TRESSFX 5.0/ UNREAL ENGINE INTEGRATION DEVELOPER GUIDE

For AMD TressFX/Epic Games Unreal Engine Integration

THIS GUIDE:

- Describes new features of the TressFX integration into Unreal Engine 4.27.2 (branch available to developers with an Epic Games EULA/GitHub account).
- Documents shader parameters, architecture, new features, Maya Exporter, and more.
- Includes basic tutorials for creating hair, using TressFX Exporter, and importing into Unreal.

Contents

TressFX 5.0 – Overview

This version of TressFX has been developed on Unreal Engine 4.26 and UE4.27, but we will support UE5 in the next revision.

In this version, we have improved the Maya exporter to handle more complicated animation assets, with support for up to 16 binding bones. We have also improved the asset editors and visualization tools.

Lastly, we have also improved light handling for TressFX, including cast/receive shadow implementation support for more rendering features in UE4 (TAA, SkyLight, Marschner Shading Model) which are more compatible with the Unreal Engine rendering pipeline.

TressFX – from 4.1 to 5.0

This section briefly covers the changes in TressFX from 4.1 to 5.0.

Maya Python Exporter

- Both strand and collision mesh vertices now support up to 16 binding bones. The previous limit was 4.
- Caution: Strand/Collision Mesh needs to be exported in the right initial position and keyframe.

Asset Editor & Visualization Tools

- LevelMap/Asset/Blueprint editor support.
- Created a new visualization toolchain to verify assets correctness (tangents, collision mesh, etc.).

Simulation

- Resolved issue with animation data lagging a frame behind UE4 skinned mesh animation data.
- Improved simulation editor and implemented SDF editor.
- Moved SDF BoundingBox computation to GPU-side.
- Improved SDF feature.

Rendering

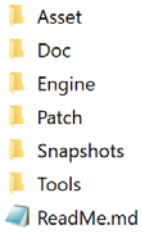
- TAA/SkyLight/StrandsUV support.
- Marschner Shading Model support.
- Cast/Receive Shadows support.
- Triangle based Skinning support.
- Improved resource management for editors.

TressFX 5.0 Unreal Engine Integration

This section details how to integrate TressFX 5.0 into Unreal Engine.

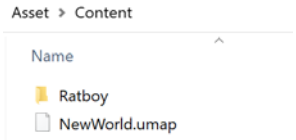
Folder Structure

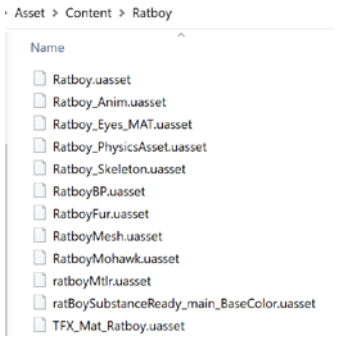
TressFX 5.0 package is broken down into several subfolders, as shown below.



Asset

Ratboy Assets have been imported into UE4. You can directly copy it into the **Content** folder of your UE project.



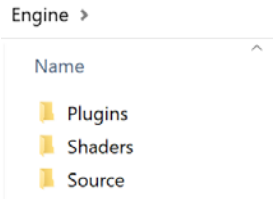


Doc

Folder containing this document: TressFX5.0 DeveloperGuide.pdf

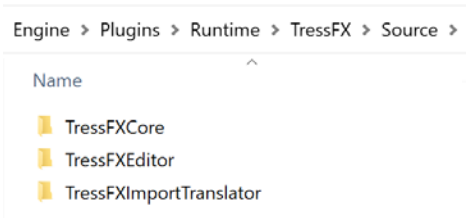
Engine

Folder containing all of the source code for this project divided into three subfolders.

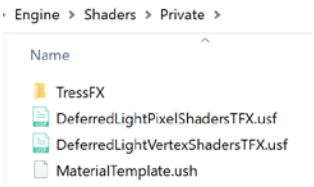


Plugins

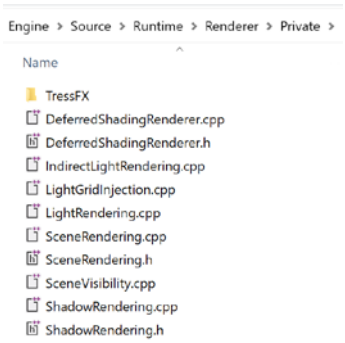
Folder containing the Editors/Importer/Simulation for UE4 TressFX 5.0.



Shaders
Folder containing the shaders for UE4 TressFX 5.0. There is only one modified Unreal Engine shader in this folder, MaterialTemplate.ush.



Source
Folder containing the engine C++ files for UE4 TressFX 5.0. Modified engine C++ files are marked with an `AMD TressFX BEGIN/AMD TressFX END` segment to mark where code has been changed.



Patch
Folder containing the git patch file of UE4.27.2.

Snapshots
Folder containing some screenshots of UE4 TressFX 5.0.

Tools
Folder containing the Maya2018 exporter which is used to export *.tfx, *.tfxbone, and *.tfxmesh files.

Integration steps (UE4.27.2)

In order to integrate TressFX 5.0 into UE4 follow these simple steps:

1. If you haven't changed the UE4 engine files or shaders in your project, you can directly integrate the patch highlighted in the previous section: **Patch/UE4.27.2_TressFX5.0.patch**:

   ```
   git am < UE4.27.2_TressFX5.0.patch
   ```

   Alternatively, copy and replace the **Engine** folder to your local **Engine** folder.
   - Engine/Plugins
   - Engine/Shaders
   - Engine/Source
2. If you have already made changes to the UE4 engine files that TressFX is going to replace, copy the `AMD TressFX BEGIN/END` segments into the corresponding C++ files and shaders.
3. Run GenerateProjectFiles.bat
4. Recompile UE4.sln
5. Create a new Project with StarterContent
6. Enable the plugin TressFX 5.0 in PluginManager
7. Copy the **Asset/Content** folder to the new project's **Content** folder
8. Startup the project and open the **NewWorld** LevelMap. Make sure to enable *CastDeepShadow* for all Lights (except SkyLight) to ensure everything renders correctly

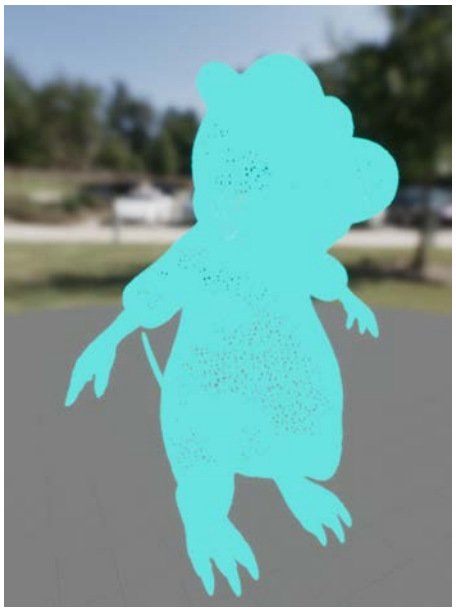
UE4 TressFX 5.0 Architecture

UE4 TressFX 5.0 is a bone-based skinning system for hair/fur simulation and rendering which uses SDF for collision. The general process flow is:

1. Model hair as splines (NURBS curves).
2. Export the splines using the rigged (skinned) mesh as a reference into the required data files.
3. Import those files and turn them into data on the CPU side.
4. Simulate and render using its shaders. This includes getting the required information from the engine/rendering pipeline in order to properly react and simulate (bone information, lighting, etc.).

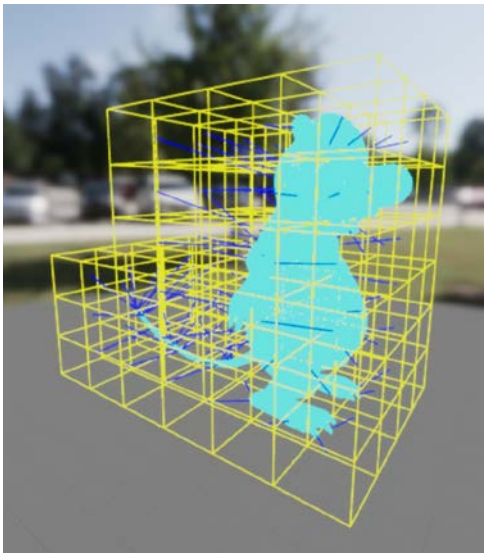
Collision Mesh

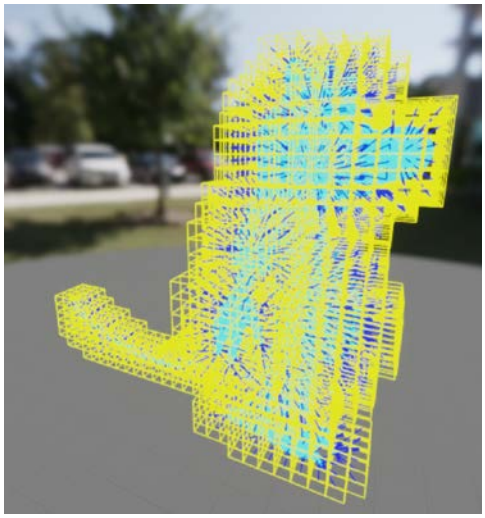
A collision mesh is a triangle mesh and input for the SDF. It does not need to be the same as the rendering mesh but is recommended to be a closed mesh. It is also recommended to be non-overlapping.



Signed Distance Field (SDF)

An SDF is a grid-based representation for a polygonal mesh.





In TressFX, a dense grid is used and the memory gets allocated up front. Because of this, pre-allocation of memory, the grid, and cell sizes should be chosen carefully so they are big enough to enclose all possible animations. However, this should not be too excessive, as this might cause potential memory waste. It would be wise therefore to break down mesh parts and use different grid and cell sizes. In the Ratboy demo, for example, the character has three body parts (main body and two hands).

The Signed Distance Field is best defined using closed meshes. Although a water-tight mesh is not required, it's best to minimize any holes in the mesh that are larger than a grid cell unless this is a portion that won't interact with the hair.

Velocity Shock Propagation (VSP)

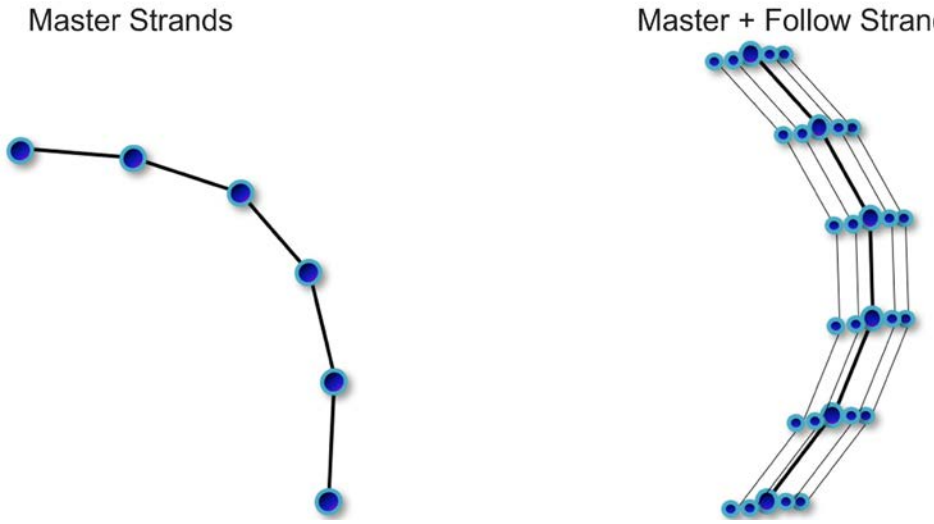
The main purpose of VSP is to handle fast-moving animations. When a character changes its speed or direction, it generates a high acceleration and consequently a big external force. Since TressFX hair simulation uses an iteration-based constraint solver, when a high acceleration gets applied, hair can easily lose its physically-correct shape and show unpleasant elongation.

VSP has a control value ranging from 0 to 1. If the value is 0, there will be no VSP. If VSP is 1, then full velocity of root vertex will be propagated to the rest of vertices in the strand. VSP can handle both linear and rotational velocities.

In addition to propagating velocity, there is a VSP acceleration threshold value which increases the VSP value to 1 when the pseudo-acceleration passes it. It is designed to be particularly effective when the character makes a sudden movement.

Guide and Follow Hair System

This system was introduced in TressFX 2.0 and is used the same way in TressFX 5.0. In terms of collision with the SDF, both guide and follow hair will get a response from the system. However, it is still possible to redesign such that only guide hair would be affected by the SDF, while the overhead is small enough to use SDF collision for all hair.



Unreal Engine Integration Architecture

The TressFX integration is primarily split into two modules, TressFX Plugin & TressFX Renderer. You can find these in **Plugins/Runtime/TressFX/Source/** and **Engine/Source/Runtime/Renderer/Private/TressFX/**, respectively.

TressFX Engine Hooks

TressFX is implemented as an add-on to Unreal Engine. There are a few locations in Unreal Engine's code base that hooks must be added in order to call into various parts of the TressFX implementation, as well as for general scene and visibility tracking.

We've strived to keep these changes minimal, and have attempted to bookend all code changes in the Unreal Engine code with:

```
// AMD TressFX BEGIN
// AMD TressFX END
```

Many files within Unreal Engine are modified to accommodate TressFX. The modified files are listed below alongside how they support TressFX.

These files add TressFX-related material support:

- ***Material.h/Material.cpp***
- ***MaterialInterface.h/MaterialInterface.cpp***
- ***MaterialRelevance.h***
- ***MaterialShared.h/MaterialShared.cpp***
- ***MeshBatch.h***

This file adds TressFX-related relevances:

- ***PrimitiveViewRelevance.h***

This file calls into our simulation and rendering stages Unreal Engine's main rendering loop:

- ***DeferredShadingRender.cpp***

These files allow TressFX to render with Unreal Engine's lights:

- ***LightRendering.cpp***
- ***IndirectLightRendering.cpp***
- ***LightGridInjection.cpp***
- ***LightComponent.cpp***

These files add arrays of primitive scene information for TressFX-related functionality:

- ***SceneRendering.h***
- ***SceneRendering.cpp***

This file sets up TressFX-related views:

- ***SceneVisiblity.cpp***

This file adds DeepShadow-related support for TressFX:

- ***SceneManagement.h***

This file allows TressFX to render part of Unreal Engine's shadow map rendering:

- ***ShadowRendering.h***

TressFX Plugin Module

As the name implies, this is a plugin module. It contains the Importer, Editors, and Simulation of TressFX.

TressFXImporter

TressFXImporter contains the code required to import TressFXAsset and the asset actions for them.

The TressFXFactory files contain the factories for the TressFXAsset as well as the TressFXMeshAsset. Both factories use import data classes (UTressFXImportOptions and UTressFXMeshImportOptions) which will contain the user adjustable data at runtime. Widgets are created based on these classes. Users should make sure they set the proper skeleton they plan on using at runtime. The factory will go over the Bone data stored in the TFX files and compare it to the skeleton you selected. If the bones have improper indexes, it will try to find the correct bone by name and change the indexes for the Assets to match the skeleton.

The TressFXAsset factory will first import the data into the FTressFXAsset class, then create a UTressFXAsset from that data. The TressFXMeshFactory will put all the data into a string which will then get parsed for the information.

TressFXEditors

TressFXAsset Editor contains 4 panels to edit the parameters of TressFX strands data which are Material, Strands, LOD, and Simulation.

TressFXMeshAsset Editor is used to edit the Collision Mesh and signed distance field (SDF) resolution.

TressFXManager

TressFXManager contains the registered TressFXGroupInstances and TressFXMeshGroupInstances which are used in Simulation passes.

TressFXComponent

TressFXComponent contains a TressFXAsset, generates and registers/unregisters a TressFXGroupInstance in TressFXManager, InitResources() and ReleaseResources() to create and release related resources.

TressFXGroupInstance

TressFXGroupInstance contains both guides and follow strands resources information, contains the variables which would be used in Simulation and Rendering passes.

TressFXSDFComponent

TressFXSDFComponent contains a TressFXMeshAsset, generates and registers/unregisters a TressFXMeshGroupInstance in TressFXManager, InitResources() and ReleaseResources() to create and release related resources, turn on/off SDF, set the local SDF Id.

TressFXMeshGroupInstance

TressFXMeshGroupInstance contains Collision Mesh rest resources and deformed resources information. The BoundingBox would be computed on GPU-side based on deformed Collision Mesh, and the BoundingBox then would be used to compute SDF.

TressFXResources

TressFXComponent is the primary interface for the hair and the editor. This class needs to be attached to the appropriate SkeletalMeshComponent that is associated with the hair asset you plan to use with this component. This is the class where you can set individual settings for different hair assets. TressFXResources contains all the resources of guides and follow strands, both rest and deformed resources, deformed guides resources would be interpolated to deformed strands resources for rendering. These resources would be create or init in TressFXComponent::InitResources() and delete or destroy in TressFXComponent::ReleaseResources().

TressFXSceneProxy

The scene proxy is responsible for combining the TressFXGroupInstance and TressFXVertexFactory and generating the MeshBatch.

TressFXAsset and TressFXMeshAsset

These are the two data classes which will be serialized and saved to disk. They contain the static data for the hair assets and the Collision Mesh assets. The data will be read from disk during the serialize function, then the related resources get initialized during the InitResources() function.

TressFX Renderer Module

The renderer is responsible for all rendering and GPU activity for TressFX. The proxies will get collected by the engine, then placed inside their respective arrays inside SceneRendering.h. The TressFX Renderer will then go through these lists and call the appropriate shaders to draw the hair.

TressFXPreBasePass

This generates DeepShadowMap and DeepOpacityMap for each TressFXComponent and each Light which has enabled CastDeepShadow. The results are used in ShadowRendering and TressFX Shading. We have implemented a DeepShadow approximation for AMD GPUs.

TressFXBasePass

This generates translucency results, using the simplified ShortCut algorithm from TressFX 4.1. Only the 1st layer will be shaded. Supports Unreal Engine's deferred shading pipeline, updates the velocity buffer to support TAA, and supports StrandsUV in TressFXVertexFactory.ush.

RenderLights

This generates TressFXShadowTexture in TressFXShadowMask pass which is called in ShadowProjectionOnOpaque pass. It generates Transmittance result in TressFXTransmittanceMask pass and applies the Marschner Shading Model in StandardDeferredLighting_TressFX pass.

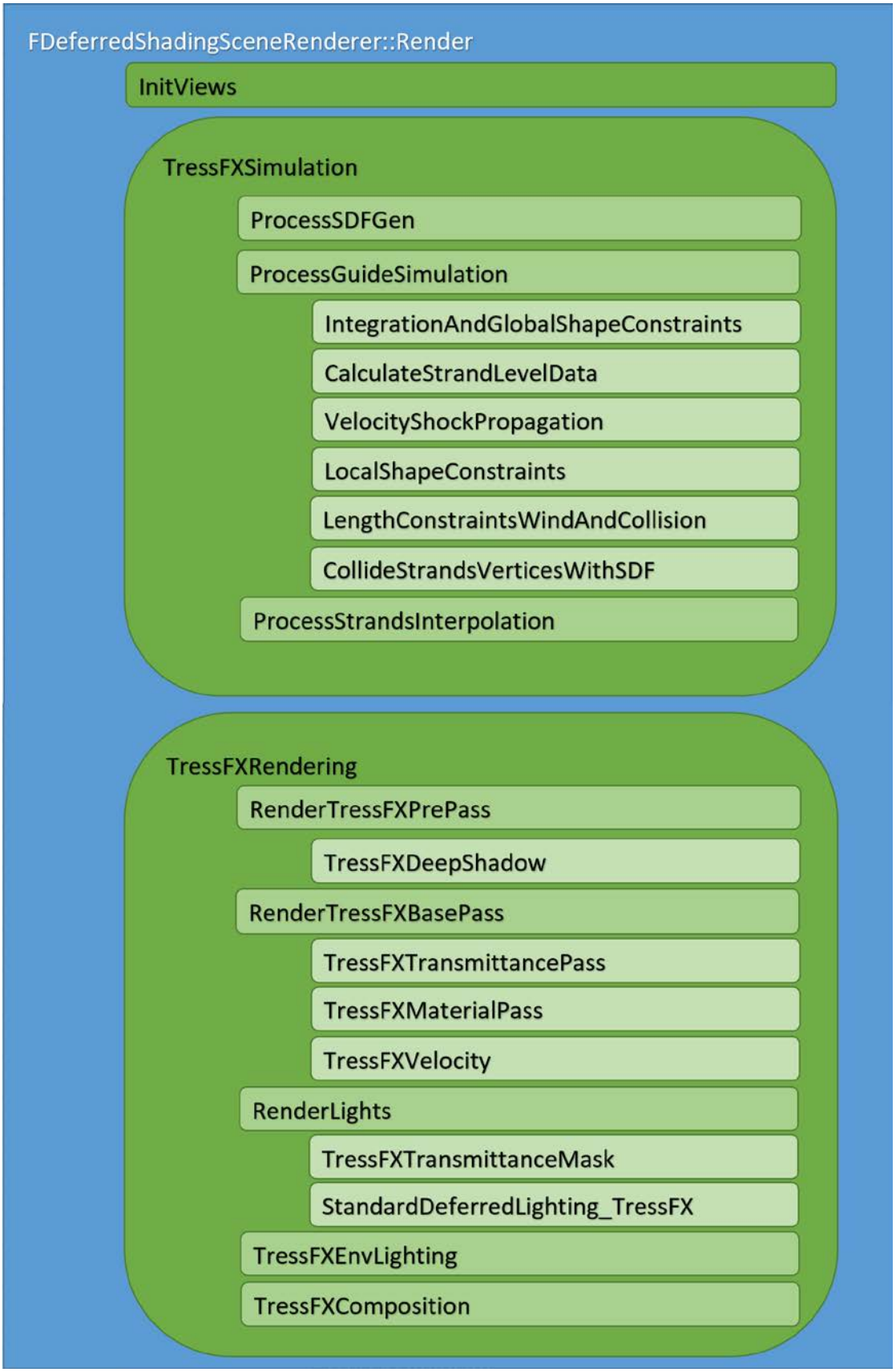
TressFXEnvLighting

TressFX supports Environment Sky Lighting.

TressFXComposition

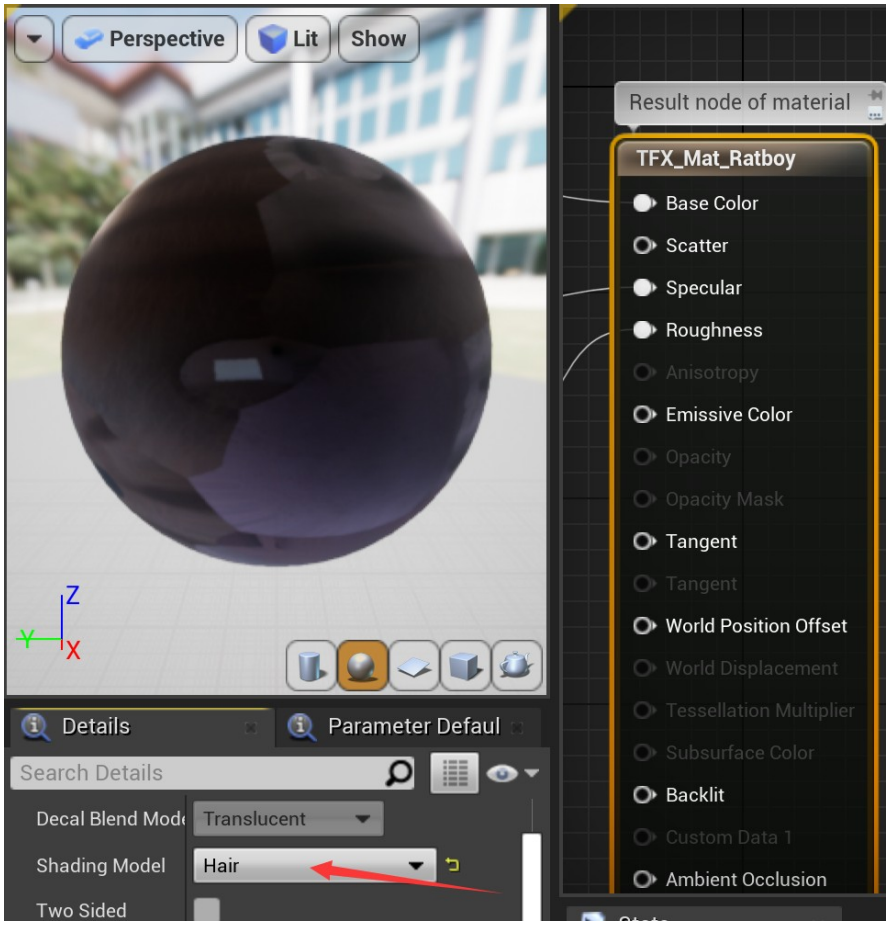
The results of lighting and shading will be composited into the SceneColorDeferred render target.

