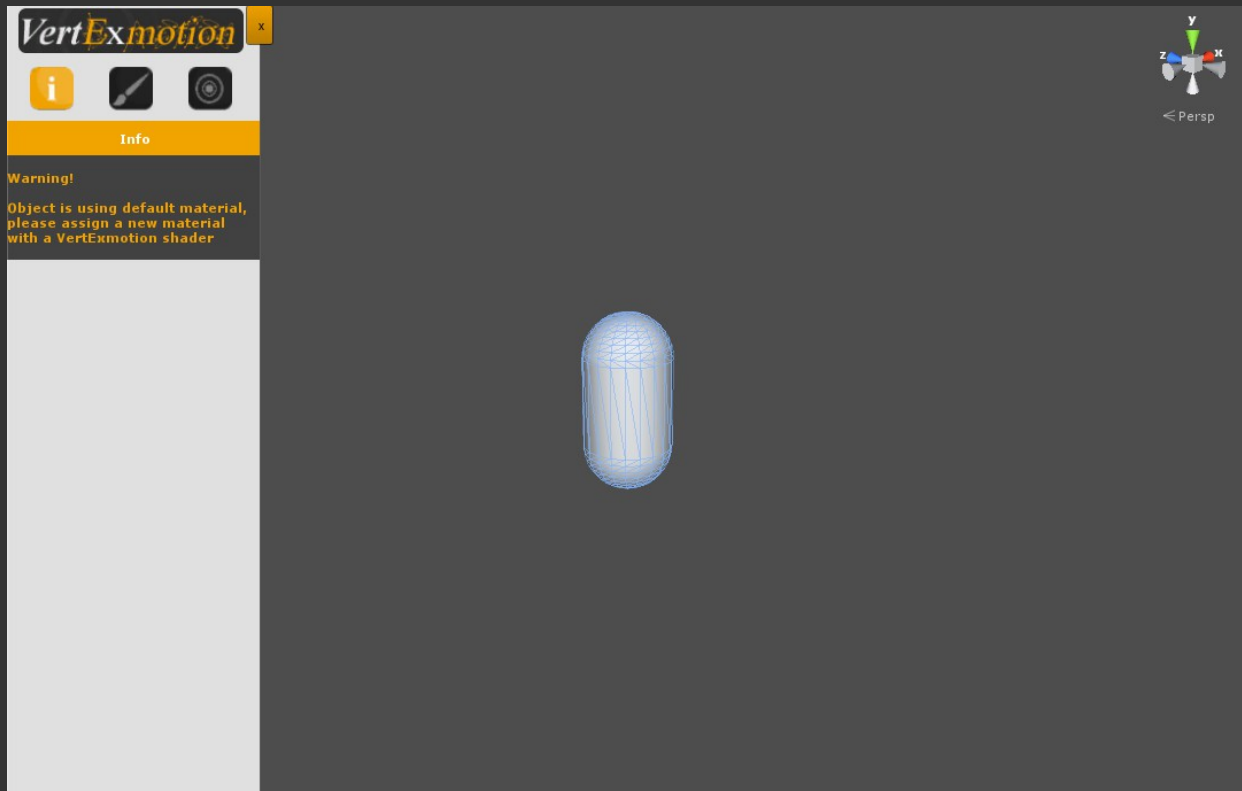Easy to include in your custom shaders.
Works with static mesh or skinned mesh.