Main flow of TressFX Simulation and Rendering



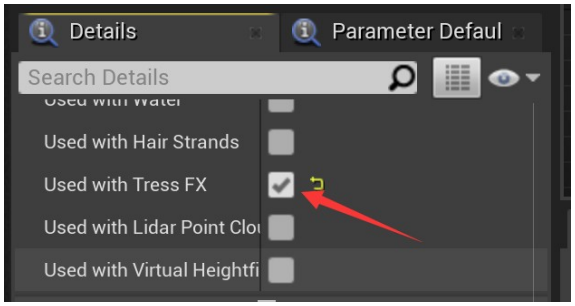
UE4 TressFX 5.0 and Material and Editors

TressFX Material



Shading Model

The shading model of the material. It should be set to *Hair* for TressFX.



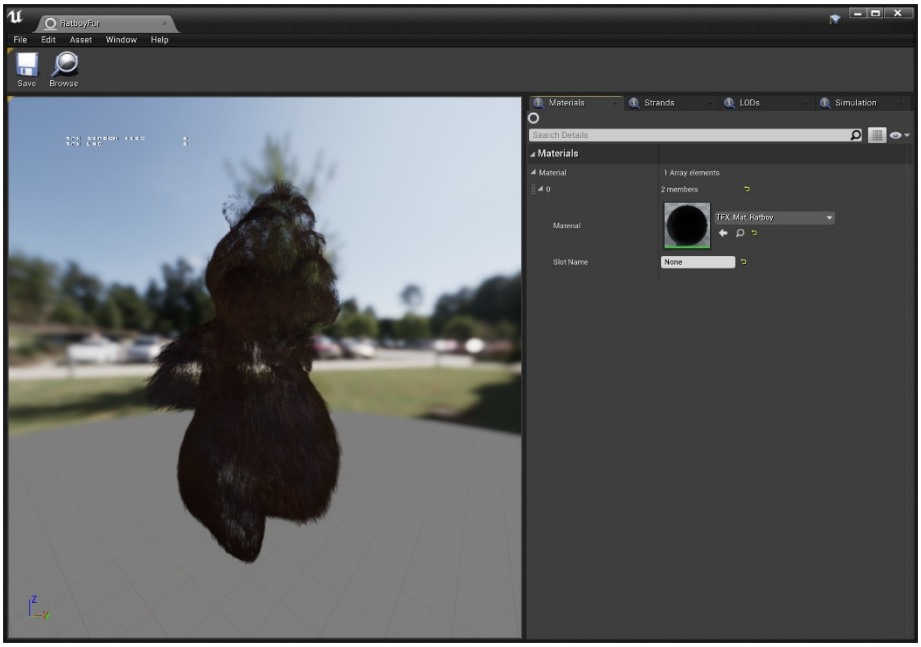
Used with TressFX

Enable it to support TressFX material.

For a tutorial that walks you through creating UE4 TressFX 5.0 material, see the tutorial in this document: Creating UE4 TressFX 5.0 Material.

TressFX AssetEditor

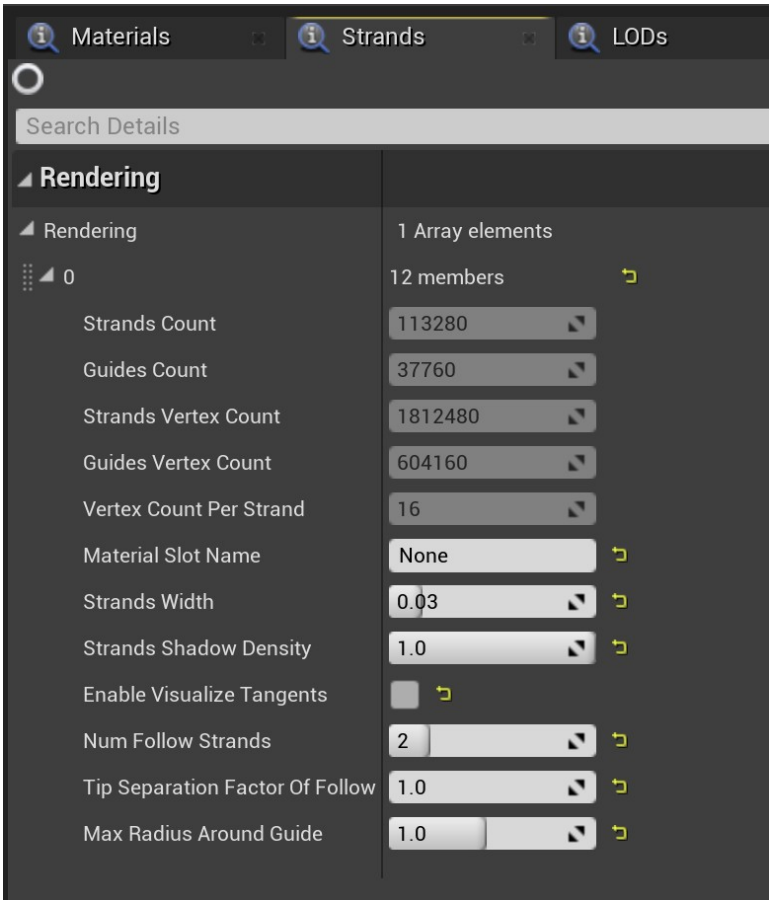
Material Panel



Material

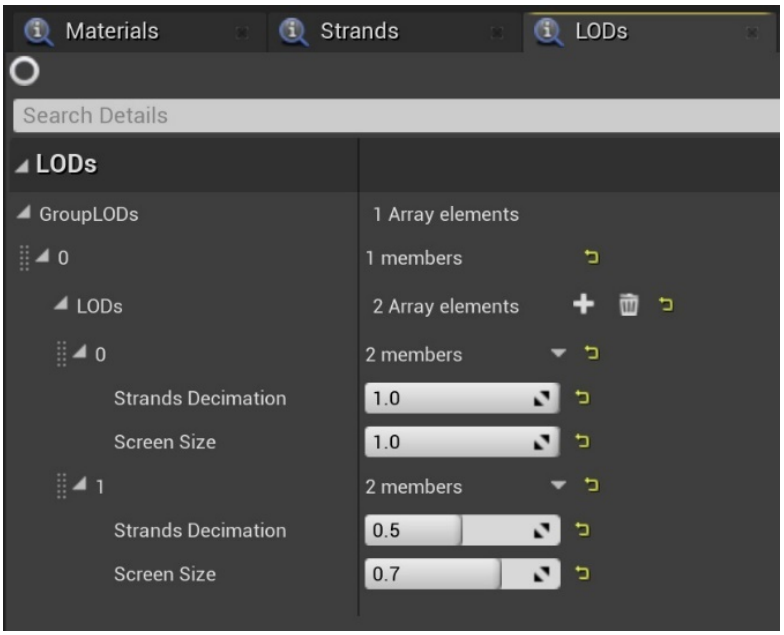
From the *Materials* tab, choose and set the **TFX_Mat_Ratboy** material.

Strands Panel



StrandsCount
The count of both guide and follow strands.
GuidesCount
The count of the strands which compute simulation.
StrandsVertexCount
Total vertex count of all strands.
GuidesVertexCount
Vertex count of all guide strands.
VertexCountPerStrand
Vertex count of one strand in this group.
MaterialSlotName
The material name for this group.
StrandsWidth
The width of strands in this group.
StrandsShadowDensity
Controls the shadow density.
EnableVisualizeTangents
Visualize the tangents of the strands.
NumFollowStrands
The number of follow strands for each guide strand.
TipSeparationFactorOfFollow
Adjusts the distribution of follow strands.
MaxRadiusAroundGuide
Controls the distribution radius of follow strands.

LODs Panel



StrandsDecimation
The strand simplification factor for the current LOD level.
ScreenSize
The screen size of the TressFXComponent bounding box.

Simulation Panel

EnableSimulation

Enable or disable simulation of this TressFX group.

Damping

Damping of gravity.

GlobalShapeStiffness

The stiffness of global shape constraint.

GlobalShapeEffectiveRange

The range of global shape constraint from root.

VSPCoefficient

The coefficient of velocity shock propagation works when acceleration exceeds the two thresholds (Min and Max).

VSPAccelThresholdMin

The Min threshold of VSP acceleration.

VSPAccelThresholdMax

The Max threshold of VSP acceleration.

LengthConstraintIterations

The iterations of length constraint.

WindDirection

The direction of wind.

WindMagnitude

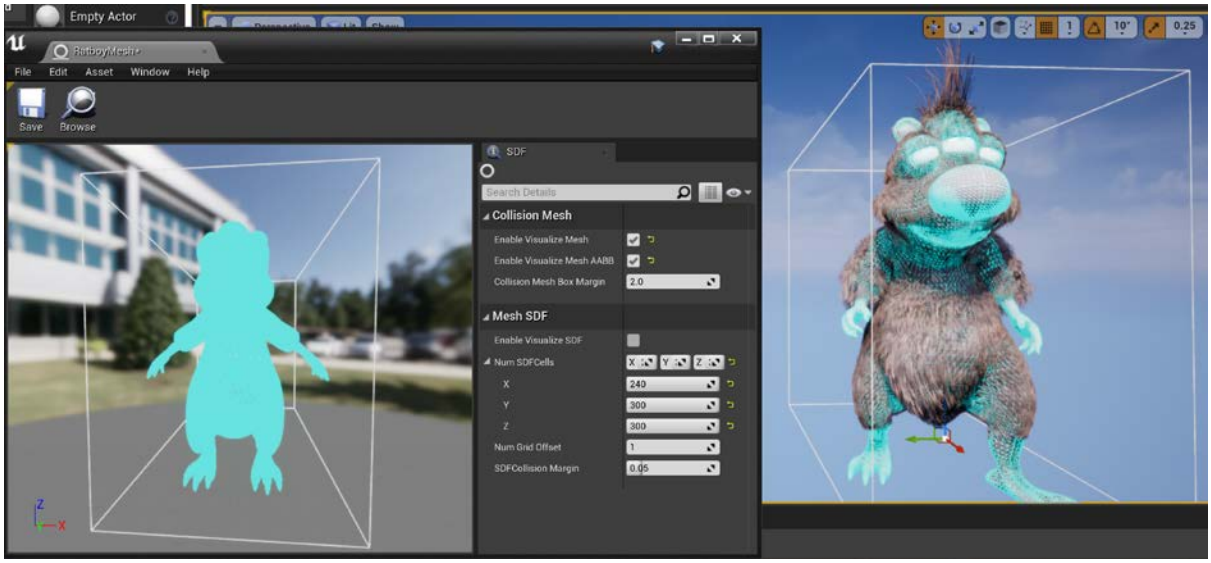
The magnitude of wind.

LocalShapeStiffness

The stiffness of local shape constraint.

TressFXMesh AssetEditor

Collision Mesh



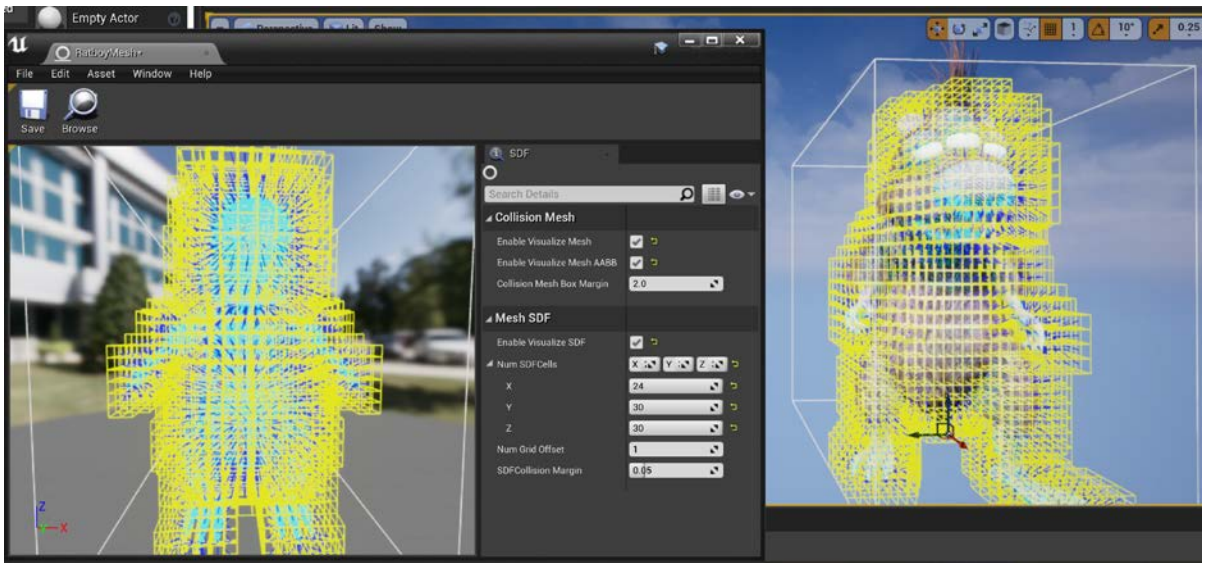
EnableVisualizeMesh

Visualize the collision mesh.

EnableVisualizeMeshAABB

Visualize the bounding box of the collision mesh.

SDF



EnableVisualizeSDF

Visualize the SDF of the collision mesh.

NumSDFCells

Resolution of the SDF.

NumGridOffset

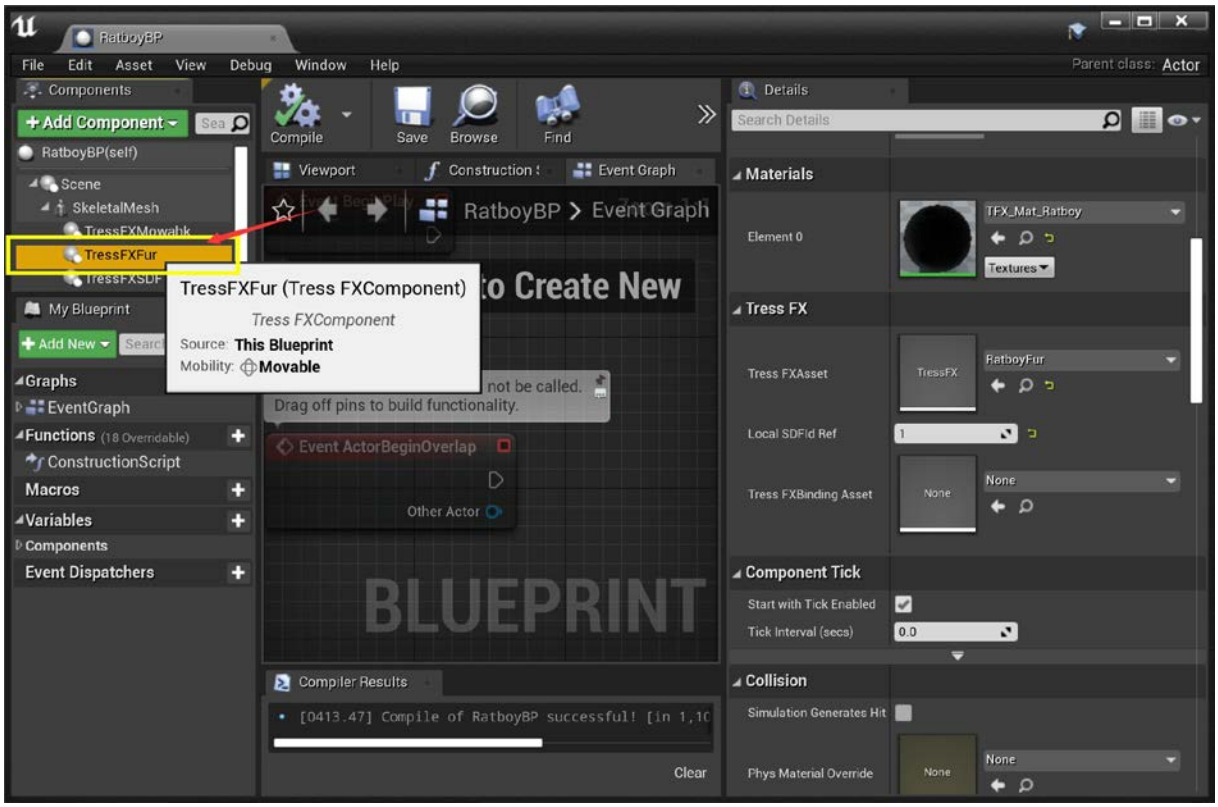
Controls the precision of SDF computation for triangles of the collision mesh.

SDFCollisionMargin

Controls the collision margin which starts to detect collision between SDF and guides vertex.

TressFX BlueprintEditor

TressFXComponent



Materials

All the materials of this TressFXComponent.

TressFXAsset

Choose and set the TressFXAsset for this TressFXComponent.

LocalSDFIdRef

The local SDF Id reference of TressFXSDFComponent. It corresponds to the local SDF Id in TressFXSDFComponent and turns on SDF for this TressFXComponent to avoid penetration.

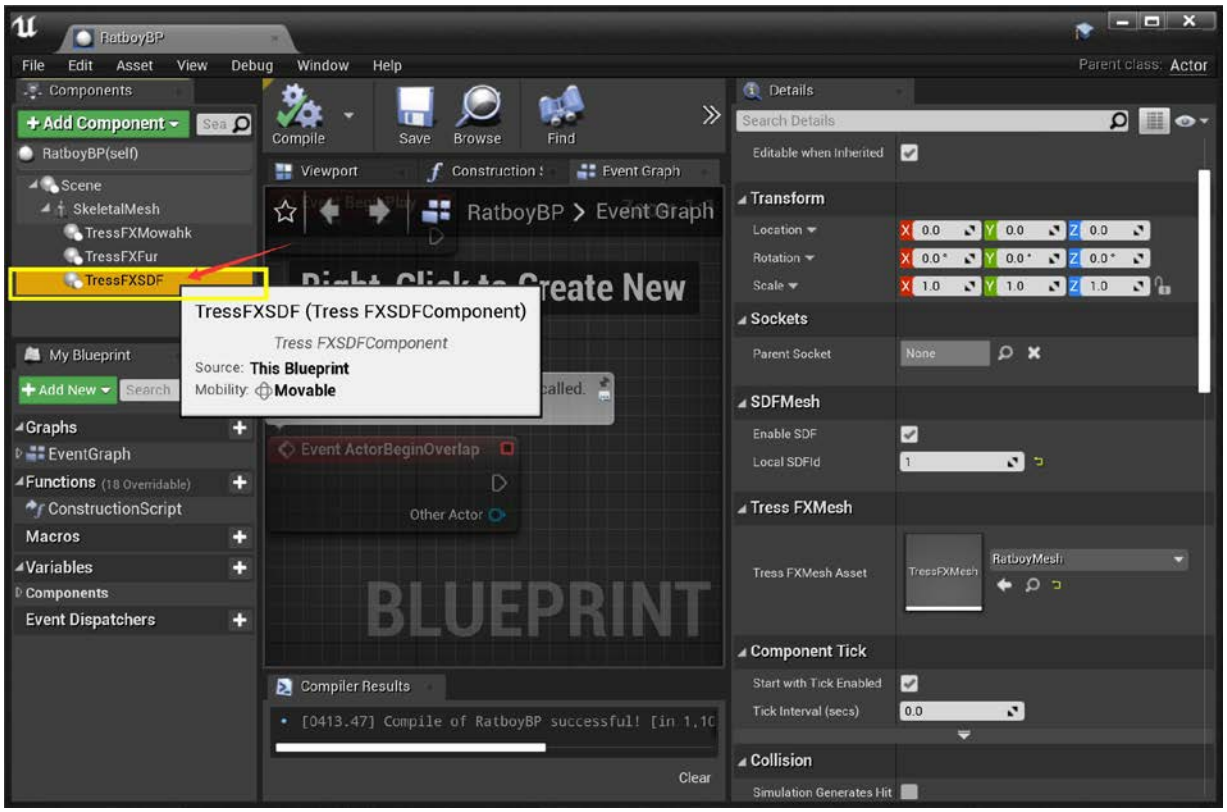
TressFXBindingAsset

Choose and set the TressFXBindingAsset for this TressFXComponent, to enable Triangle based Skinning for BlendShape animations support.

For a tutorial that walks you through creating a UE4 TressFX 5.0 component, see this tutorial in this document: Creating UE4 TressFX 5.0 Blueprint Actor.

For a tutorial that walks you through using UE4 TressFX 5.0 Triangle based Skinning, see this tutorial in this document: Using UE4 TressFX 5.0 Triangle based Skinning.

TressFXSDFComponent



EnableSDF

Turn on/off SDF.

LocalSDFId

The local SDF Id of this TressFXSDFComponent, it corresponds to the local SDF Id reference in TressFXComponent.

TressFXMeshAsset

Choose and set the TressFX collision mesh asset.

For a tutorial that walks you through creating a UE4 TressFX 5.0 SDF component, see this tutorial in this document: Creating UE4 TressFX 5.0 Blueprint Actor.

Lights

UE4 TressFX 5.0 supports 4 Unreal Engine light types:

- Point Light
- Directional Light
- Spot Light
- Sky Light.

The light (except Sky Light) should enable CastDeepShadow before it can render correctly with TressFX.

There is an upper limit of 32 lights supported across all applicable actors. As the DeepShadow pass is costly, we suggest reducing the number of lights for each TressFX actor.

Lastly, TressFX shadow casting lights must be Movable.

Note: Lights are added to the list of lights applying to TressFX as they are added to the scene (level). To remove a light, disable CastDeepShadow, or delete it (as opposed to hiding it).

Point Lights

The performance cost of these lights is close to Spot Lights because we implement the DeepShadow.

Spotlights

The performance cost of these lights is close to Point Lights because we implement the DeepShadow.

Directional Lights

We suggest using 1 Directional Light in your level

Sky Lights

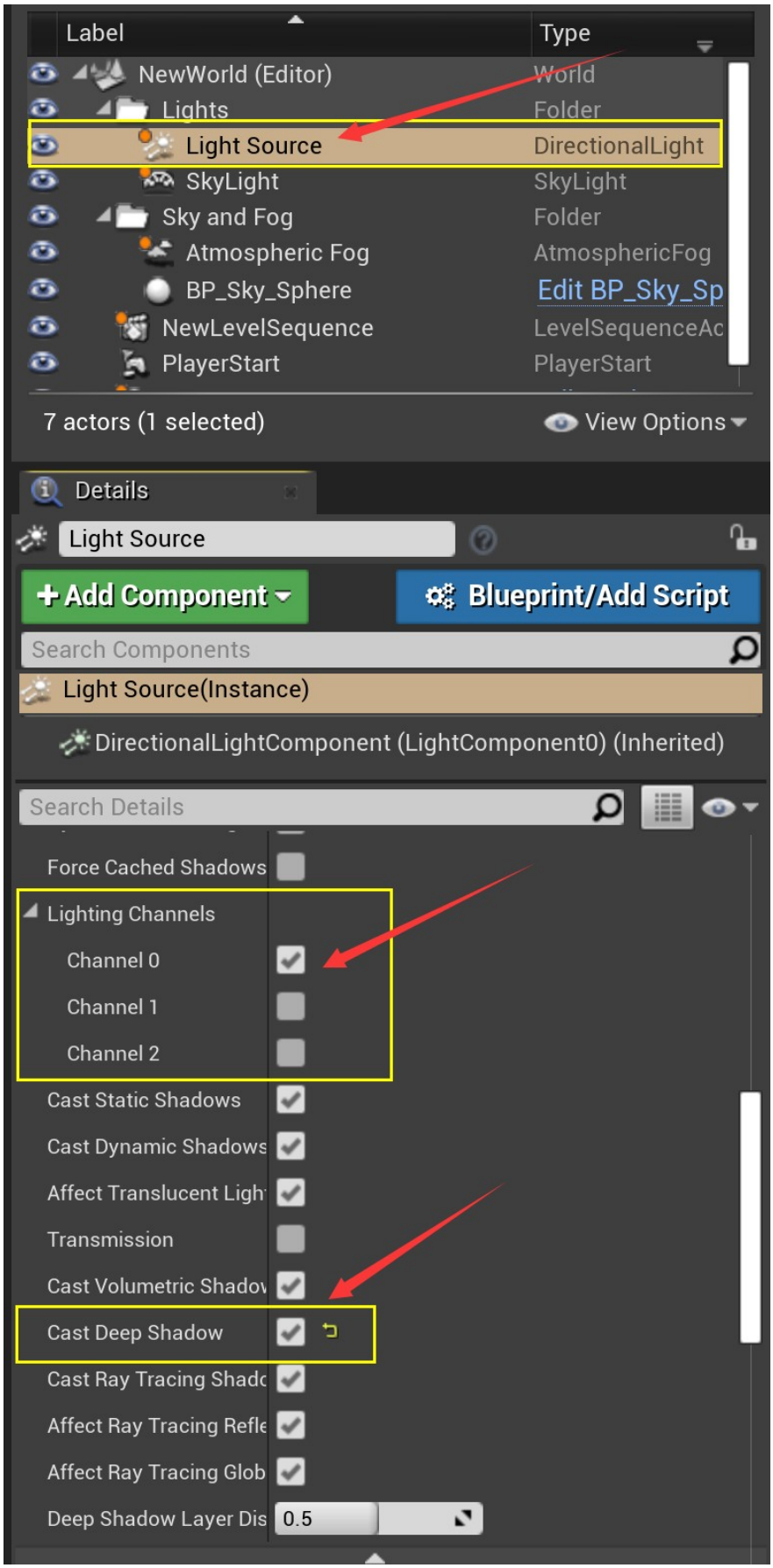
Sky Light doesn't need to set cast shadow and TressFX supports this type of light.

Lighting Channels

TressFX supports lighting channels. To enable light channels, set the *Lighting Channels* information under *Lighting* section of the selected light. Do the same on the TressFX Component (Lighting section, Lighting Channels).

Hint: click the Advanced Arrow (and the Up arrow at the base of the section, if not all of the items are showing).

It is generally recommended that you put TressFX-based lighting on a different channel than general lighting when adding an Environment or Sky Light.


Label
Type
NewWorld (Editor)
World
Lights
Folder
Light Source
DirectionalLight
SkyLight
SkyLight
Sky and Fog
Folder
Atmospheric Fog
AtmosphericFog
BP_Sky_Sphere
Edit BP_Sky_Sp
NewLevelSequence
LevelSequenceAc
PlayerStart
PlayerStart
7 actors (1 selected)
View Options
Details
Light Source
+ Add Component
Blueprint/Add Script
Search Components
Light Source(Instance)
DirectionalLightComponent (LightComponent0) (Inherited)
Search Details
Force Cached Shadows
Lighting Channels
Channel 0
Channel 1
Channel 2
Cast Static Shadows
Cast Dynamic Shadows
Affect Translucent Ligh
Transmission
Cast Volumetric Shadov
Cast Deep Shadow
Cast Ray Tracing Shadc
Affect Ray Tracing Refle
Affect Ray Tracing Glob
Deep Shadow Layer Dis
0.5

Art Export: Exporting TFXxxx files (TressFX Exporter plugin for Autodesk Maya)

Summary

TressFX is designed to be compatible with hair modeling DCCs. All modeling can be done within a modeling system of your choice, provided you can turn them into splines (NURBS curves) at the end. For demos using TressFX 5.0, we have used Shave and a Haircut for Maya (5.0/Ratboy), Autodesk Maya XGen (5.0 demos), and the Autodesk 3ds Max native hair modeler - although the 3ds Max plugin is no longer being developed and has not been included.

The Maya-based TressFX Exporter was tested on Maya 2015, Maya 2017, and Maya 2018. Maya 2019 was not tested, as at that time of writing, it had a critical bug with the XGen Guides menu which is a required element during XGen-based hair creation. Maya 2018 is recommended and was the version used to create TressFX 5.0 art assets.

For a tutorial that walks you through creating and exporting hair using the Maya TressFX Exporter, see this tutorial: Creating and Exporting TressFX 5.0 Hair from Maya.

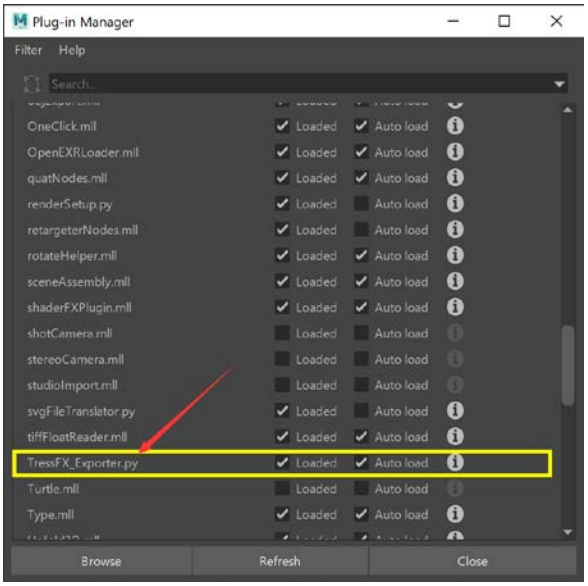
Installation of the Maya TressFX Plugin

The Maya-based TressFX exporter is a single python file residing in the following location:

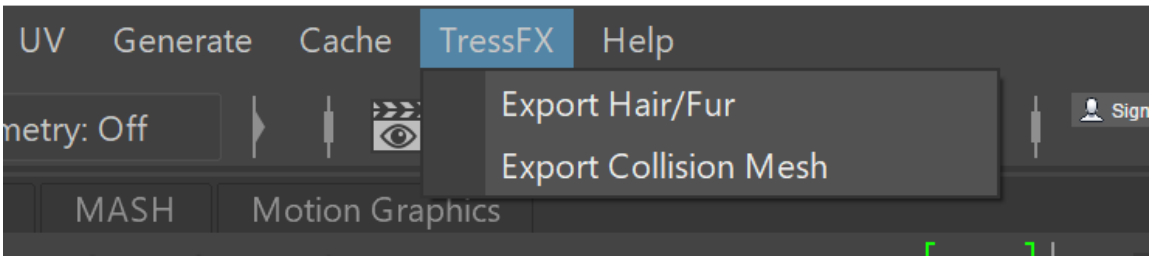
- TressFX/Unreal: **Tools\Maya2018\TressFX_Exporter.py**

To enable this plugin, follow these steps:

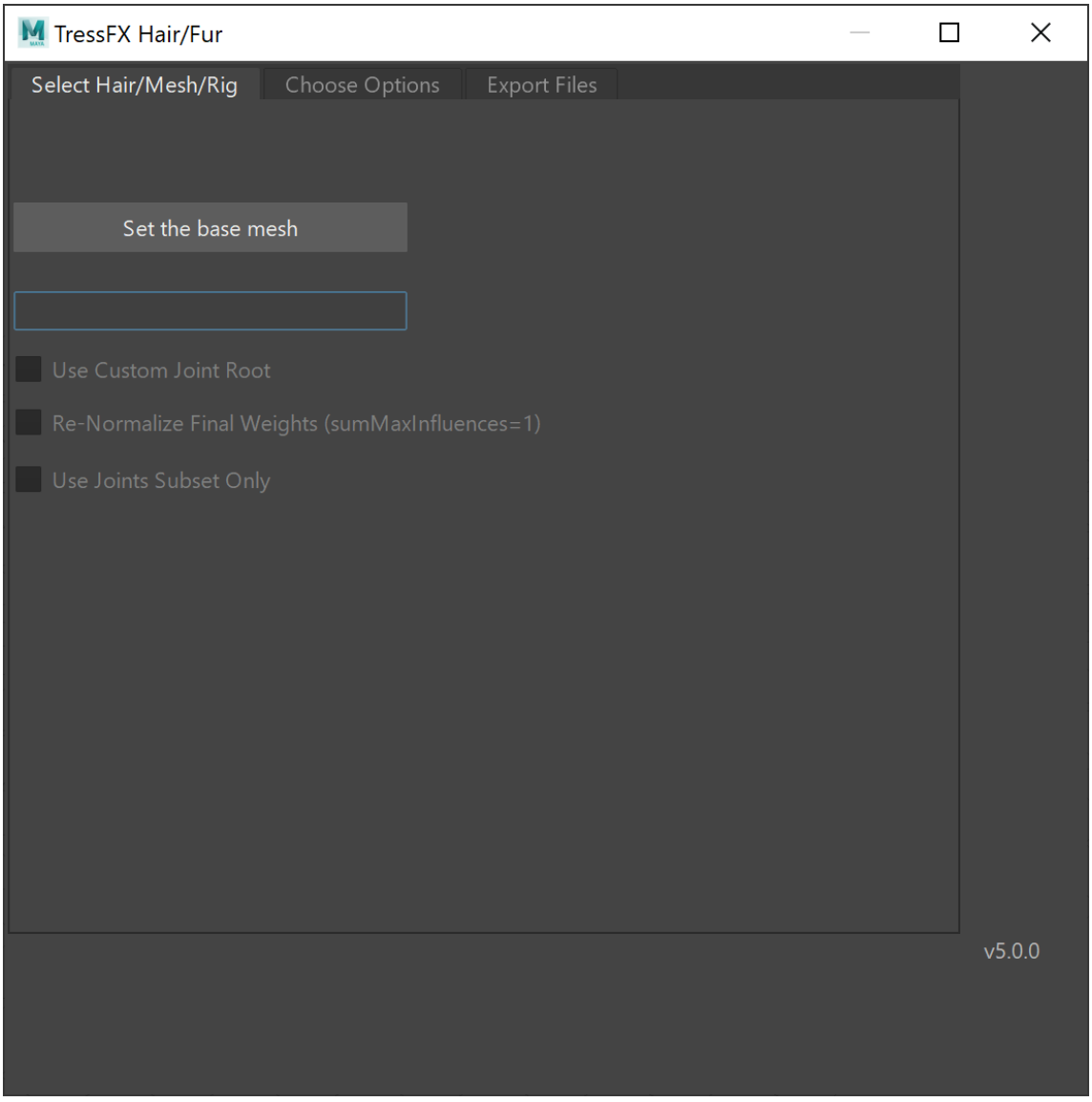
1. Copy TressFX_Exporter.py into Maya's plug-ins folder such as **C:\Users\USER_NAME\Documents\maya\plug-ins** or the main plug-in folder, such as **C:\Program Files\Autodesk\Maya2018\bin\plug-ins**.
2. Launch Maya.
3. Open the Plug-in Manager and enable the *Loaded* and *Auto load* options for TressFX_Exporter.py.



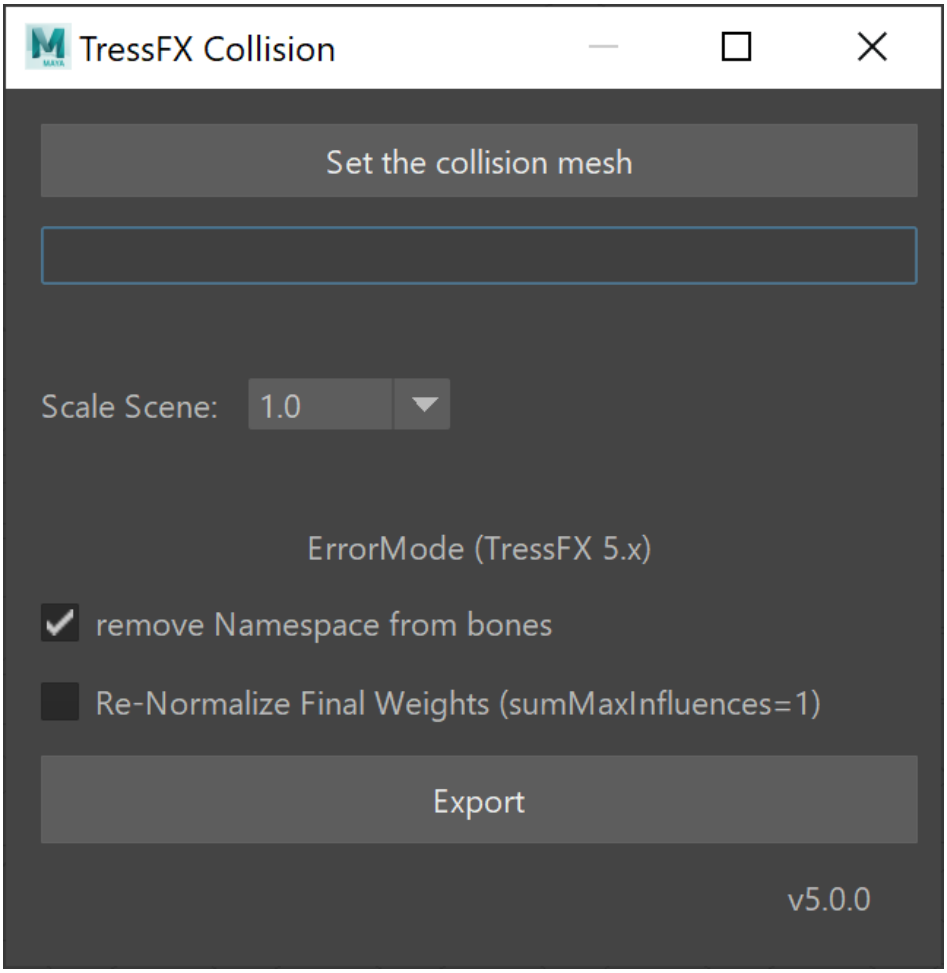
4. Now, a TressFX menu should appear on the main menu bar. Underneath it, there should two sub menu items: *Export Hair/Fur* and *Export Collision Mesh*.



5. The **Export Hair/Fur** menu item will bring up the **TressFX Hair/Fur** window.



6. Export Collision Mesh will open the TressFX Collision window.



Exporter Settings

The UI for hair/fur export is in a tabbed format. Also, the exporter includes error checking and control features.

Warnings are displayed in the status bar as well as the script window. Print messages (informational) are also displayed in the script window.

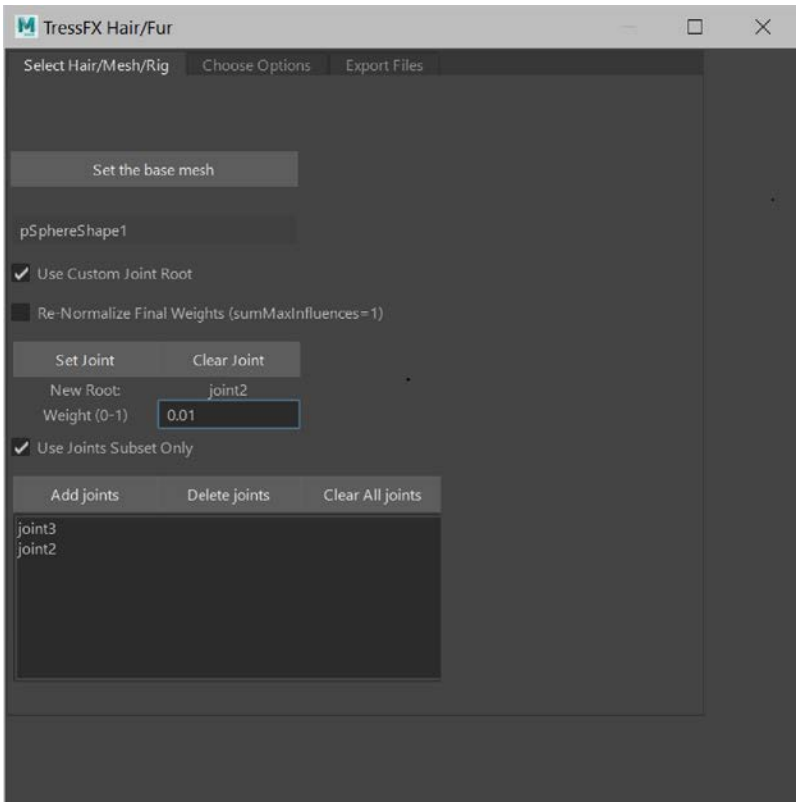
Many settings will not work unless a base mesh is selected. Export will also not work unless a set of splines is selected for export. You typically do not need to select all the splines individually (i.e. Select Hierarchy in Maya) – the exporter will do a recursive search during export, so selecting a grouped set of splines (a group) is simply a matter of selecting the containing group. Tools like XGen, when converting XGen groomable hairs into splines, will automatically 'group' the selection.

See the tutorial for a walkthrough of this process. A basic knowledge of Maya is assumed, including how to use the Outliner window and find the Plug-in Manager menu item.

The following section will go through all of the options in each of the tabs of the hair/fur export UI.

Hair Settings

(SELECT HAIR/MESH/RIG) TAB GROUP 1:



Set the base mesh (button)

Select a mesh, then click the button to set the reference mesh (shape) that will be used for .tfx and .tfxbone (hair data) export. The selected mesh name will appear in the text box below the button.

Use Custom Joint Root (checkbox)

If selected, you can choose a different joint in the base mesh's skeleton to use as the default root joint. You can also set the value for this weight. This joint and its weight will only be used if there is no other joint found for a given point, filling out the 4 joint influencer list for each position. This is useful in the case where a hair strand is not tracking, typically because the joints nearest the root position of the strand are zero (or because of barycentric calculations sending the weight to zero).
Often many joints in a complex skeleton are not used for the body animation (such as facial bones).

Note: This custom root joint also needs to be **added** to the Joint Subset ***if the Joint Subset is being used***.
It is not added automatically at this time.

Set Joint (button)

Select a joint in the Outliner, then click this button to set that joint as the default root.

Clear Joint (button)

Clear the default root back to the default (joint 0 in the hierarchy).

Weight (0-1) (float)

Set the default weight for the chosen root. If you also check **Re-Normalize Final Weights**, then all four weights will be normalized so that they sum to one (1).

Re-Normalize Final Weights (sumMaxInfluences = 1) (checkbox)

Forces the four bone weights per strand vertex to sum to one. Divides each weight by the sum of the four weights. Often, this is not needed, but can be used to ensure normalization.

Use Joints Subset Only (checkbox)

Use only a selected subset of the joints attached to the base mesh. This is useful for hair tracking issues where a nearby joint is influencing the overall weighting but shouldn't. For example, you might want to include only the arms and legs for a particular hair asset group (arm and leg hair as one hair asset) and exclude other nearby bones (like facial bones or the chest or even a belt/clothing items) or bones that should only be used for non-skin movement (like a weapon).

Add joints (button)

Select a joint in the Outliner, then click this button to add that joint to the subset to be used.

Note: If you are also using a custom root, be sure to add that joint to the subset as well. Currently, it does not happen automatically.

Delete joints (button)

Select a joint (or joints) from the joint list, then click this button to remove them from the list.

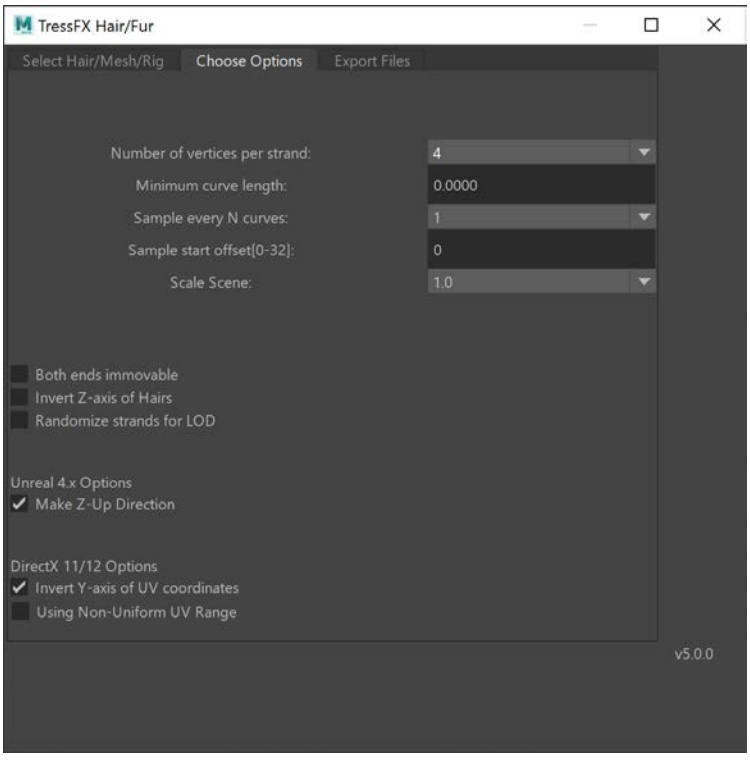
Clear All joints (button)

Click this button to clear all the joints from the joint list.

Joint list area (multi-selection listbox)

This is a multi-selection listbox (single column currently) that shows the current joints in the joint subset to be used.

(CHOOSE OPTIONS) TAB GROUP 2



Number of vertices per strand (dropdown)

The number of vertices to use to define the curve shape of a strand, the larger the value, the more definition the strand would have but at a greater hair asset size and processing (performance) cost. Generally, shorter hair looks better at 8 vertices per strand and long hair at 32, to keep a smoother curly look may need 64 vertices per strand.

Minimum curve length (float)

The exporter uses this value to filter out hair that is shorter than this length. In some situations, it may be difficult to get rid of short hair within the modeling tool. If the value is set to 0.0 (default), no filtering will take place.

Sample every N curves (dropdown)

The exporter will only export every Nth strand. This can be useful when you have a lot of hair (splines) modeled, often more than really needed or wanted for good performance, and just needs a quick way to reduce the number of strands exported.

Note: all hair exported using the Exporter will be guide hair on import to TressFX/Unreal and (currently) to TressFX/Cauldron. Follow hairs are generated on import into those engines, but not during export.

Sample start offset[0-32] (integer)

Used in conjunction with subsampling (**Sample every N curves**). This lets you specify where to start the subsampling process. The default is zero (0), i.e. the first strand.

Scale Scene (dropdown)

Scales the points by multiplying the scale value against each .x, .y, .z position value. A value of 1.0 is fine for Unreal Engine (cm), but Cauldron uses meters, so you need to add a scaling value of .01 to scale the data from cm->m.

Both ends immovable (checkbox)

Sets both ends of the strand (the endpoint vertices) to use zero inverse mass, which is kept in the w position coordinate (.w). This forces both ends, not just the 'root', to be immobile, although both will still track with the mesh skin itself. One example would be a loop on a mesh, where the middle of the loop can move freely but both ends are firmly fixed to the mesh itself.

Invert Z-axis of Hairs (checkbox)

Inverts the Z component of the hair vertices. This may be useful if dealing with an engine that uses a left-handed coordinate system.

Randomize strands for LOD (checkbox)

Randomizes hair strand indices so that any LOD done on strands in-engine would uniformly reduce hair. Not generally needed if using a hair creation tool like XGen, since it can randomize as it distributes groomable hair.

Make Z-Up Direction (checkbox)

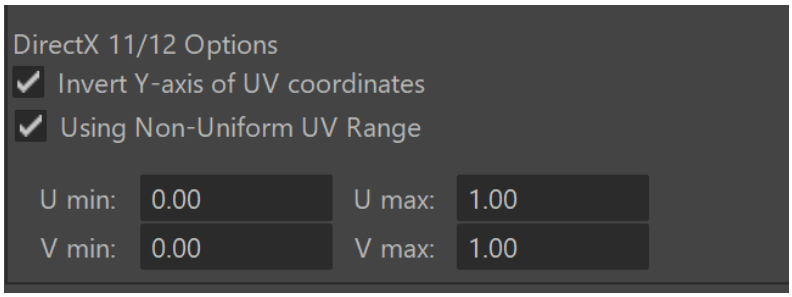
Commonly used/needed when exporting for Unreal Engine use. Maya typically uses Y for the up direction, while Unreal Engine uses Z for the up direction. If Maya is not set to use Z as the up direction, this will swap the Y and Z values for you before export. It will not alter Maya settings or the Maya scene/DAG.

Invert Y-axis of UV coordinates (checkbox)

Inverts the Y (i.e. v) component of UV coordinates. DirectX® 11/12 needs the Y-axis inverted for proper UV alignment. This is a required option when exporting for use in the TressFX/Cauldron implementation.