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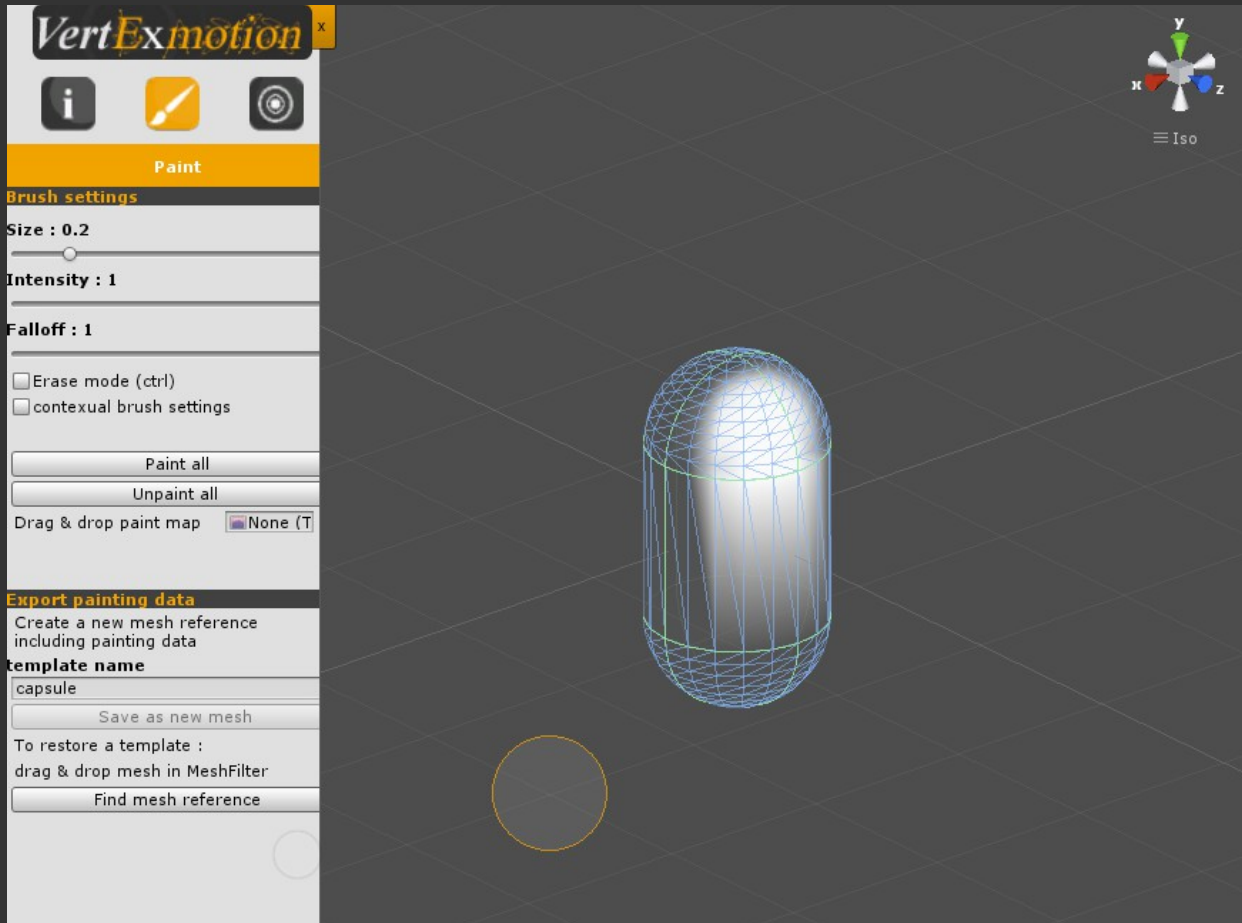
- Press ctrl to switch to erase mode.
  White vertices will be ready for motion.
  Black vertices will be static like a standard mesh.
  Intensity will affect motion.

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.
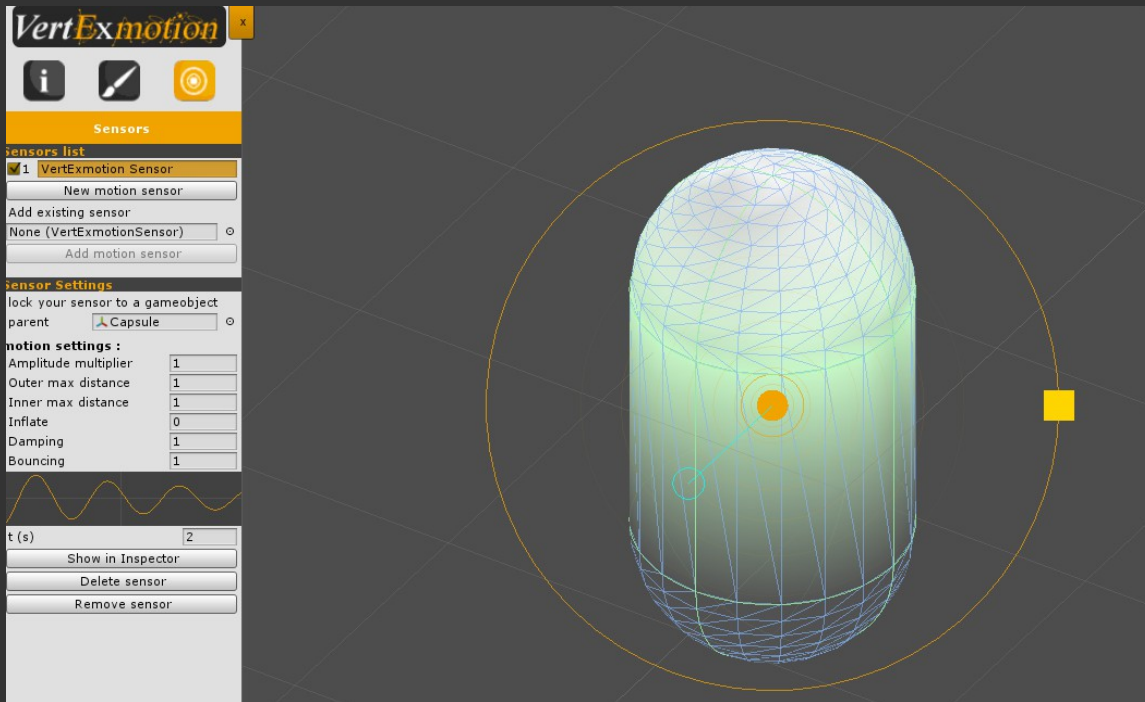
- Drag the sensor on the top of the mesh.
  Green vertices are in sensor range.
- Try to change it by draging the yellow square handler.
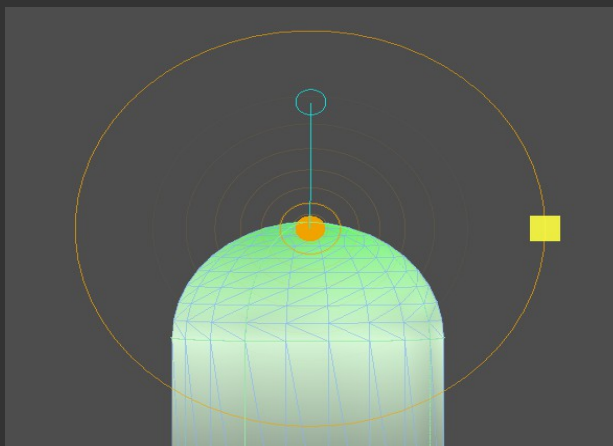
  Blue line is the sensor orientation
- Keep it out of the mesh by draging blue circle handler.
- Hit the play button.
- Move your mesh in sceneview.
  That's it! You have made your first jelly capsule!