Using Non-Uniform UV Range (checkbox)

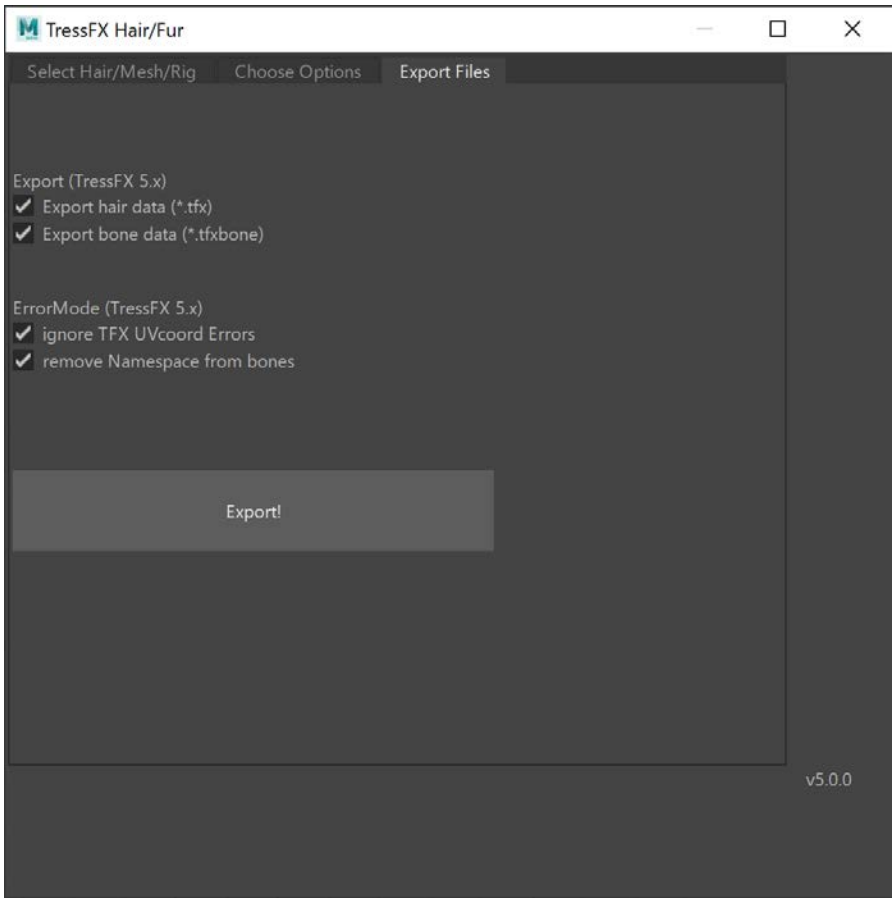
If inverting the Y-axis component of UV coordinates is selected, but the UV mapping is non-uniform (u:0-1, v:0-1 is uniform), this control lets you specify the UV mapping that your mesh is using. The v min and max values from this will be used during the Y-axis invert for v, instead of the default values of (0,1).



Note: This user-defined V min/max range is only used if **Invert Y-axis of UV coordinates** is selected**.**

Note: The U values are also taken but not used at this time. However, an informational print statement to the Maya Script window will reflect these changes, as well as a cmds warning that tells developers that they are choosing to use a non-uniform UV range.

(EXPORT FILES) TAB GROUP 3



Export hair data (*.tfx) (checkbox)

Export a binary TFX file that contains hair strand and vertices data. Usually paired with a TFXBONE file, since both are needed for import and should match, with matching names but different extensions.

Export bone data (*.tfxbone) (checkbox)

Export a TFXBONE file that contains bone animation data (see .tfx above).

ignore TFX UVcoord Errors (checkbox)

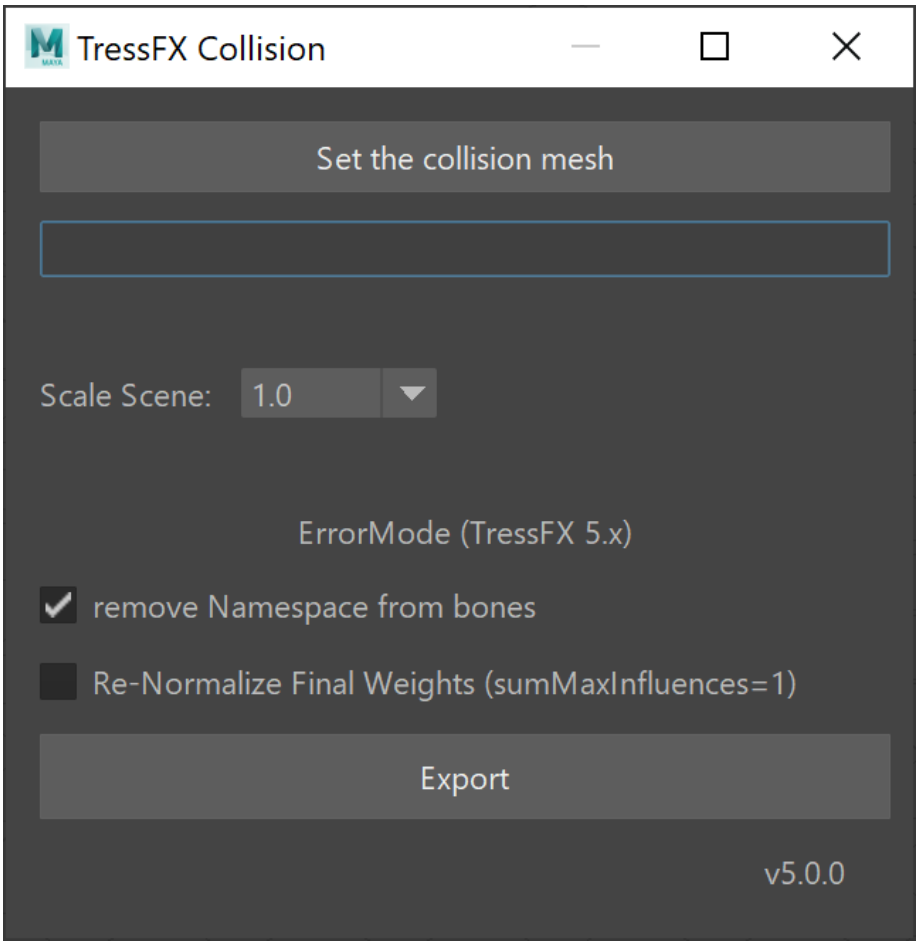
Do not warn or fail if there are any issues with the UV coordinate look up during export (see the Exporter code for more information). This helps when trying to determine any issues with a skinned mesh model.

remove Namespace from bones (checkbox)

If there is a namespace on joint names, such as *myImportedBones:joint0*, this will remove the namespace from the string, leaving simply, the joint name, i.e. *joint0*. This is important if the skinned mesh being used in the engine does not have any namespace on the joints. This situation can happen when importing an FBX or another Maya file into an open Maya file.

If the joint names do not match between the skinned mesh and the exported TFX files, then the simulation will not be able to find a joint name or might find the wrong one if there is a match. Joint names should be unique within a skeleton.

Collision Settings



Set the collision mesh (button)

Select a mesh, then click the button to set the reference mesh (shape) that will be used for .tfxmesh (collision mesh data) export. The selected mesh name will appear in the text box below the button.

Scale Scene (dropdown)

Same as for hair settings, will scale the points by multiplying the scale value against each .x, .y, .z position value. A value of 1.0 is fine for Unreal Engine (cm), but Cauldron uses meters, so needs a scaling value of .01 to scale the data from cm->m.

Remove Namespace from bones (checkbox)

Same as for hair settings, if there is a namespace on joint names, such as *myImportedBones:joint0*, this will remove the namespace from the string, leaving simply, the joint name, i.e. *joint0*. Important if the skinned mesh being used in the engine does not have any namespace on the joints. This situation can happen when importing an FBX or another Maya file into an open Maya file.

If the joint names do not match between the skinned mesh and the exported TFX files, then the simulation will not be able to find a joint name or might find the wrong one if there is a match. (Joint names should be unique within a skeleton.)

Re-Normalize Final Weights (sumMaxInfluences = 1) (checkbox)

Same as for hair settings. Forces the four bone weights per strand vertex to sum to one. Divides each weight by the sum of the four weights. Often, this is not needed, but can be used to ensure normalization

Tutorials

A Quick Tutorial on Creating a Basic Skeletal Mesh in Maya

Introduction

TressFX 5.0 hair is bound to a skeletal mesh, meaning a mesh that is bound to a skeletal rig (bones). Most users of TressFX will hire experienced modelers and animators to create their rigs, and ultimately create guide hairs attached to that skeletal mesh. However, for testing purposes it is useful to be able to create the minimum required skeletal mesh.

This tutorial will walk you through the basics of creating a sphere mesh bound (skinned) to a simple joint chain (a hierarchy of five joints...aka bones). This is the exact same skeletal mesh that is used in the Maya tutorial on creating and exporting TressFX hair.

Related Links:

Creating and Exporting TressFX 5.0 Hair from Maya

Creating UE4 TressFX 5.0 Blueprint Actor

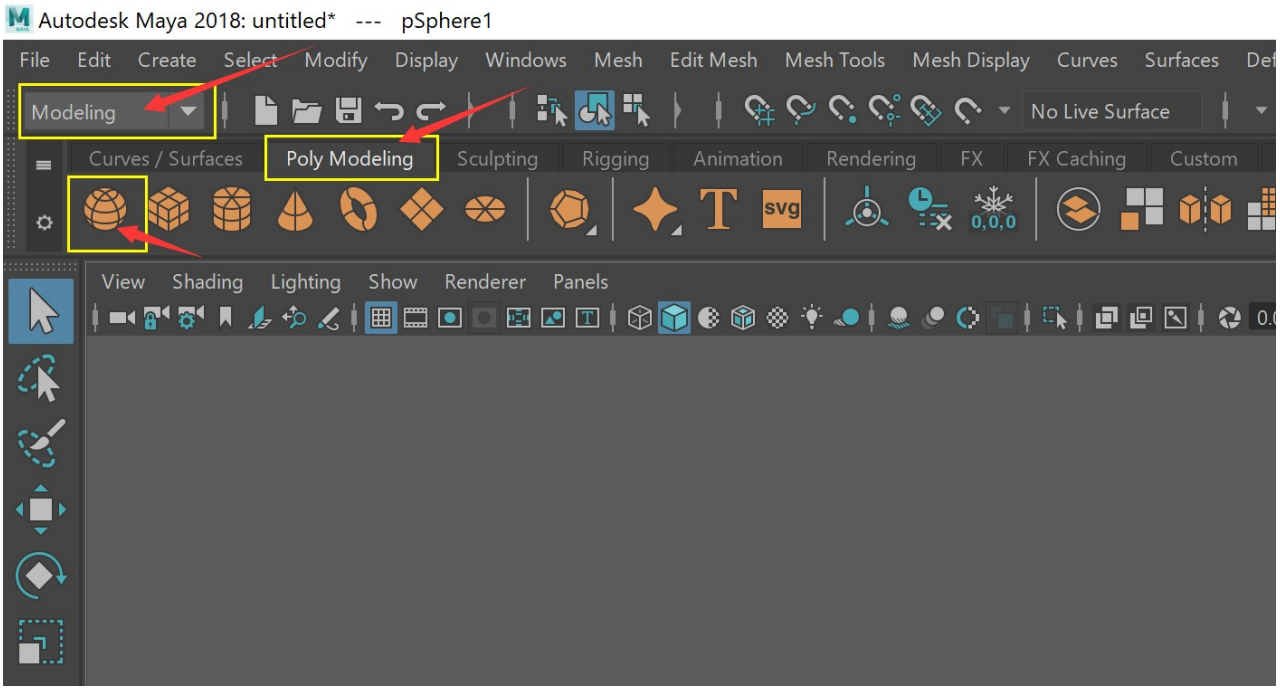
Requirements

- Autodesk Maya 2015 or higher. This tutorial uses Maya 2018, but 2015 is the minimum based on TressFX/XGen usage)
- A basic understanding of how to launch and navigate in Maya (using the move/rotate/scale tools, zooming and panning in the main perspective view)

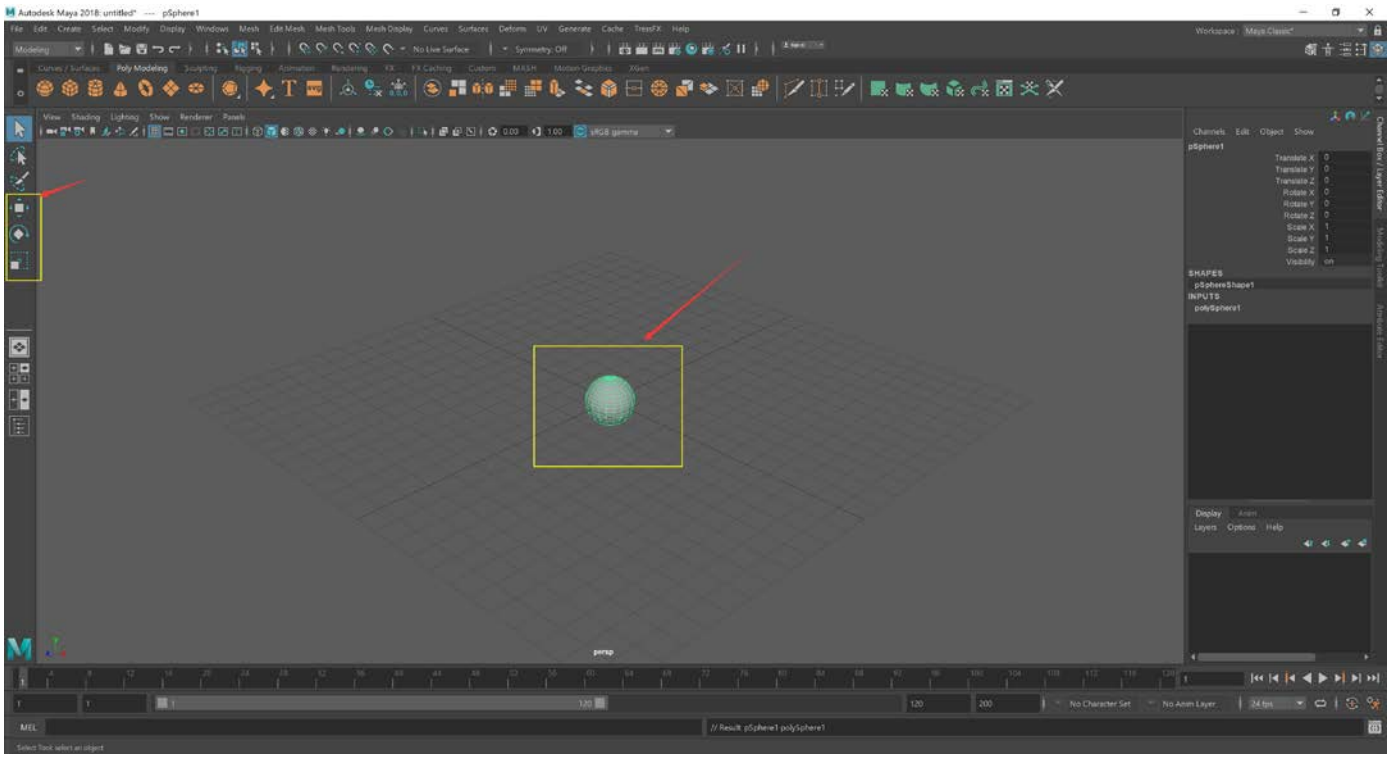
Step 1

Open a new scene in Maya and make sure you are in the **Modeling** mode. When opening Maya, a new scene is automatically generated for you.

Open the **Polygons** tab and click **Sphere**.

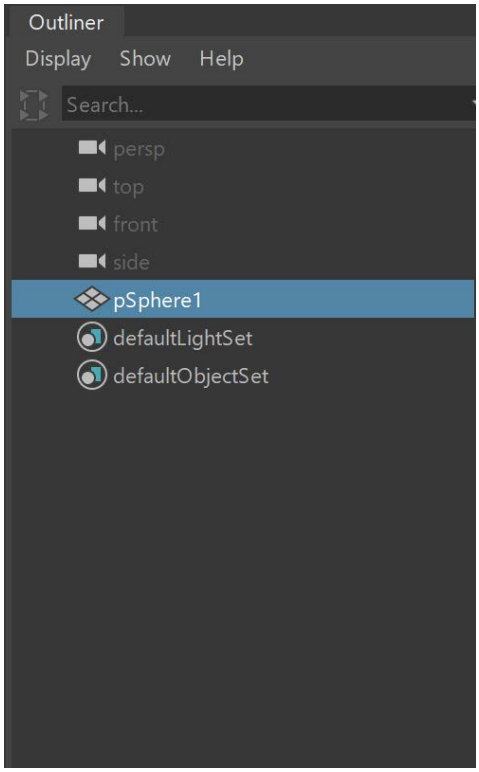


You should see in the main *perspective* view that a sphere is created.



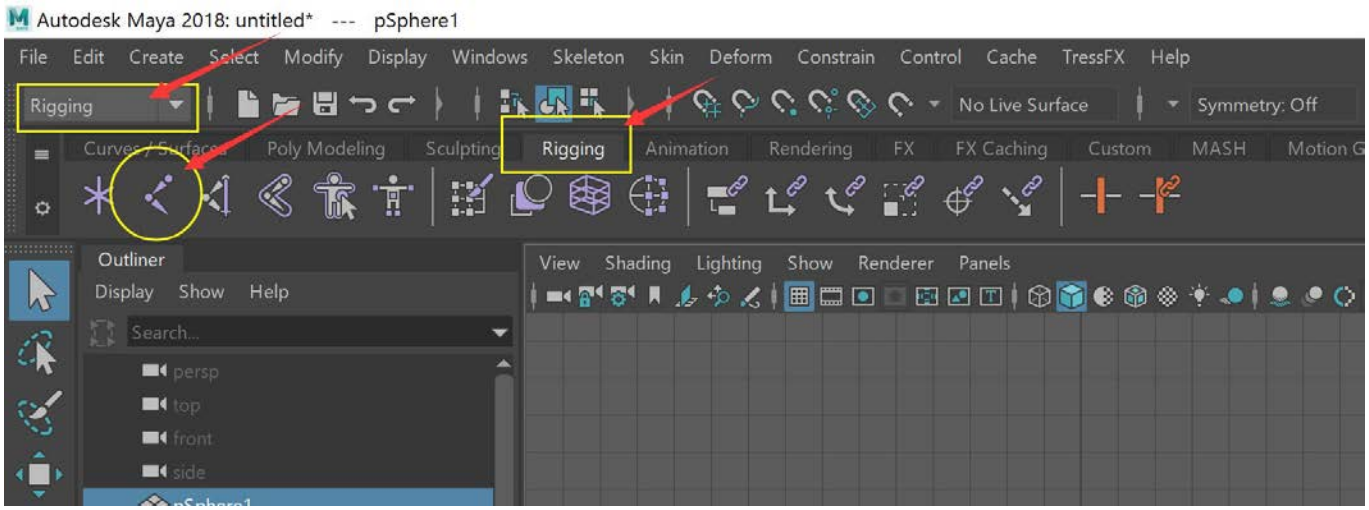
You can either make the sphere larger using the **Scaling** tool, or you can *zoom in* so you can see the sphere more easily.

If it is not already open, open the **Outliner Window** by selecting **Windows > Outliner** from the top-level menu**.** You will need this window to move the joint chain, select the joint hierarchy and mesh for binding, etc.

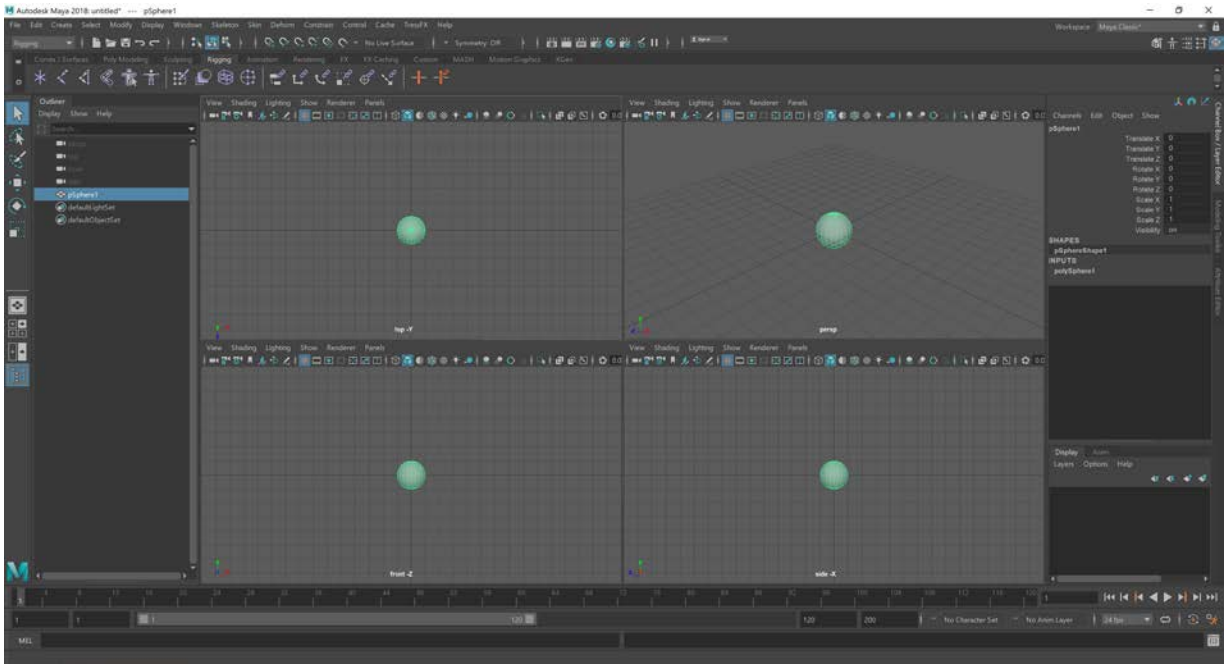


Step 2

Change to the **Rigging** mode.



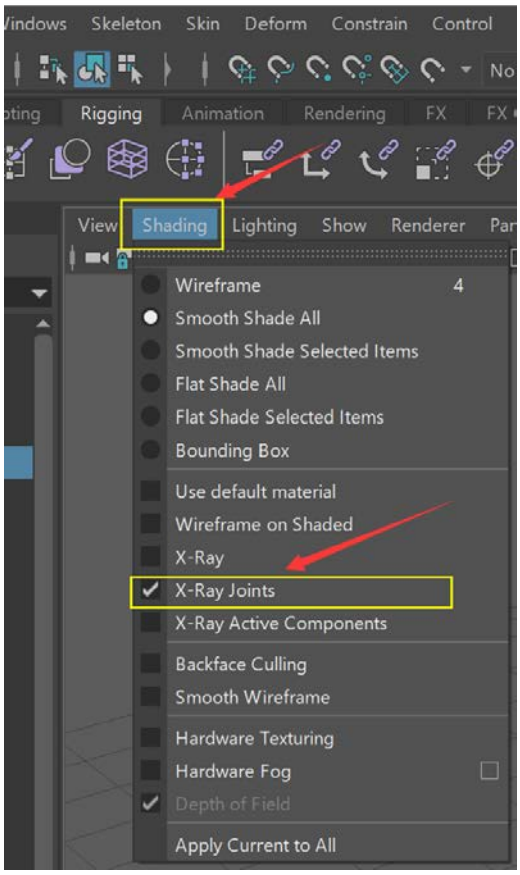
In the main *Perspective* view, tap the view to make sure it has the focus, then tap the **Spacebar**. This should send you into a four-way view: *Perspective, top, side, front.*



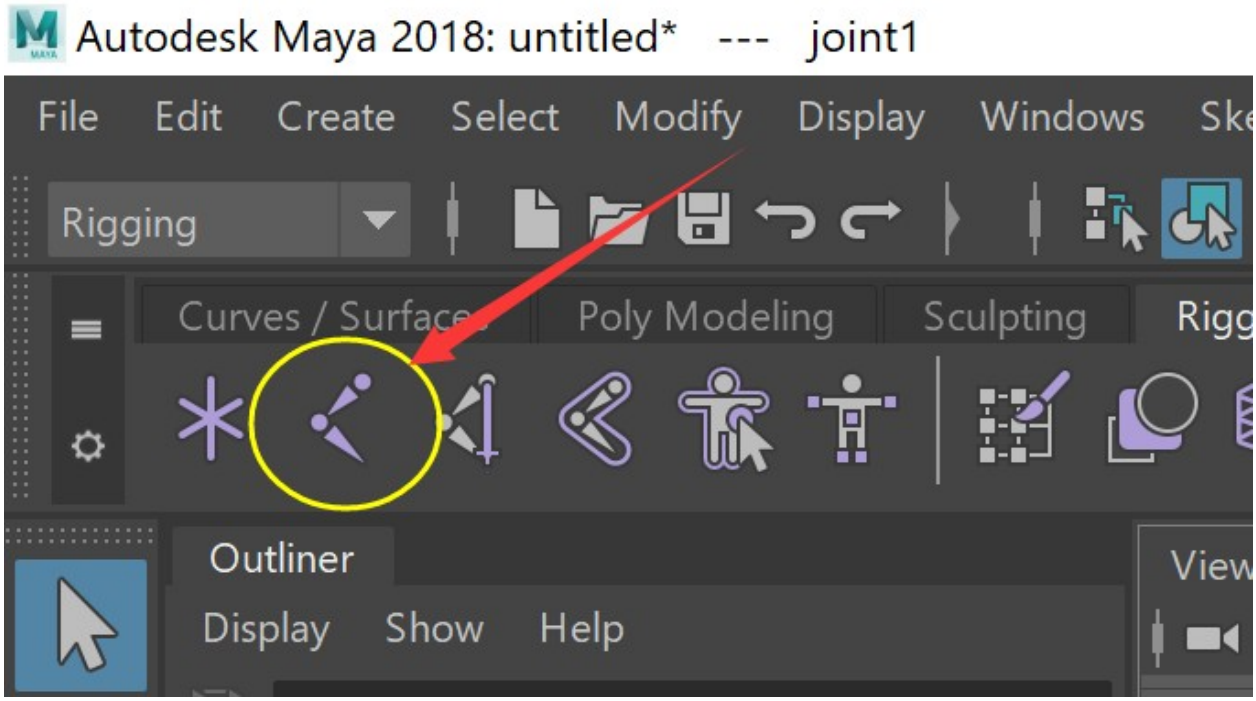
Click anywhere in the *front* view. This will activate the tool in that view.

On the **Shading** tab of the view, choose **X-Ray Joints**.

You *may* need to do this for each view you want in order to see the joints through a solid mesh. You can try to set the perspective view to X-Ray Joints, when it is the only view — not four way — and see if this setting propagates to all the views. If it doesn't, set each one individually.

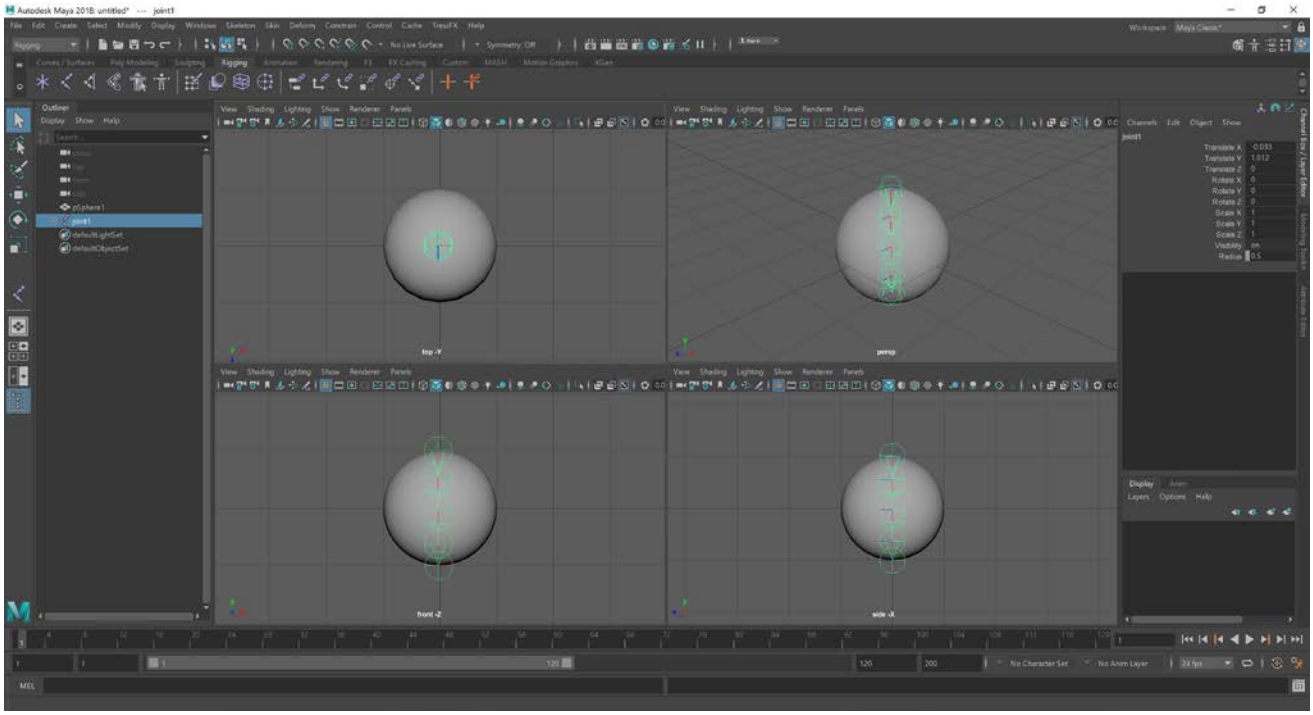


Click the **Joint** tool.



Each click will create a bone. The first click will be the first bone, the next click will be the second that is automatically parented (a child of) the first bone, and down the chain.

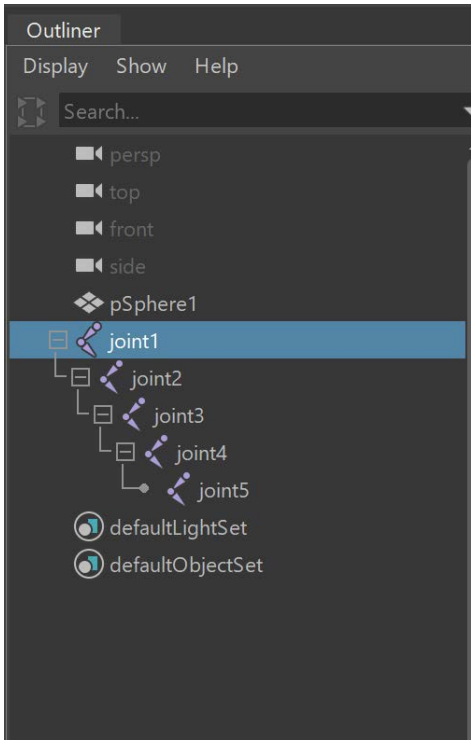
In the view, along the sphere, make five bones in roughly a long line, keeping them within the bounds of the sphere.



Click **Return** to stop the chain creation.

Look at the *side* and *top* views. If the bones are not aligned inside the sphere, click the *top of the joint chain* in the **Outliner**, then click the **Move** tool and then click in the appropriate view window (*top, side, front*) to move the joints to where you want them.

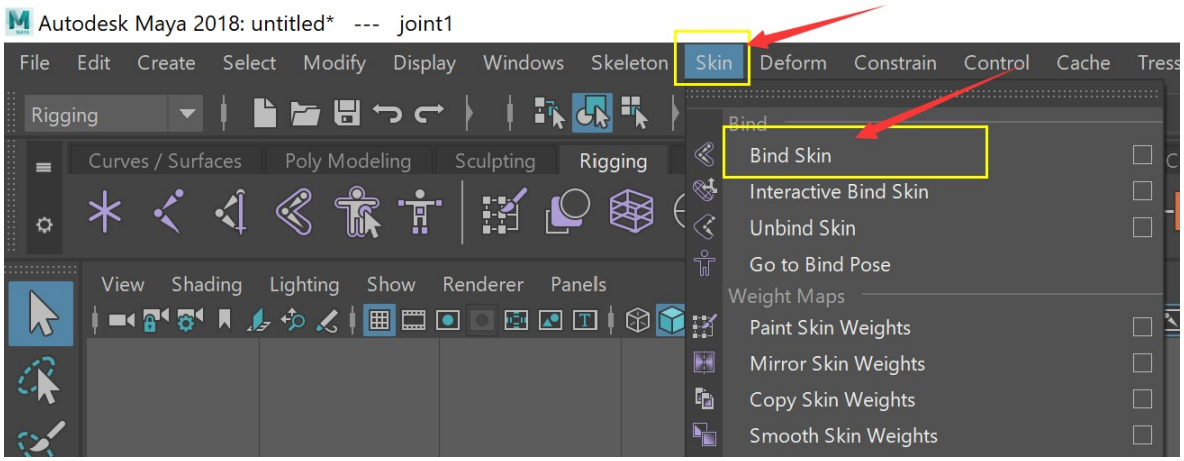
Generally, it's better to have joints within a mesh. However, experienced riggers will often do different configurations. This is just a simple skeletal mesh, so we will stay with the basics and let Maya do the rest by using its defaults.



Step 3

When you are ready, click the sphere mesh in the **Outliner**, then press and hold *Shift* and click the top joint in the joint hierarchy. You are now in **Rigging** mode.

Go to the menu item **Skin** and select **Bind Skin**.

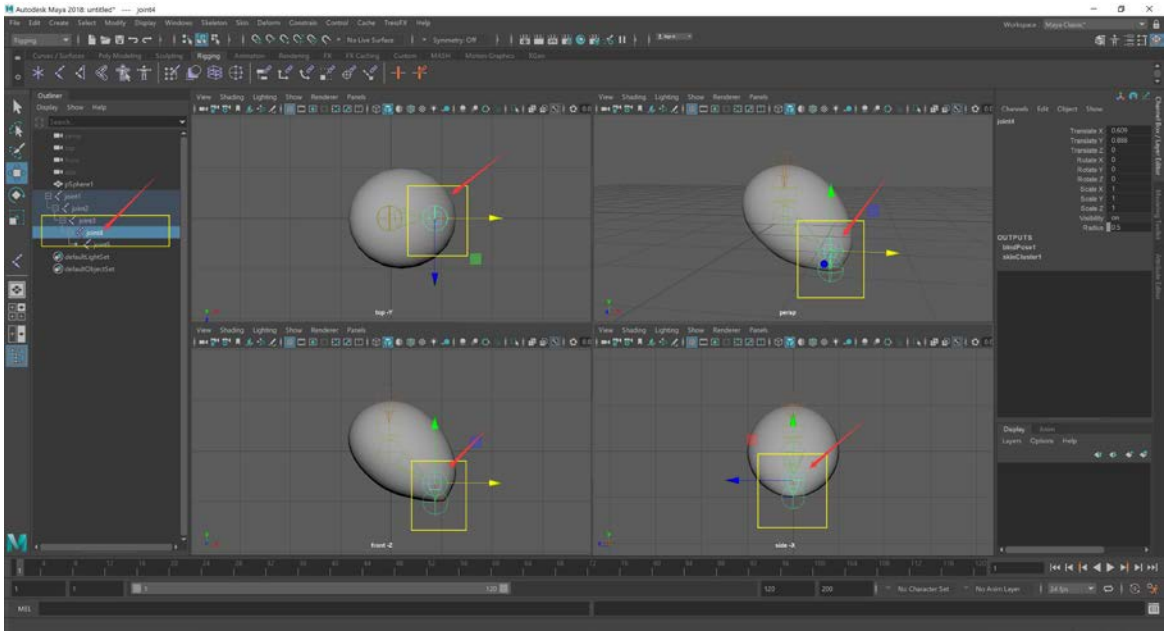


The joint chain should be bound to the mesh now.

To test this, in the **Outliner**, select one of the joints, and use the **Move** tool to move the joint. Not only should the children of that joint in the chain move (if any children), but also the skin should warp and move as well.

In the picture below, joint 5 is being moved. The skin stretches and deforms. Try other joints and other ways to manipulate it (move or rotate, for the most part).

Note: If you scaled the sphere larger, you may notice the bones appear smaller than in these pictures. That is normal.

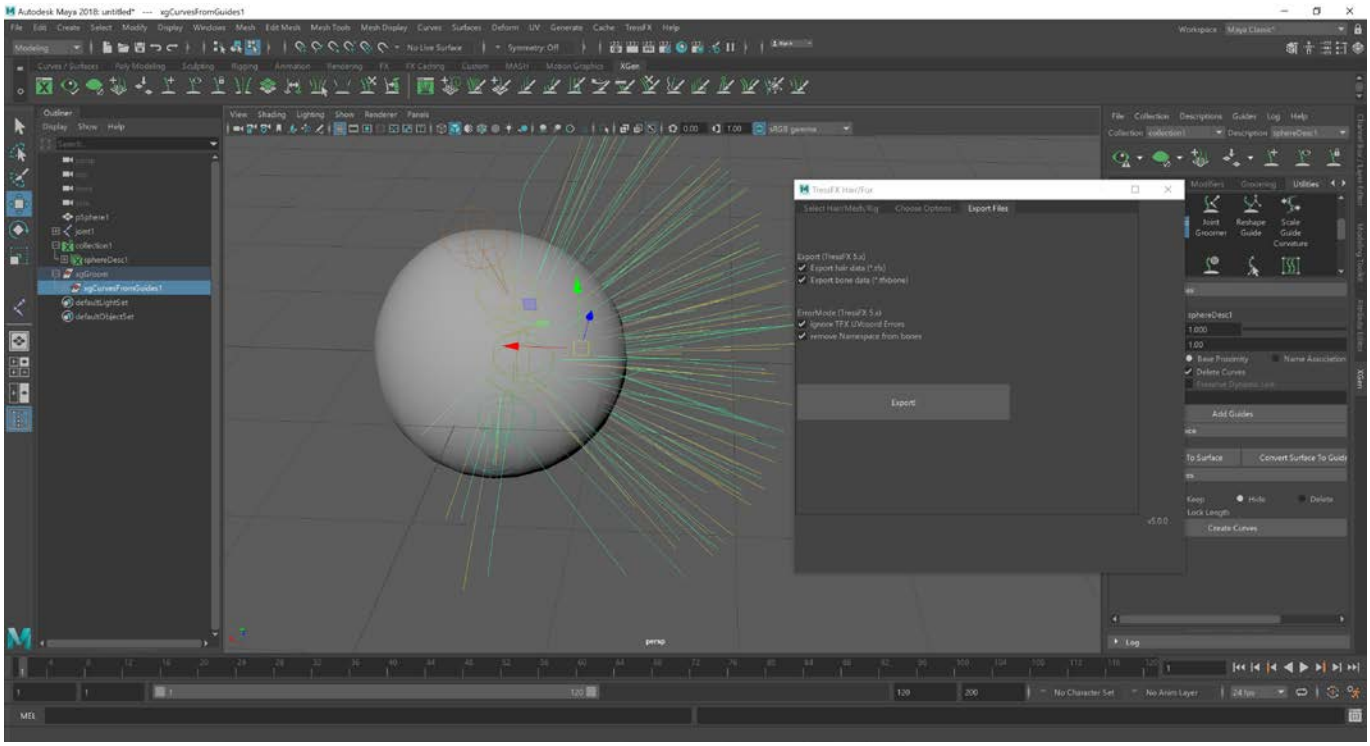


You are now ready to use this simple skeletal mesh — to animate it, add hair, and so on.

Creating and Exporting TressFX 5.0 Hair from Maya

Introduction

The following is a quick walkthrough on how to use Maya's XGen to create 'guide hairs' and also how to export those ‘guide hairs’ using the TressFX for Maya Exporter. The files produced (.tfx, .tfxbone and .tfxmesh) can then be used, along with the skeletal mesh, in the TressFX Unreal Engine 4 build to create a skeletal mesh asset with hair.



TressFX shader parameters have been set in that Unreal Engine skeletal mesh to be wide strands, few strands, and globally stiff so the original shape in Maya is easy to discern.

Related Links

Creating UE4 TressFX 5.0 Blueprint Actor

A Quick Tutorial on Creating a Basic Skeletal Mesh in Maya

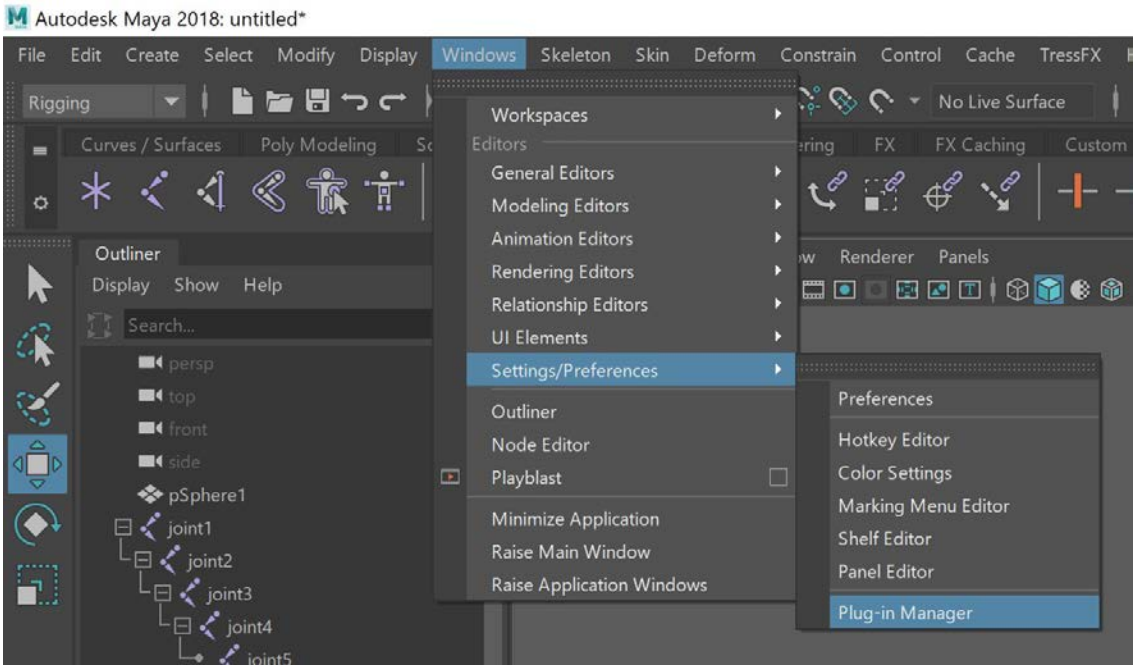
Requirements

- Maya 2017 (you can also use Maya 2015 or Maya 2018 or later as well). This tutorial uses Maya 2018.
- The TressFX Exporter for Maya (TressFX_Exporter.py).
  Note: The Exporter was updated. Be sure to use the most recent Exporter (TressFX 5.0).

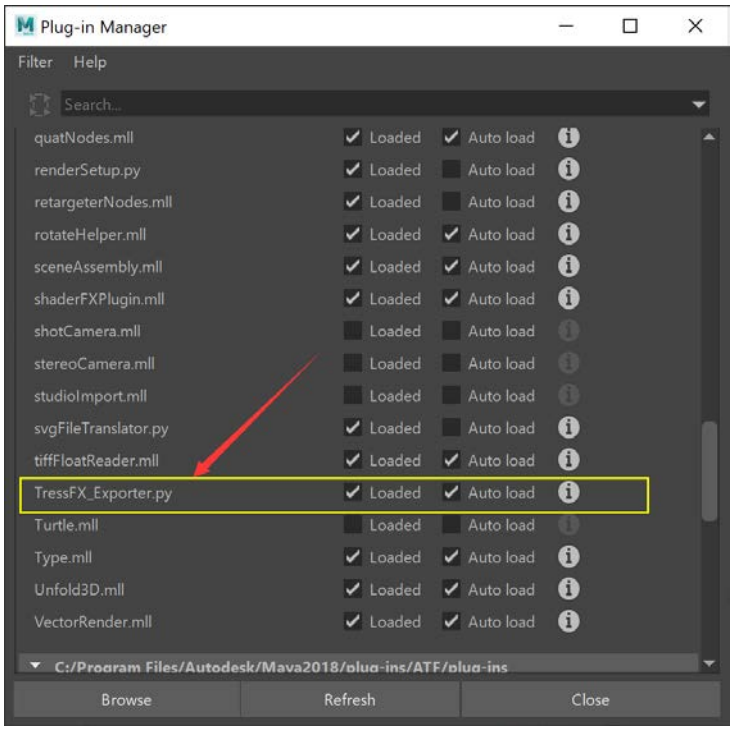
Step 1

Install the TressFX Exporter as a plugin.

An easy way to do this is to take the Exporter, which is a Python script, and copy it to the **bin\plugins** directory of Maya. For example, if you installed Maya in **C:\Program Files\Autodesk\Maya 2018**, the Exporter should be copied to **C:\Program Files\Autodesk\Maya 2018\bin\plugins**.

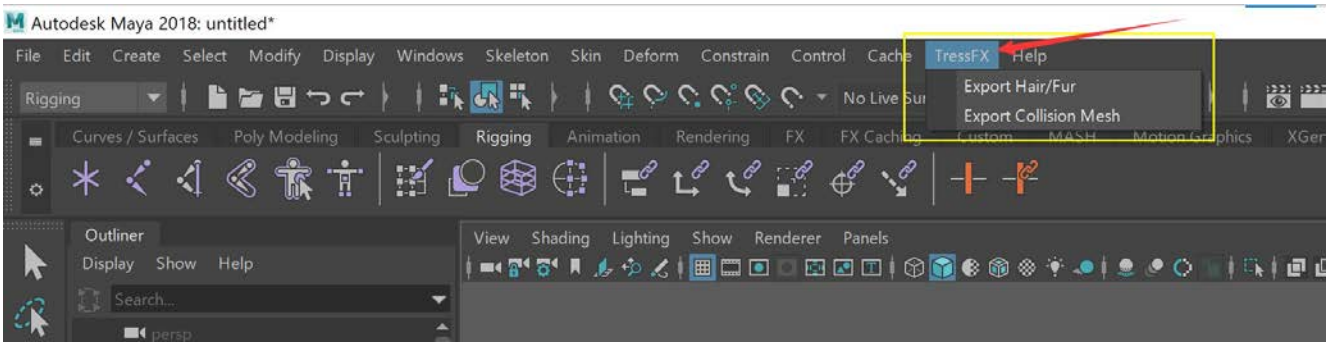


Once the Exporter has been copied, launch Maya and open the Plugin Manager. The Plugin Manager can be found in **Windows > Settings/Preferences > Plug-in Manager**.



Scroll down until you find the TressFX plugin. Select both the Load and Autoload options and then close the window.

You should now see the TressFX menu on the top-level menu bar.



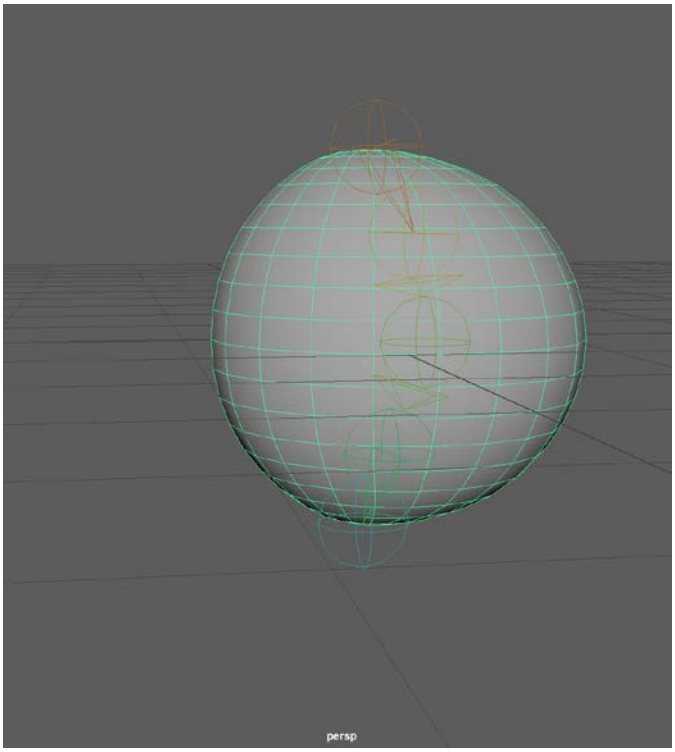
Step 2

Load or create a skeletal mesh model into Maya.

You will need a well-behaved, well-formed mesh (*quad modeled* has proved to be the best so far) that has been skinned to a rig with at least four joints.

You might be able to get away with fewer joints, but for testing purposes, having five joints has proven easiest. TressFX 5.0 binds the hair to the bones (influences) and uses a maximum of 16 bones per vertex.

For this tutorial, we are using a basic sphere (with a 'not artistic but useful' texture) that is rigged (and animated).



The mesh will be used as the reference mesh that the exporter requires. The rig needs to be skinned to the mesh before exporting.

The following steps will show you how to create hair in Maya and then export the three files that can be used by Unreal Engine:

- A TFX file, which contains the guide hairs (as spline curves).
- A TFXBONE file, which contains the required bone (influencer) information.
- A TFXMESH file, which contains the collision mesh to use for this TFX/TFXBONE file set. Typically, developers will use a simplified but fairly form hugging mesh that mimics your skeletal mesh. You can use multiple collision meshes (and TFX/TFXBONE sets) for a single model. The Ratboy example uses three collision meshes.