- Now, have a look to these sensor settings

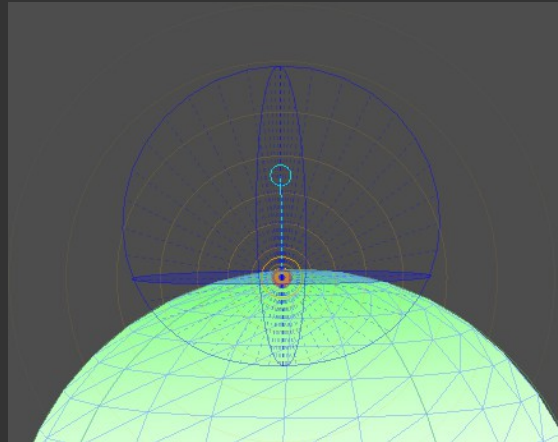The blue circles define the limits of the vertices displacement for the current sensor. The standard mode is based on the Z axis of the sensor transform.

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.
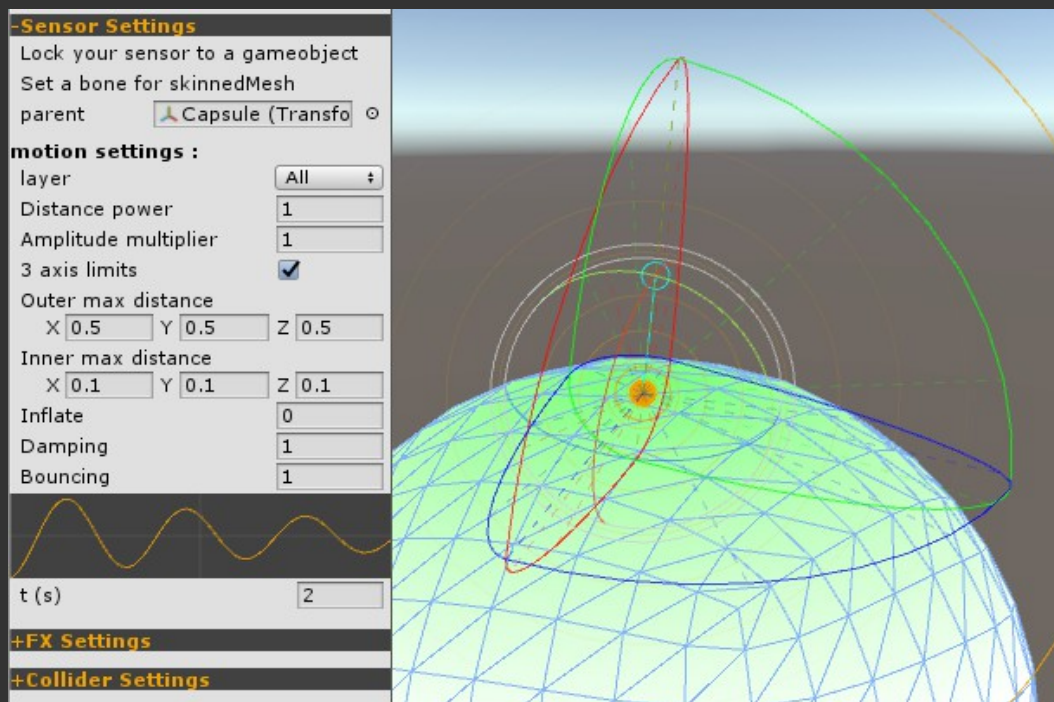
- Try to change settings and move your object.