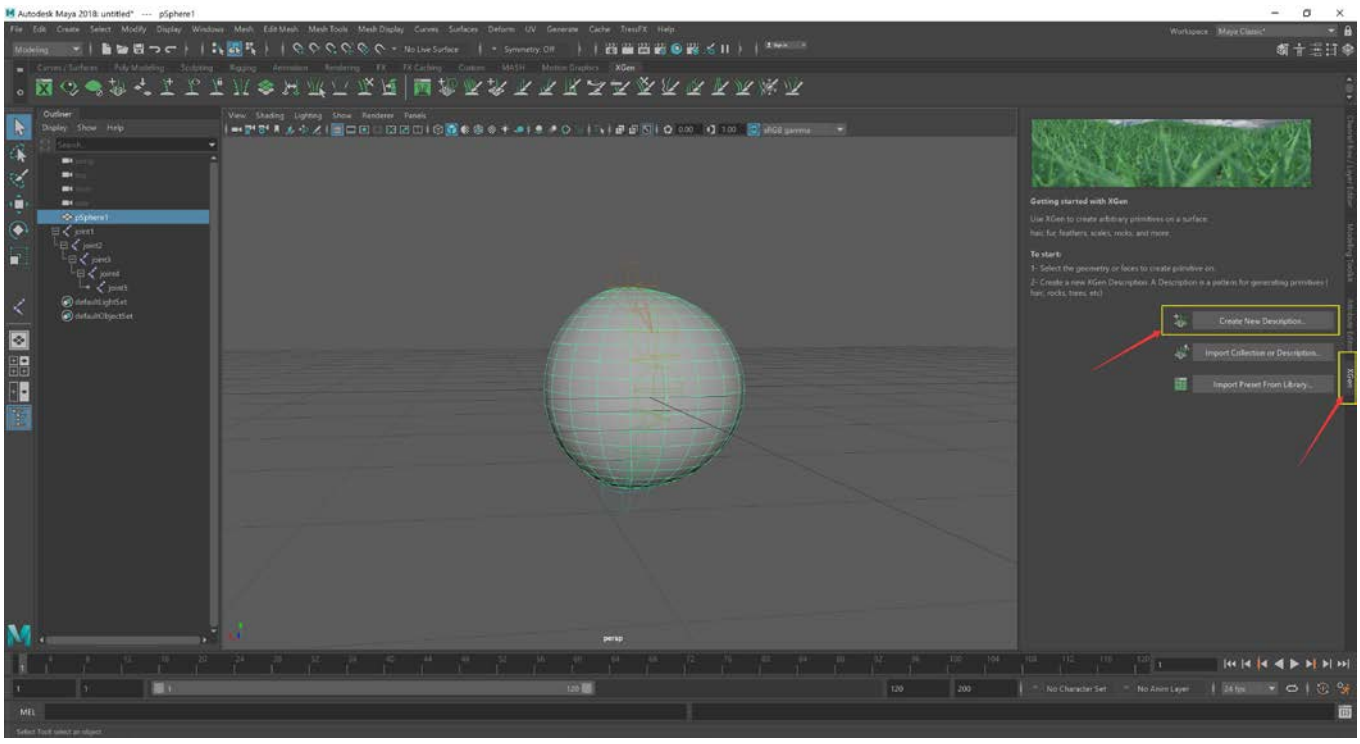
The collision mesh is used to prevent the hairs from penetrating a surface, such as the head or body mesh. Having a basic collision mesh with detailed collision meshes for smaller parts of the model, such as the hands, is a normal use of collision meshes.

Note: See the tutorial on hooking up collision meshes and SDF fields to hair assets for more information on collision/SDF/hair asset interactions.

Step 3

Under XGen settings tab on the main display (typically on the far right), click and bring up the XGen settings window

If you haven't got the ChannelBox/Attribute/... Panel open, you will need to open the panel to see XGen.

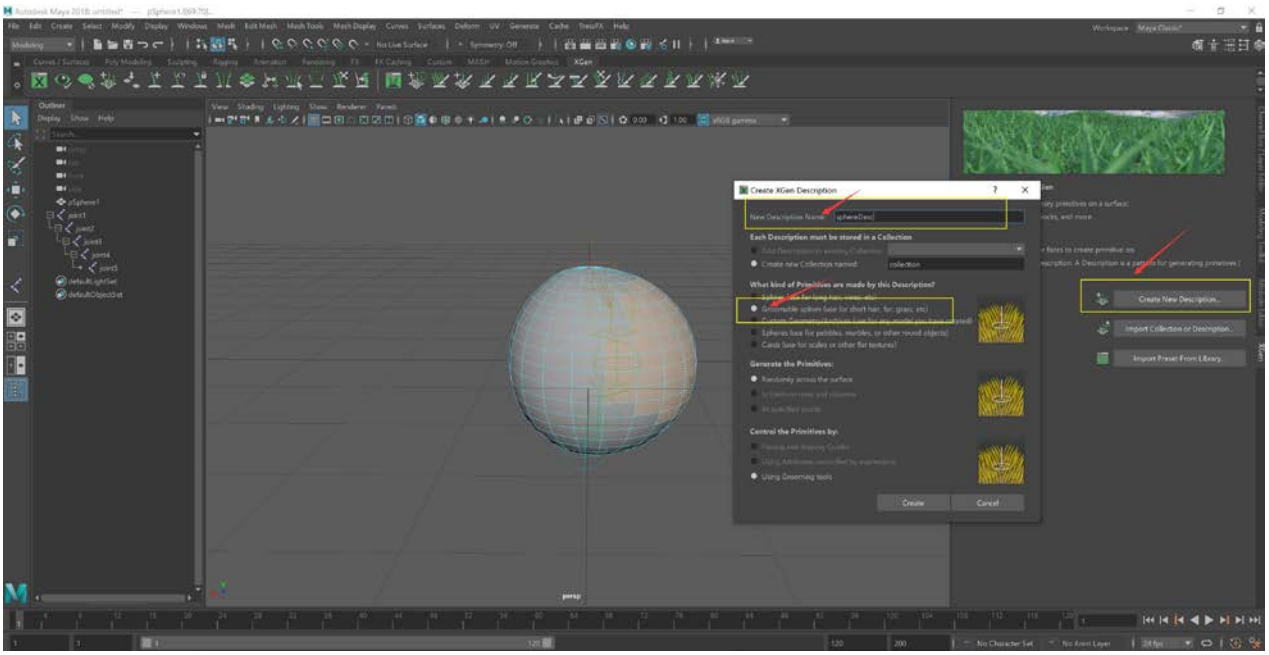


Step 4

Put the Mesh in the Bind Pose.

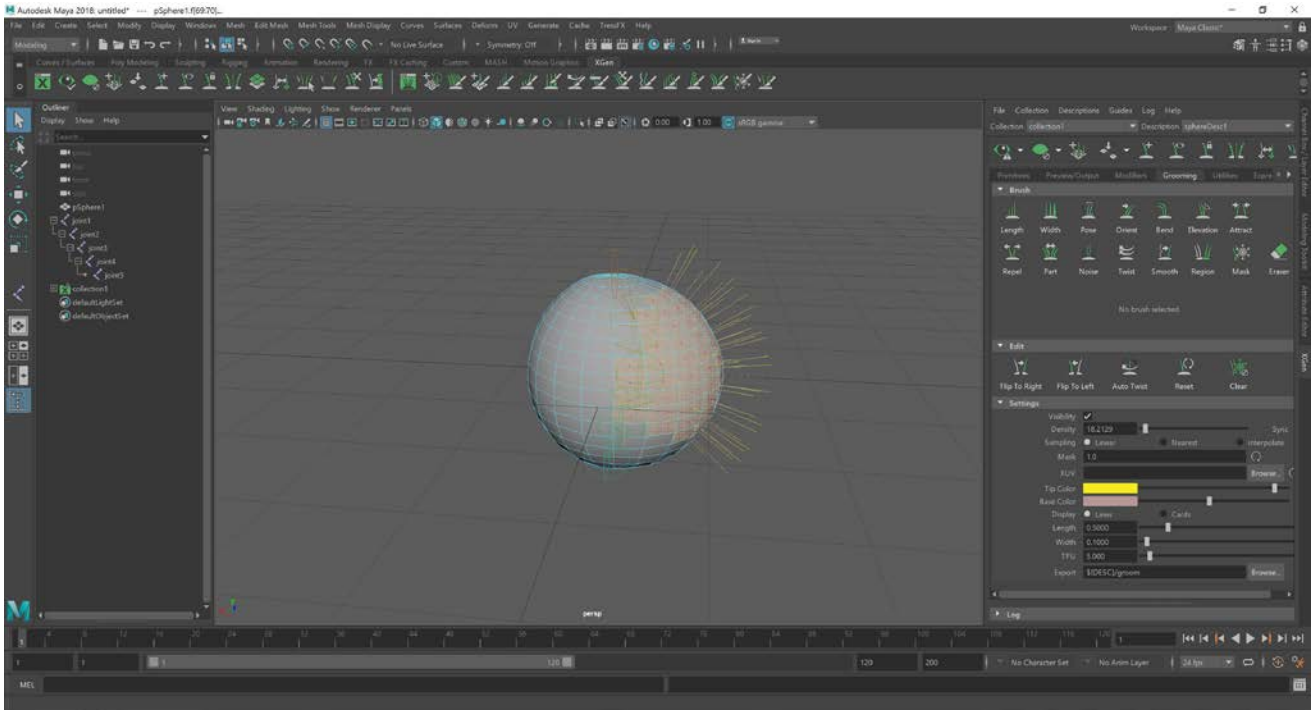
TIP: Try to make the Bind Pose as close to location 0,0,0 as possible. This is to avoid any possible conversion errors that may result in animation offset errors.

Select the entire mesh or a subset of faces of the mesh then create a new description. This can either be done using the button in the Panel or the button in the row of buttons under the XGen tab.

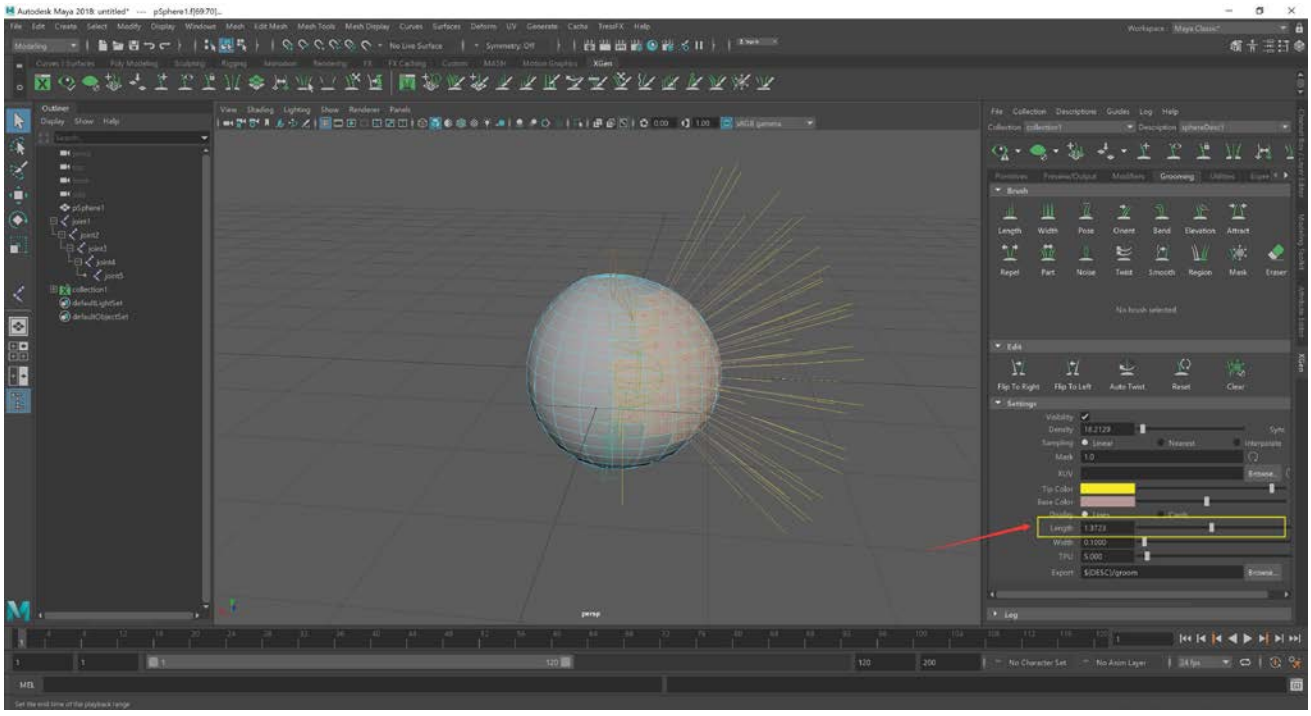


Next, name the description and the collection which holds multiple descriptions. For example, *sphereSection* and *sphereHair* respectively.

You can choose either Splines or Groomable Splines. Splines need to be placed individually, so for this tutorial, we are doing the quicker (and slightly more complicated) groomable splines. Choose **Groomable Splines.**



You can see tiny guide hairs randomly distributed across the selected faces.



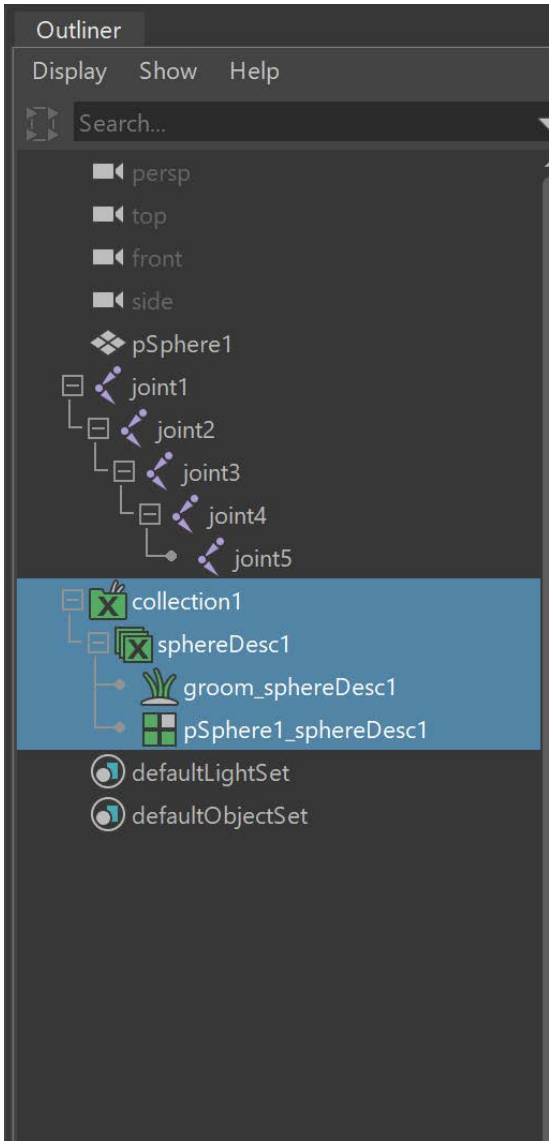
You can change the **Length** setting in the Settings area of the Grooming Tab to something more dramatic, like 10 or 20. You will need to type it in as the slider has a limited range based on the current value.

You can use the grooming tools to shape the hair. We do not do this here, but feel free to play before continuing.

Note: You cannot edit the grooming splines directly via control vertices. See the next step.

Step 5

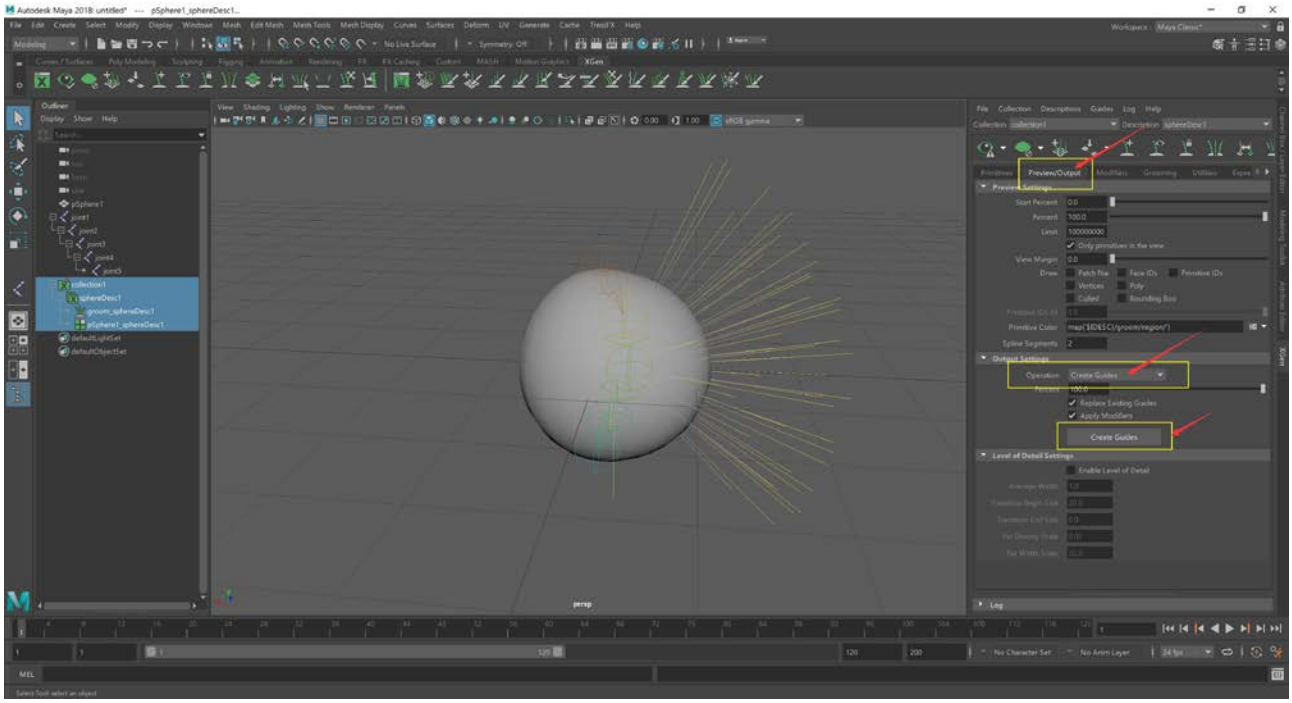
Now we are going to convert those groomable splines to actual curves.



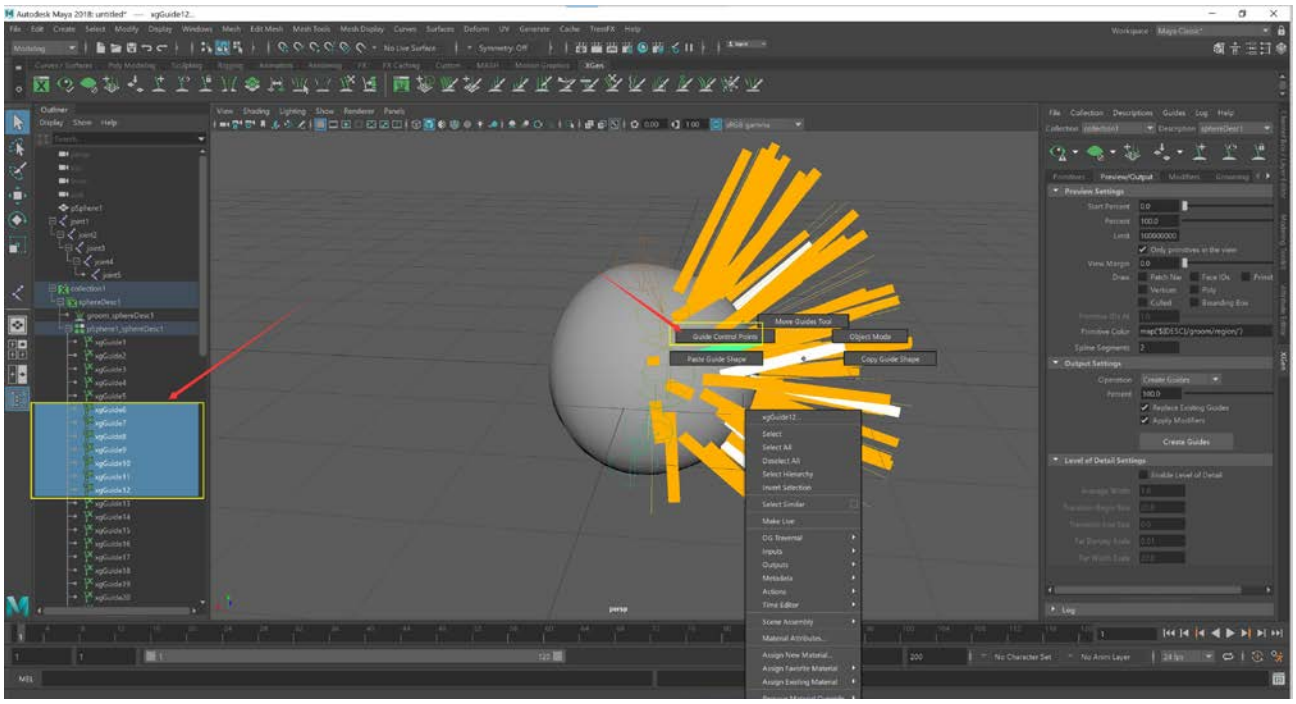
If you look in the Outliner tab under the collection you created, you will notice that you only have descriptive information, but not any actual splines. That is why you cannot edit these groomable splines any direct way but must use the grooming tools.

Since we require actual splines in order to export, we will convert these groomable splines into individual splines:

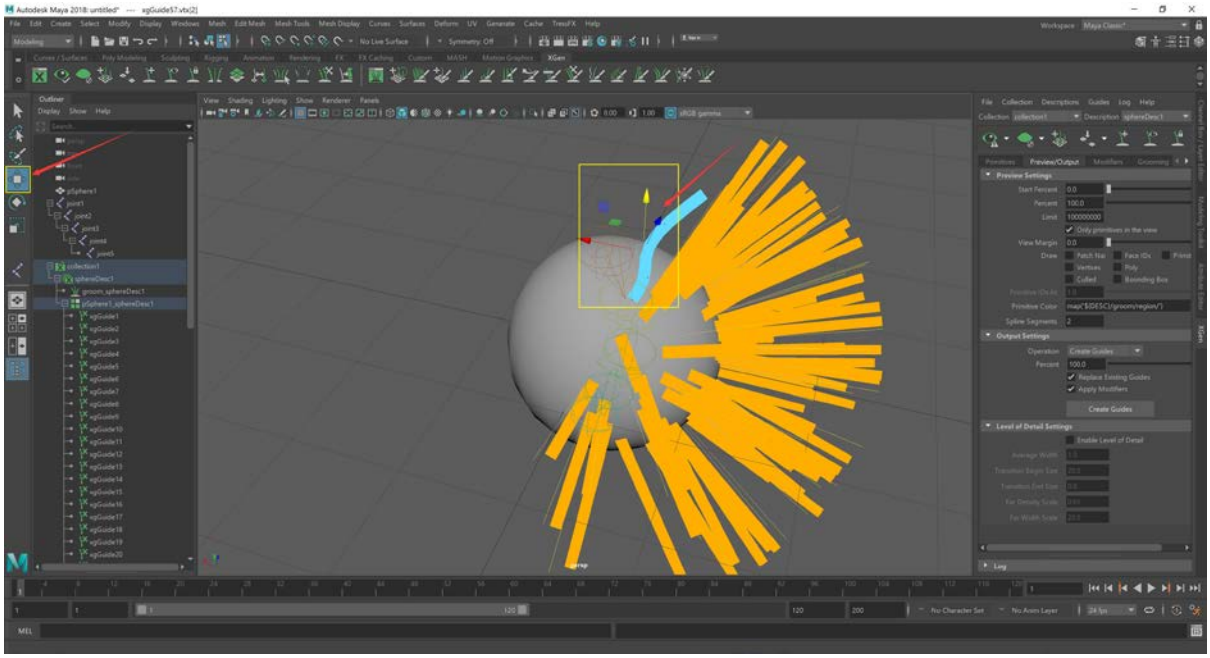
1. Go to the **Preview/Output** tab in the XGen Panel.
2. Go down to the *Output Settings* section.
3. Click the dropdown for **Operation**.
4. Change it from the default *Render* to *Create Guides*.
5. Click the **Create Guides** button that appears.



Guides that you can directly manipulate are now created. These guides can be seen in the Outliner.



You can manipulate these guides by right-clicking on them, holding for the markup menu, and then selecting *Guide ControlPoints*.



You can then select and move the control points with the standard transformation tools.