  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 ⊃ .
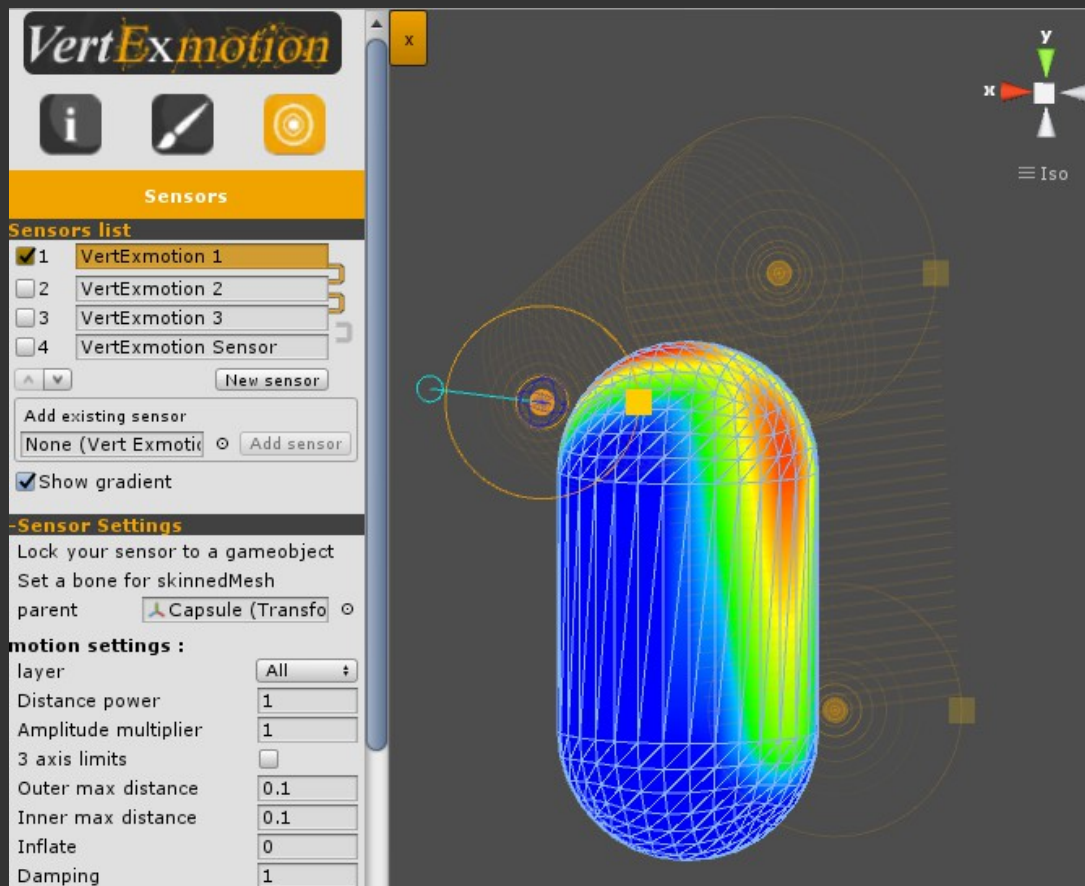
When 2 sensors are linked, all the vertices included in the range of the capsule (dual-sphere) will receive the motion data. The deformation informations are merged according to distance of the two sensors.

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

How to setup the collision system ?

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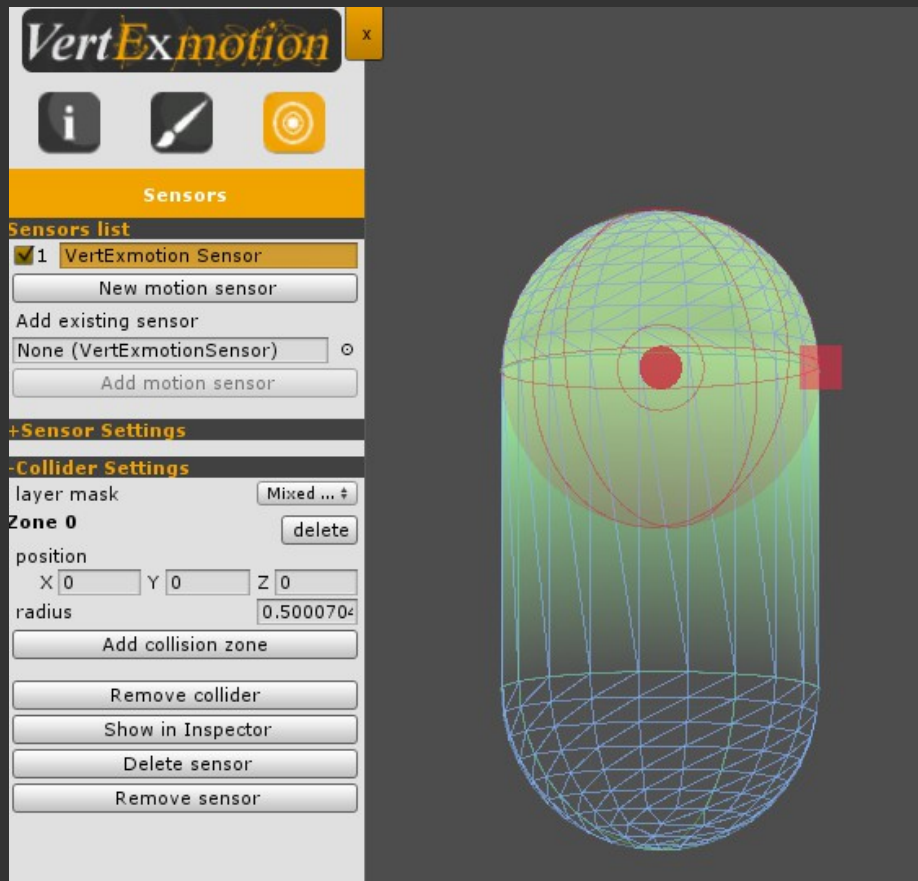
- Click on 'collider settings'
- Press 'Add Collider'
- Set the physics layer mask to collide with.
- Add a new collider zone.
- Change the position and the radius.
- Add other collision zones to suit the mesh surface.

If your mesh has already a physic collider (SphereCollider, BoxCollider...), collision zones must be larger than collider area.
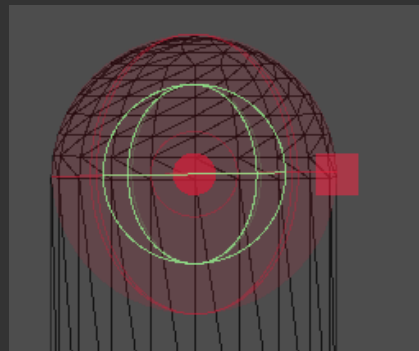
How to setup the sensor's layers

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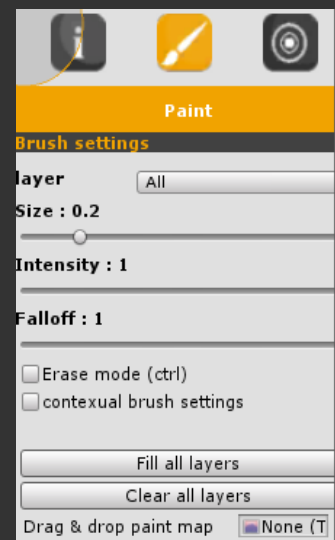
In the following case, the paint informations are not available on the 3rd layers, the sensor should be assign to '1' or 'All'.
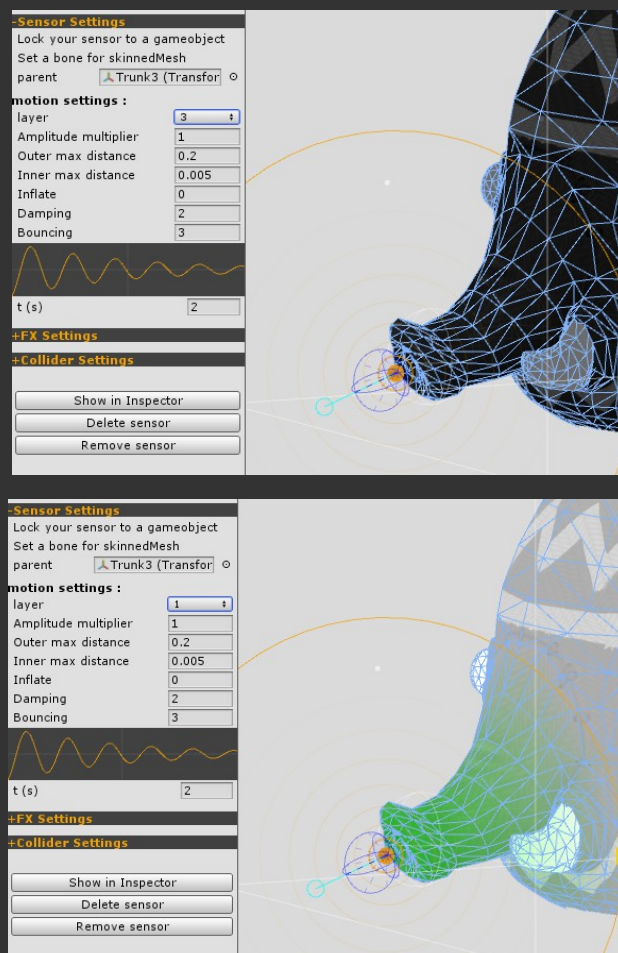
How to import paint data from a map