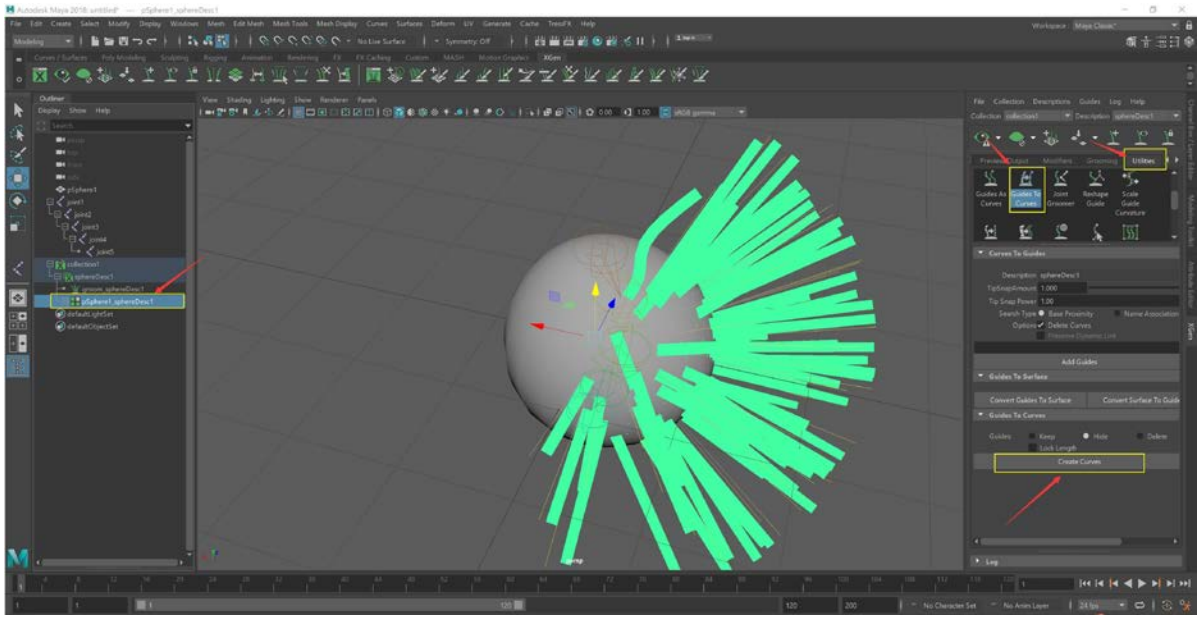
Step 6

Now we need to convert these guides into curves so we can export them with the TressFX Exporter.

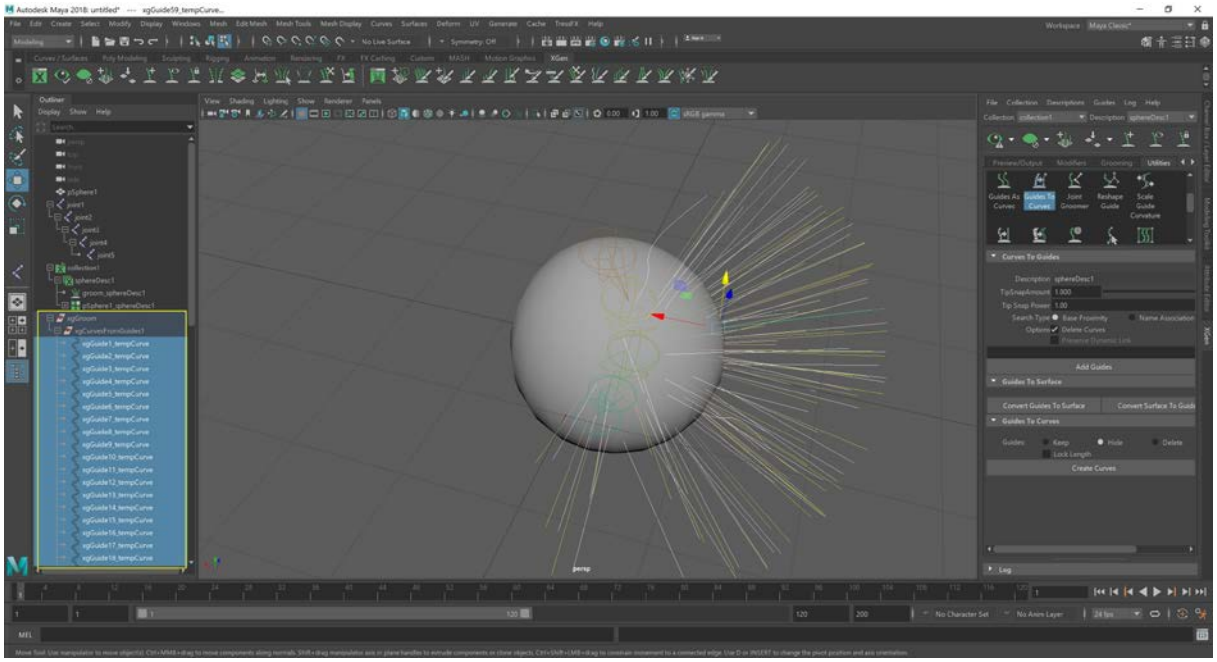
Go to the **Utilities** tab in the XGen Panel.

Click the **Guides to Curves** tool. This makes it available for use in the area below the tools list. You should the section called *Guides To Curves* appear. Make sure it is open, so you can see the options and button.

In the Outliner, press and hold Shift and select all the individual guides. You should select all the guides you want to convert.



Click the **Create Curves** button in the *Guides To Curves* subsection of the XGen Panel.



The guides should be converted to curves now.

The curves will be in their own group in the Outliner, under the default name of **xgGroom**.

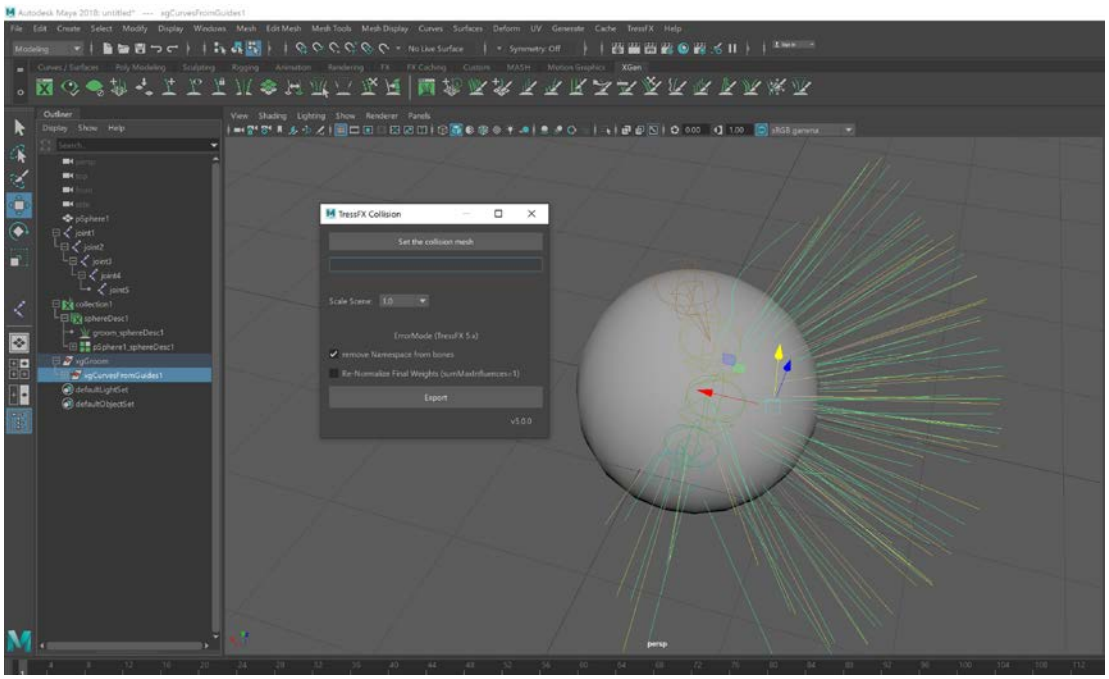
Step 7

Now it is time to start exporting using the TressFX Exporter. We'll start with the collision mesh.

Open the menu item: **TressFX →Export Collision Mesh**

In the Outliner, click on the sphere mesh (not the joint hierarchy).

Note: Make sure the joint hierarchy is skinned to the mesh, though!



In the TressFX Collision dialog, click the **Set the collision mesh** button. You should see the shape of the mesh now set.

Go ahead and Export. You will be asked for a name and a file location. It is best to keep all these files together, so we recommend having a folder that contains the Maya file.

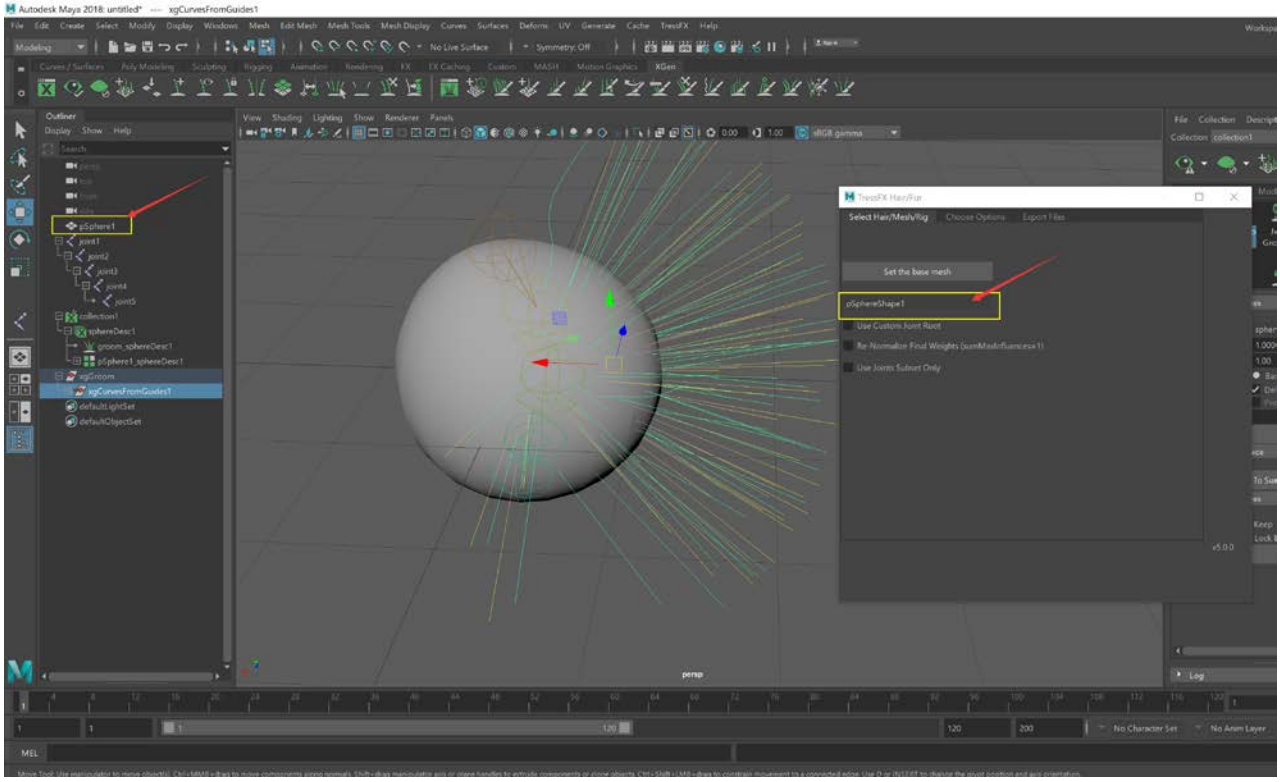
The TFXMESH file will now have been exported, so you can close the TressFX Collision dialog.

Step 8

Next, we want to export the hair and bone data.

Open the menu item: **TressFX→Export Hair/Fur**

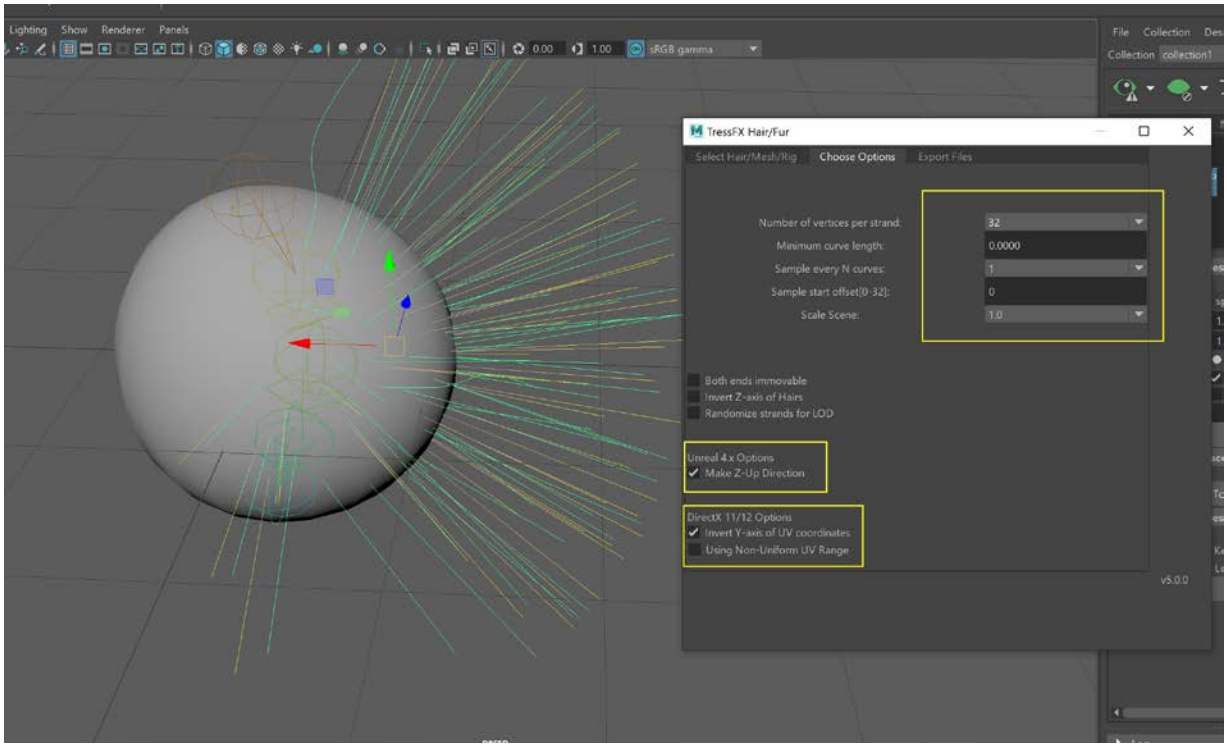
Again in the Outliner, click the sphere mesh. With the mesh selected click **Set the base mesh** in the TressFX Hair/Fur dialog.



In the TressFX Hair/Fur dialog keep most of the defaults in all the three tabs. For this example, set the **Number of vertices per strand** to 32 (in the Options tab) and leave the **Minimum curve length** at 0.0000.

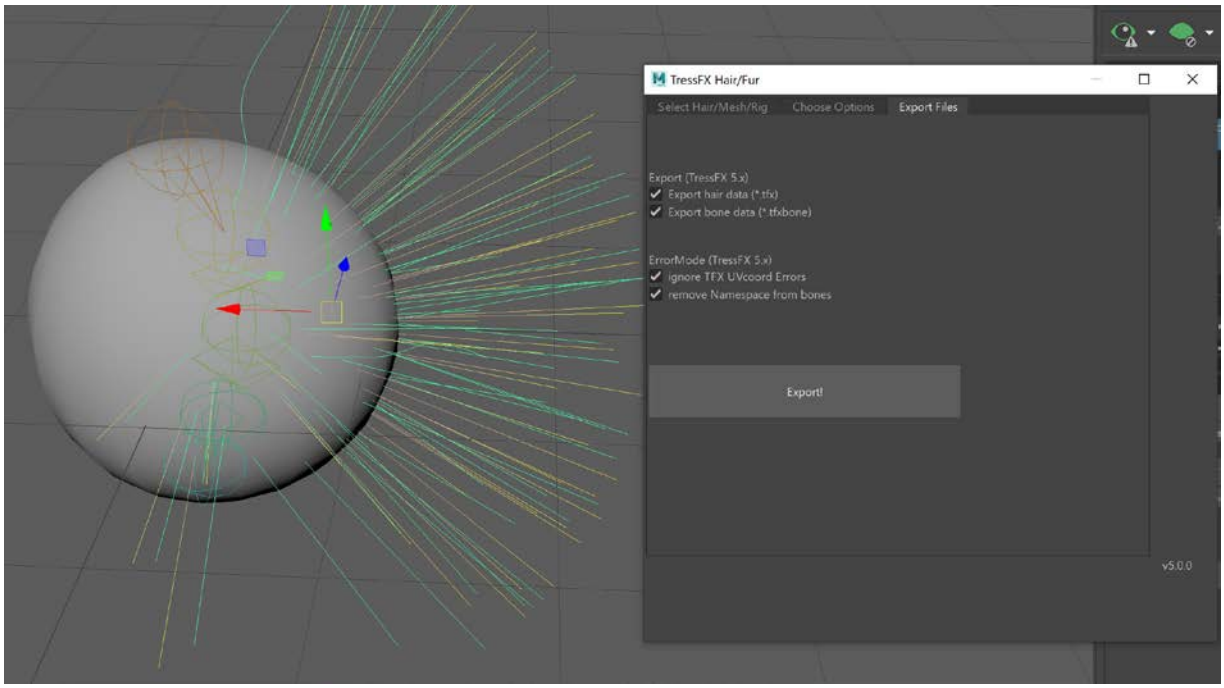
For Unreal Engine, make sure the option to have Z as the UP axis is selected, as Unreal Engine requires this. This should be the default setting, so for other engines that use Y as the UP axis, uncheck this box.

If exporting to Windows® (Cauldron and Unreal Engine on Windows), make sure the DirectX® options are selected as well.



Click the checkbox **Export bone data (*.tfxbone)**. This is the third tab in the dialog.

Both **Export hair data (*.tfx)** and **Export bone data (*.tfxbone)** should be selected, as we are exporting both.



Note: If you need/want to export more hair groupings on the same mesh (with the same bound rig), for now, you will need to re-export the bone data. This is because the current Unreal Engine TressFX build looks for a TFXBONE file that has the same name as the specified TFX file. You can just copy the TFXBONE file and rename it, but just keep in mind the current naming requirements for a TFX/TFXBONE set.

Important: You need to select the curves that you want to export. You are selecting the curves that have been **converted** from grooming splines into NURBS curves (splines). In other words, the curves that are under **xgGroom**.

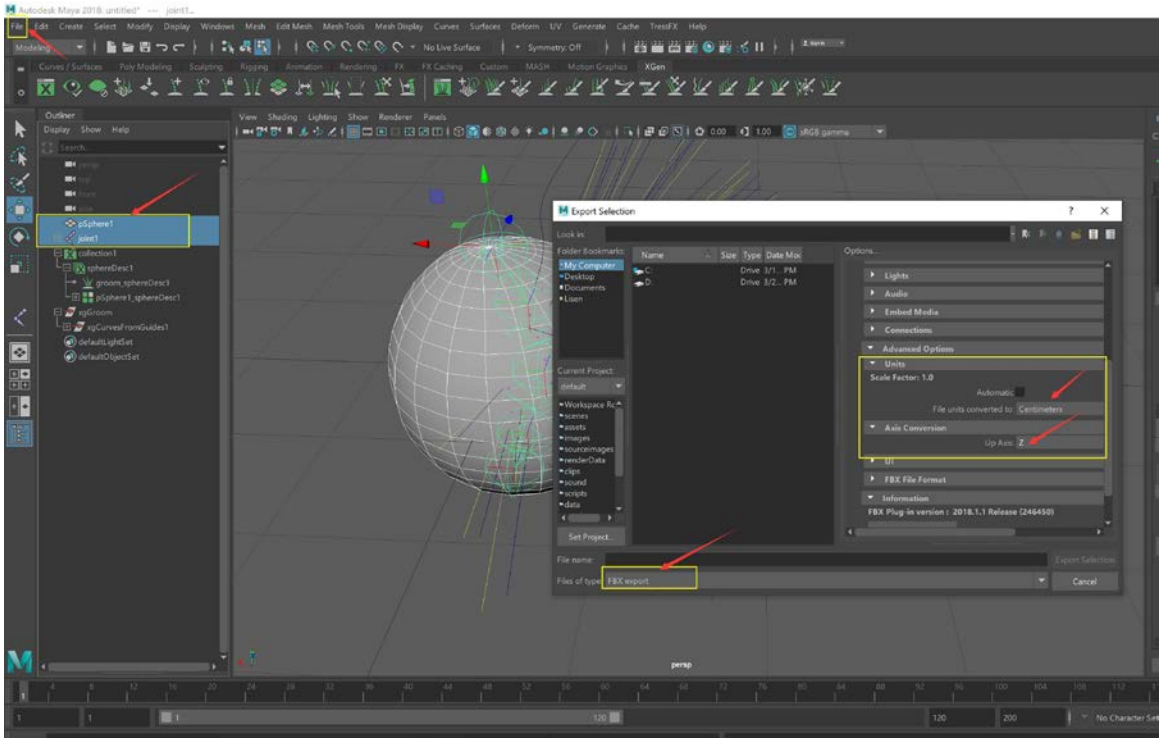
Note: You should be able to just select the hair group (or groups). If that does not seem to get all the hair (when importing and inspecting), then try shift-selecting all the individual splines. The exporter is designed to do a recursive search for splines (nurb curves), but it relies on the OpenMaya API.

After you've selected the curves, and set these options in the dialog, go ahead and click **Export!** Again, you will be asked for a filename and location for both the TFX and the TFXBONE files.

At this point you should have exported the TFX (.tfx), TFXBONE (.tfxbone) and TFXMESH Collision (.tfxmesh) files.

Step 9

Finally, you need to export the mesh and rig as an FBX file.
This should be straightforward, but if you're using Unreal Engine make sure that you've set the units value to 1 unit = 1 centimeter and make sure that the Bind Pose is in the frame 0 slot of your animation.



Once the FBX file has been exported, you are now ready to import your FBX and TressFX files into either Cauldron or Unreal Engine, and to create a TressFX aware skeletal mesh.

See the quick tutorial on how to create a TressFX aware skeletal mesh in related links.

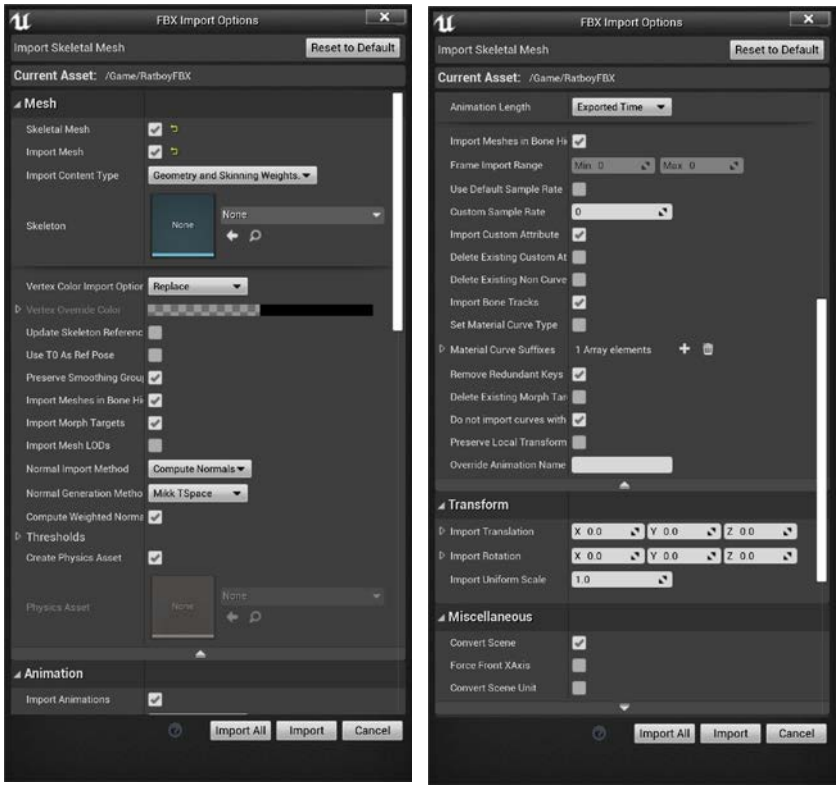
Note: Tutorial results may vary and there may be other, easier ways to make sure the Bind Pose is exported out of Maya and into Unreal Engine. The exporter wants you to model the hair in Bind Pose. If this is not done and the Bind Pose is not imported as the skeletal mesh in Unreal Engine, then this will lead to problems with hair offsetting. This can be very dramatic errors in some cases. The current rule is to use the Bind Pose, make the Bind Pose as close to 0,0,0 as you can in Maya, and then export the Bind Pose to Unreal Engine. If you make it frame 0, you can easily set an option during import to take T0 as the Bind Pose in Unreal Engine.