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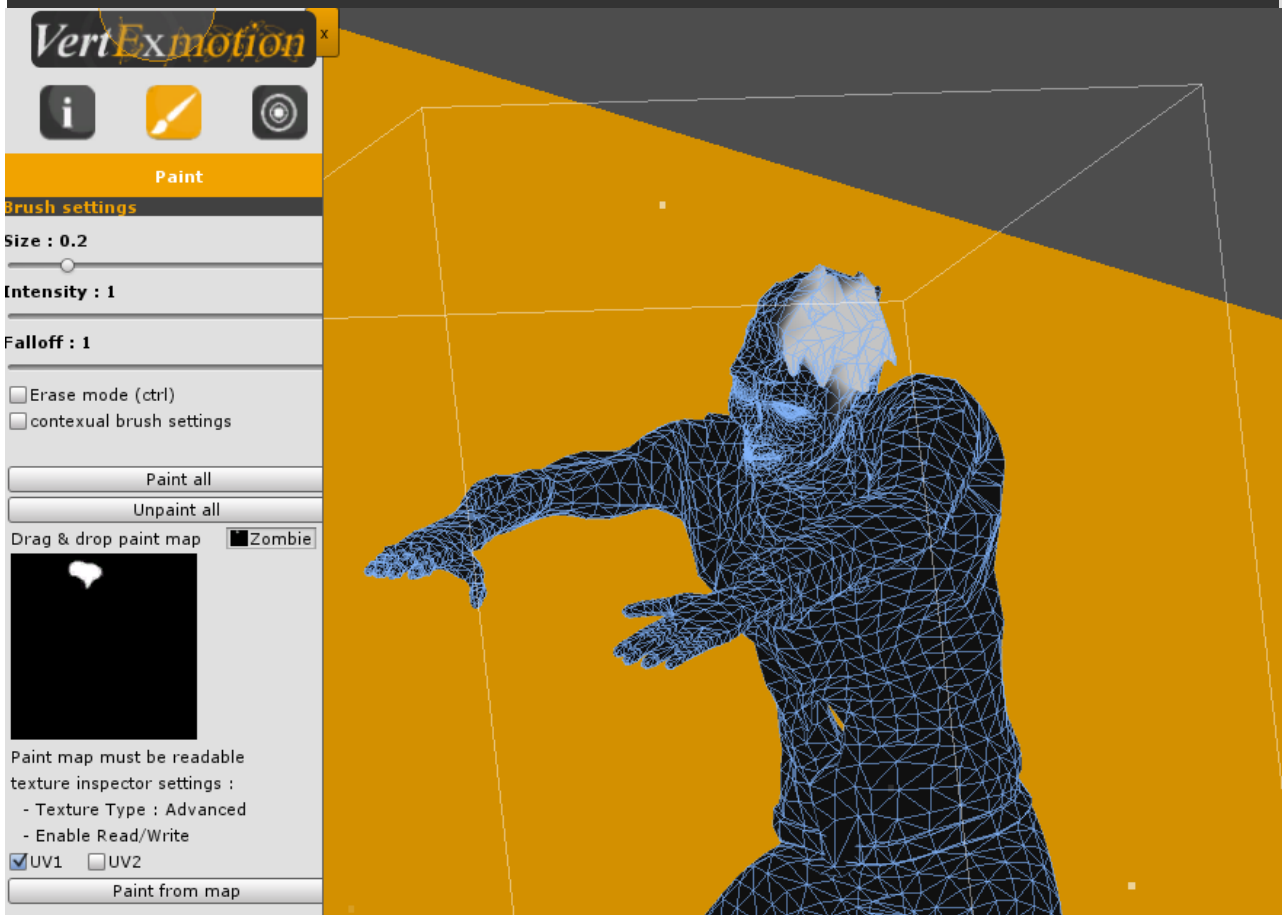
Here a tutorial :

- *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.
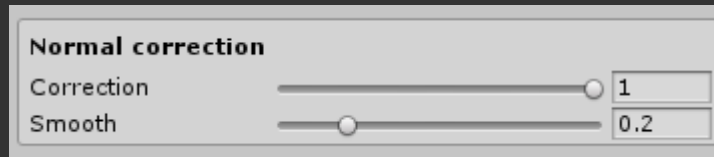
How to share sensors between mesh ?

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**'
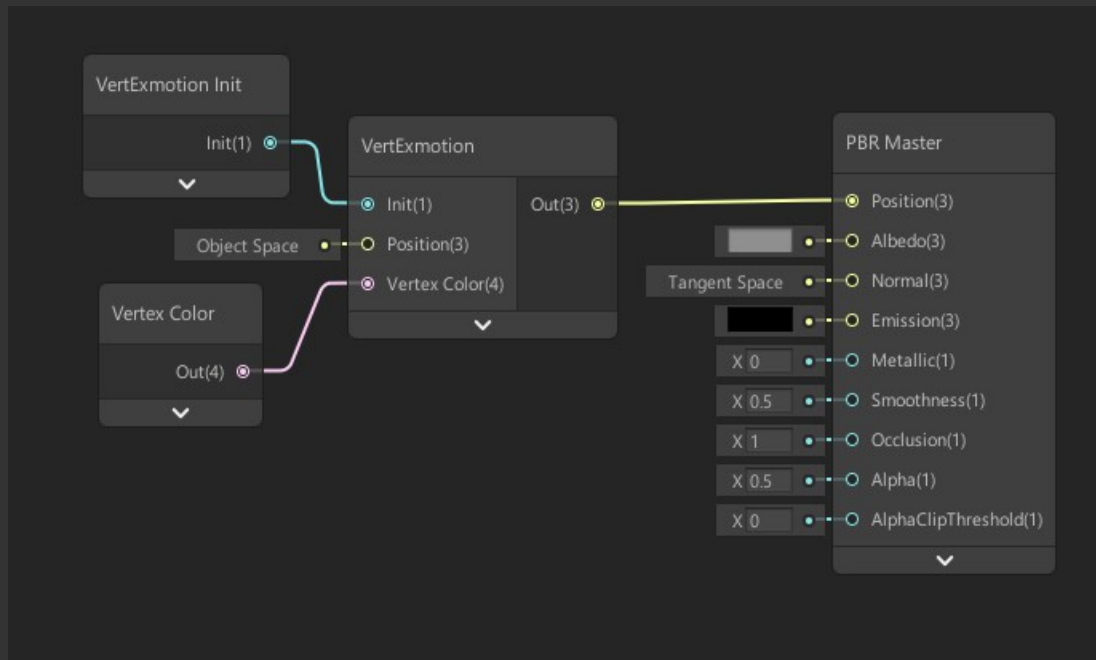
ShaderGraph 4.X :
- Drag & drop the VertExmotion Subraph (VertExmotion/Editor/ShaderGraph/)
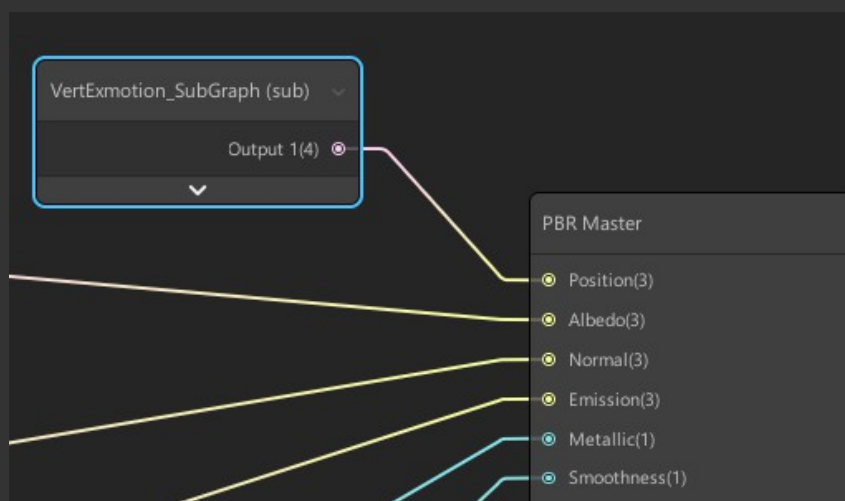
How to include VertExmotion in Amplify Shader Editor

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 Includes' section**.
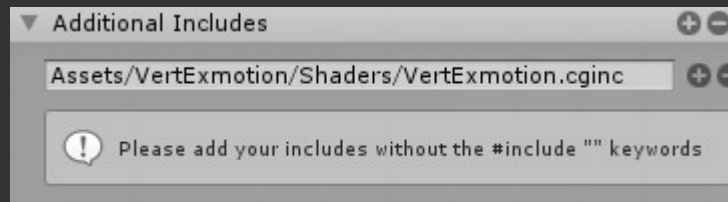
- The new node category 'VertExmotion' is available.
- Drag & drop the VertExmotion node.
- Connect it to the **'Local Vertex Offset' input**.
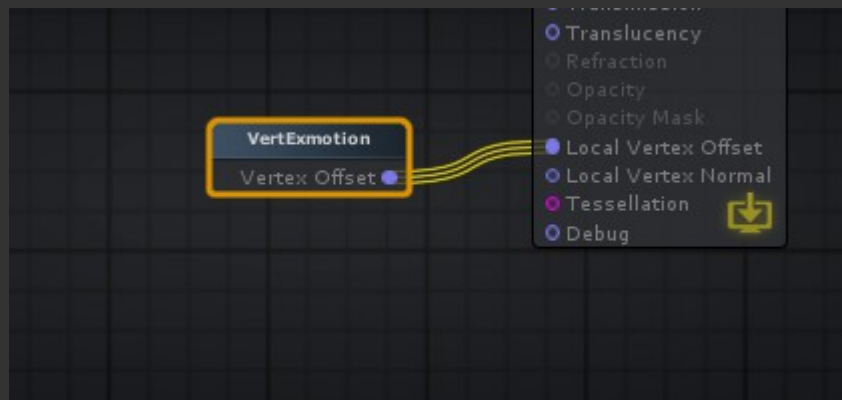
If you're using **tessellation** or other nodes that modify the 'local Vertex Offset', you can use the advanced version.
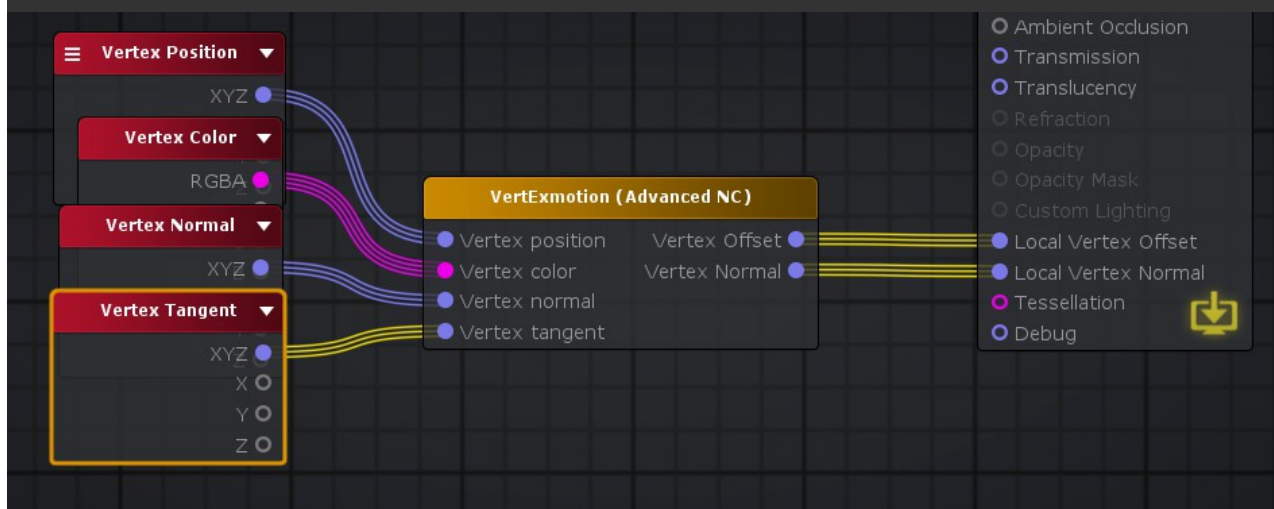
How to include VertExmotion in Shader forge
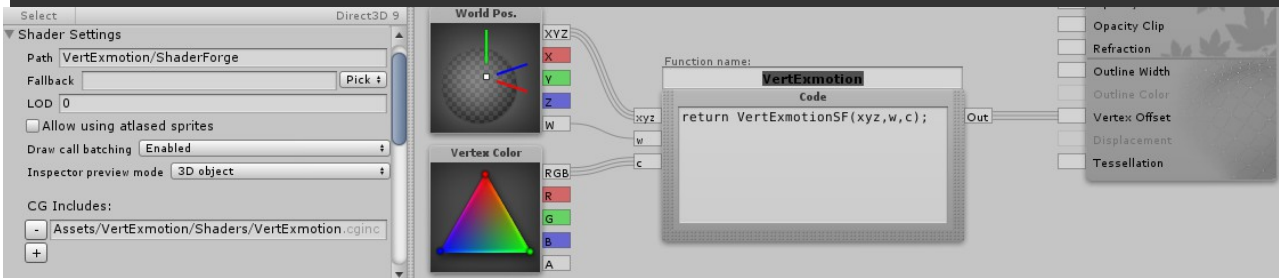
- Open the shader in shaderForge editor.
- in shader Settings, set the path to 'VertExmotion/ShaderName' (ex : 'VertExmotion/ShaderForge/test1')
- Add a new CG include line : 'Assets/VertExmotion/Shaders/VertExmotion.cginc'
- Add a code node 'VertExmotion'
- copy the code :

      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.

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 for enable shadow deformation

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

UBER Shader

Here a sample for the tesselation shader :

- Duplicate the file '_UBERShader_Specular_Tessellation'

- Rename the file to '_UBERShader_Specular_Tessellation-VM'

- Rename the shader name to 'VertExmotion/UBER - Specular Setup/Tessellation/ Core'

- search "#pragma vertex" in the file

- Find the first vertex program name (tessvert_surf)

- Find it in the include file "../Includes/UBER_StandardCore.cginc"

- Duplicate "../Includes/UBER_StandardCore.cginc" to "../Includes/UBER_StandardCore-VM.cginc"

- Open it and add '#include "VertExmotion.cginc"' at the top of the file

- search the vertex program (tessvert_surf)

- Add the line :
      o.vertex = VertExmotion(v.vertex, v.color);

- Go back to the shader file and replace the cginc include reference :
  #include "../Includes/UBER_StandardCore.cginc" -> #include "../Includes/UBER_StandardCore-VM.cginc"

- Do this for all the "#pragma vertex ..." and the .cginc

When all the vertex program have been patched in the cginc file, you can easily convert all the shaders by changing the include reference.

Support

*More informations about **VertExmotion** on the forum :*
http://forum.unity3d.com/threads/vertexmotion-released.277294/

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

Email for support : contact@kalagaan.com