Using UE4 TressFX 5.0 Components and Materials

Importing UE4 TressFX 5.0 Assets

Importing the FBX Asset

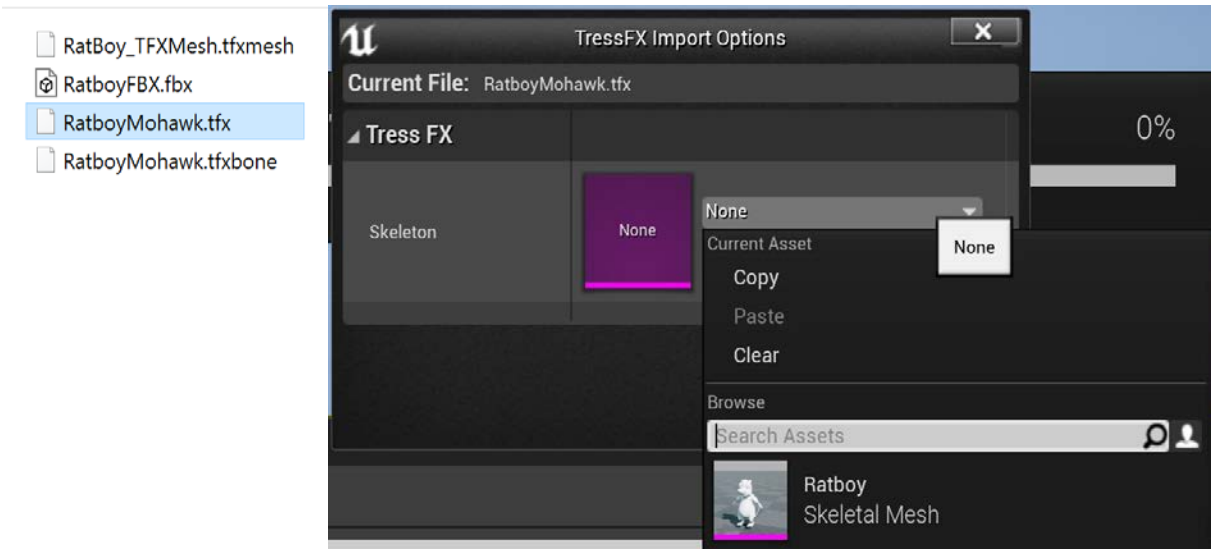
As an example, here we import our Ratboy Asset: RatboyFBX.fbx.



Importing the tfx/tfxbone Assets

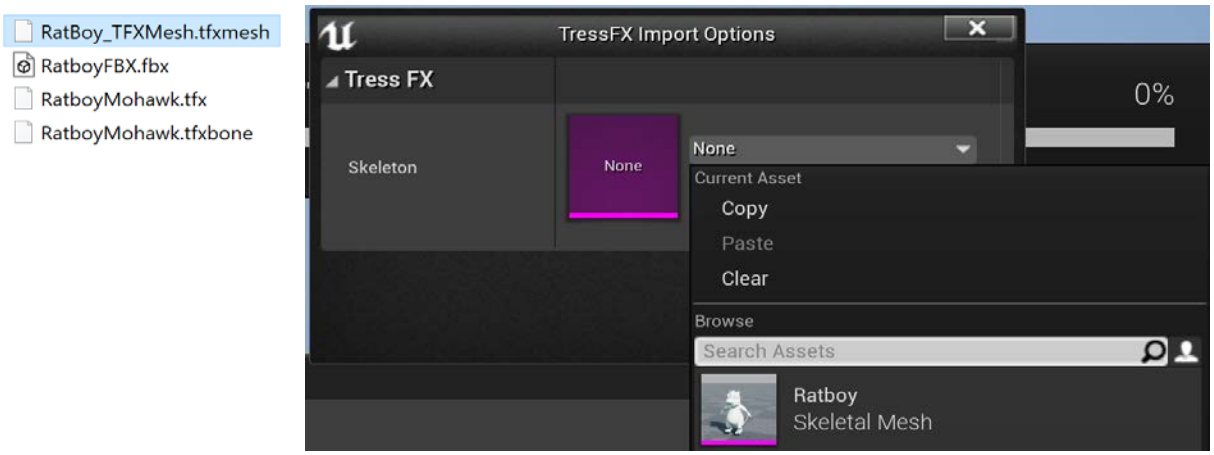
You only need to import the tfx file which contains the data of strands. tfxbone file will be imported automatically and should have the same name as tfx file.

To import the tfx file, select the corresponding Skeleton in **TressFX Import Options** window.



Importing the tfxmesh Asset

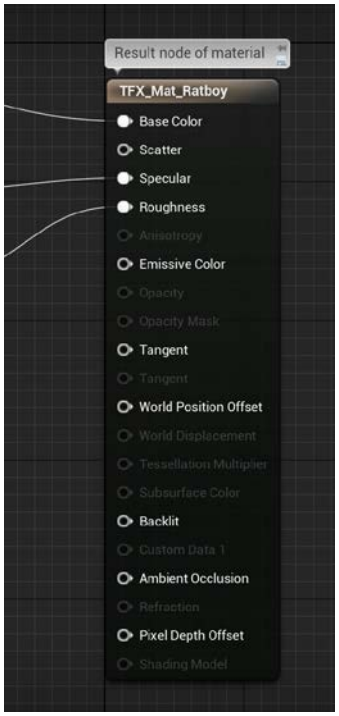
The tfxmesh file contains the collision mesh of Ratboy. To import this asset, select the corresponding Skeleton in the TressFX Import Options window.



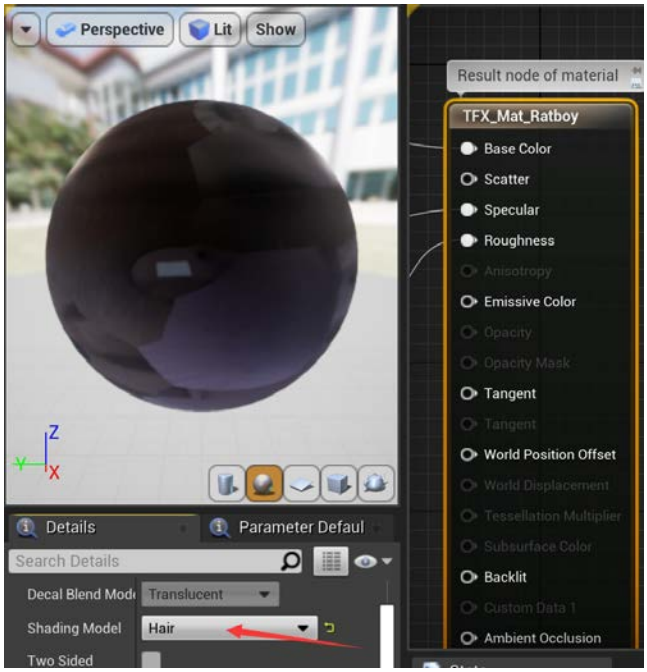
Creating UE4 TressFX 5.0 Material

1. Create and open the material.
2. Name the TressFX material with the prefix: TFX_Mat_* (recommended).

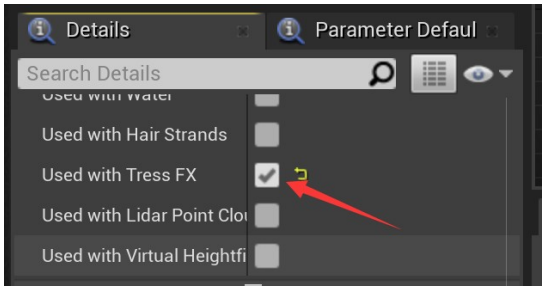
3. Select *Result node of material*.



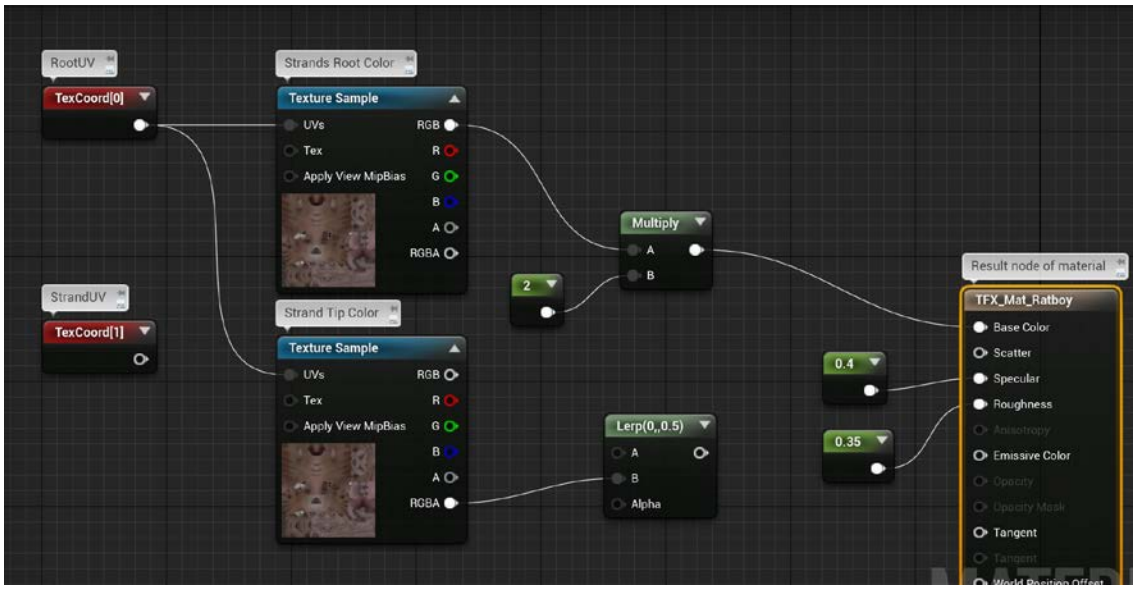
4. Set *Shading Model* to *Hair*.



5. Enable *Used with TressFX*.

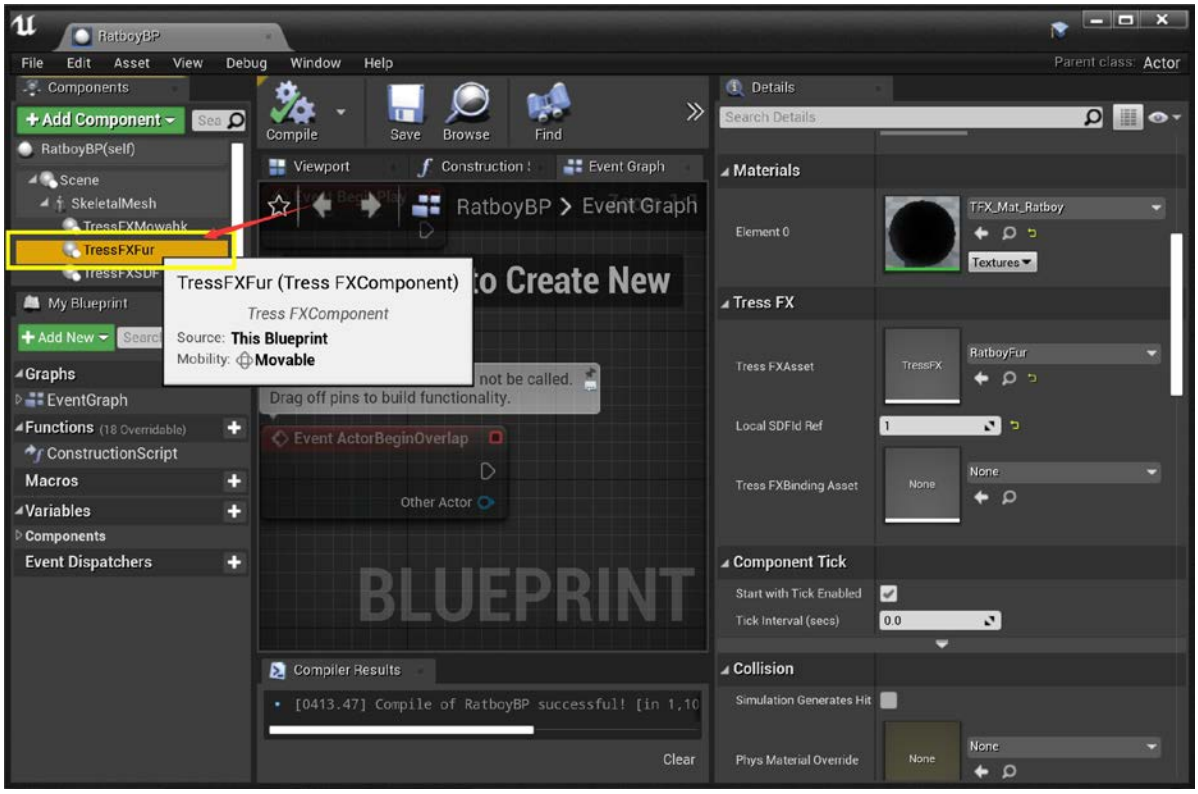


6. Use these UVs(TexCoord[0] - RootUV, TexCoord[1] - StrandUV) to generate a more interesting basecolor.

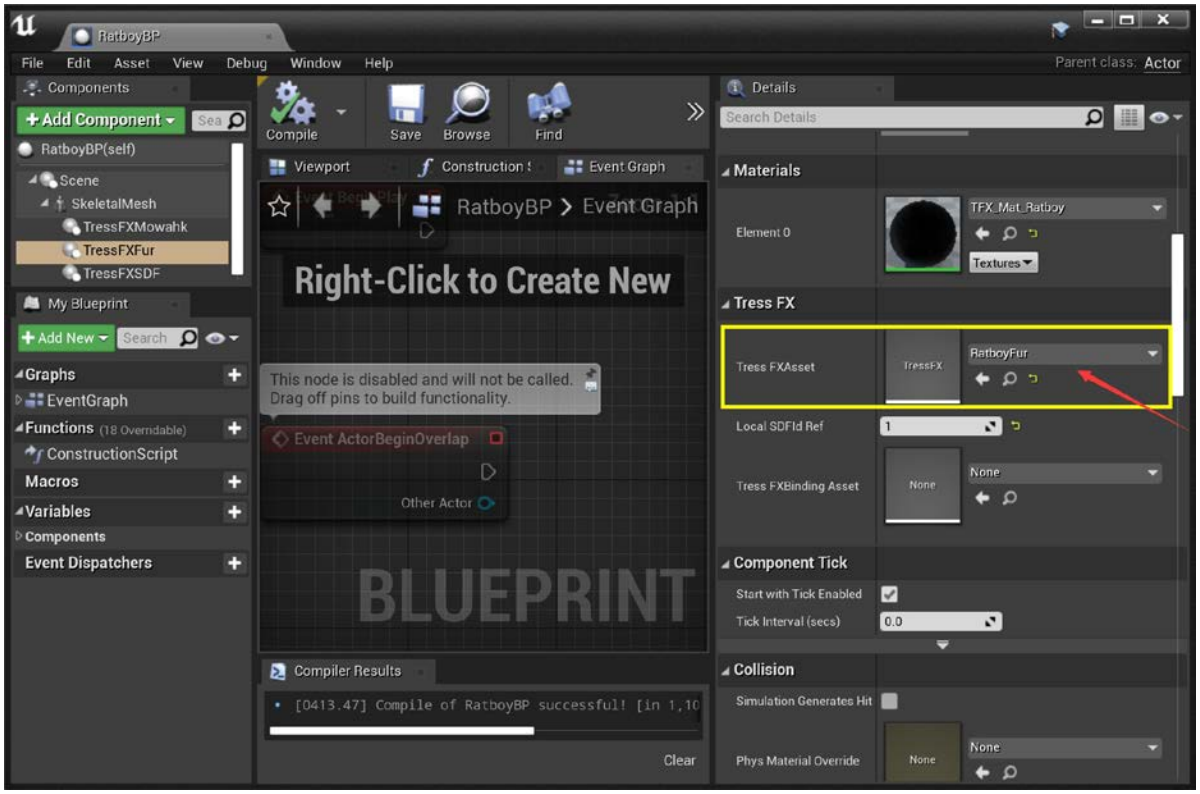


Creating UE4 TressFX 5.0 Blueprint Actor

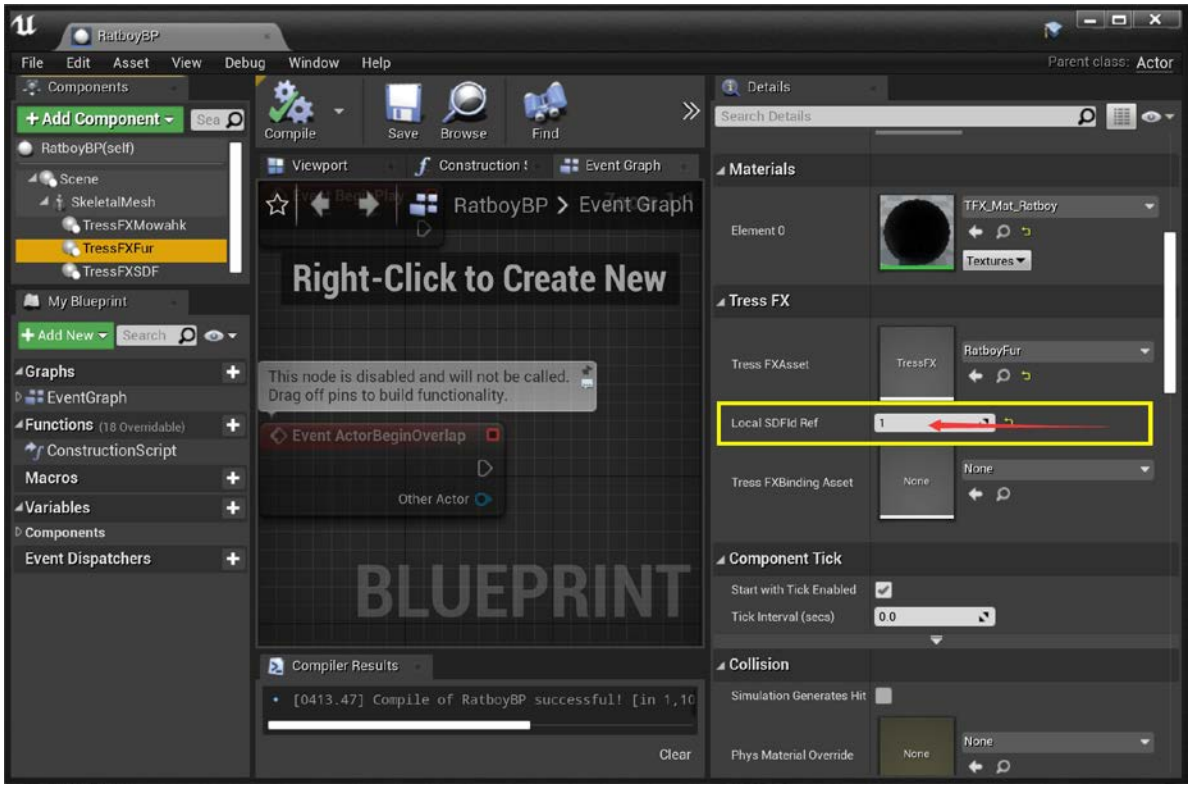
1. Attach the TressFXComponent to the SkeletalMesh.



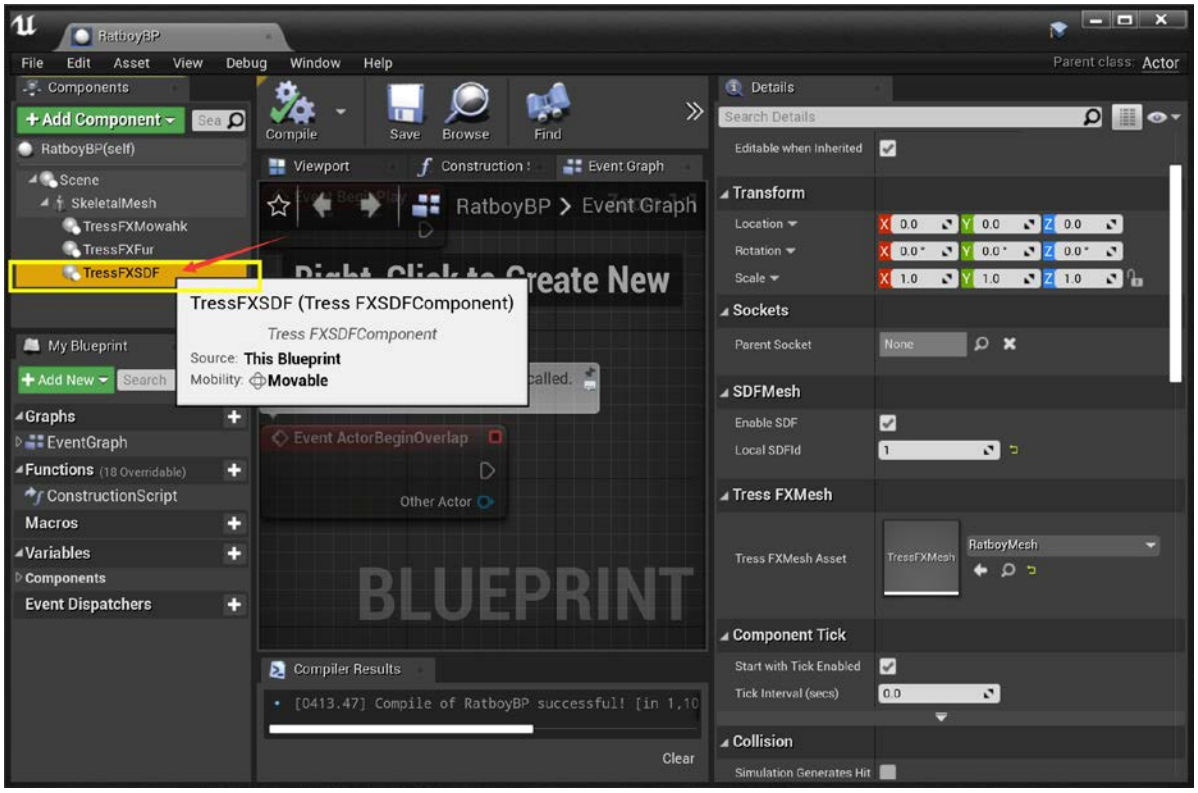
2. Set TressFXAsset.



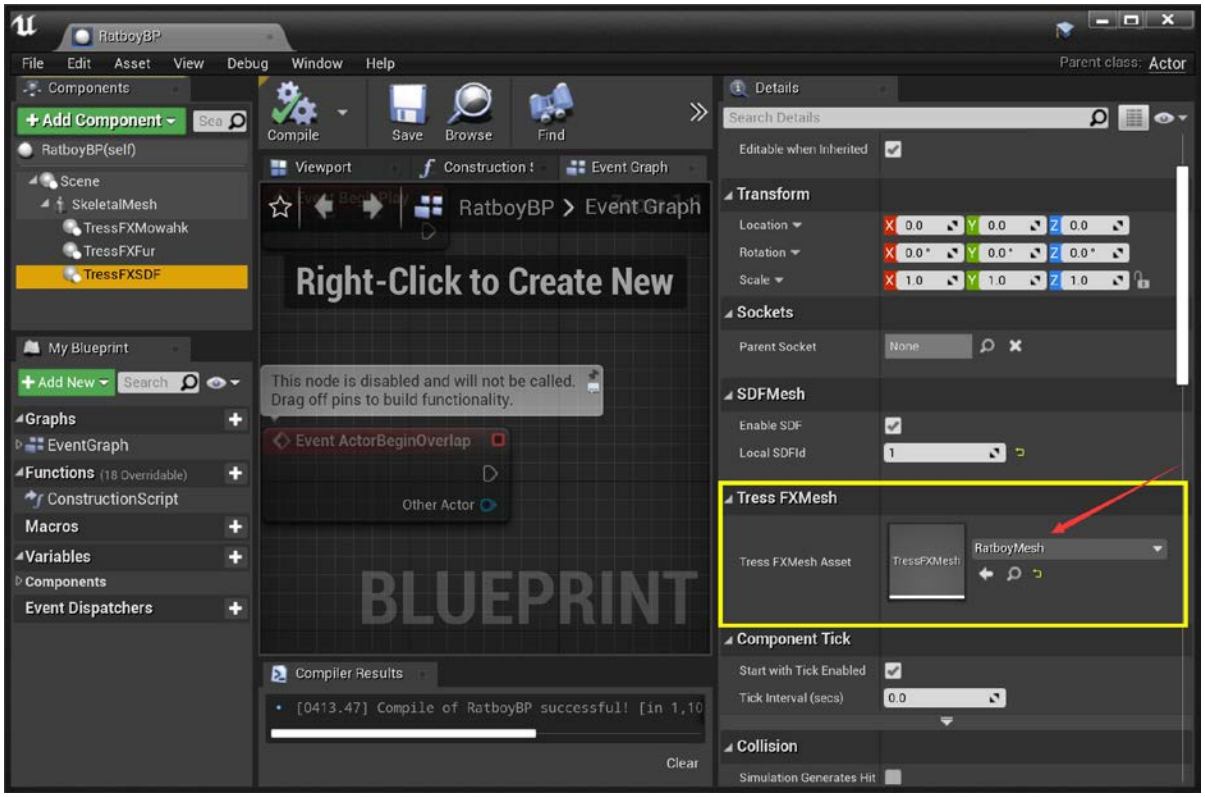
3. Set LocalSDFIdRef to enable SDF collision feature.



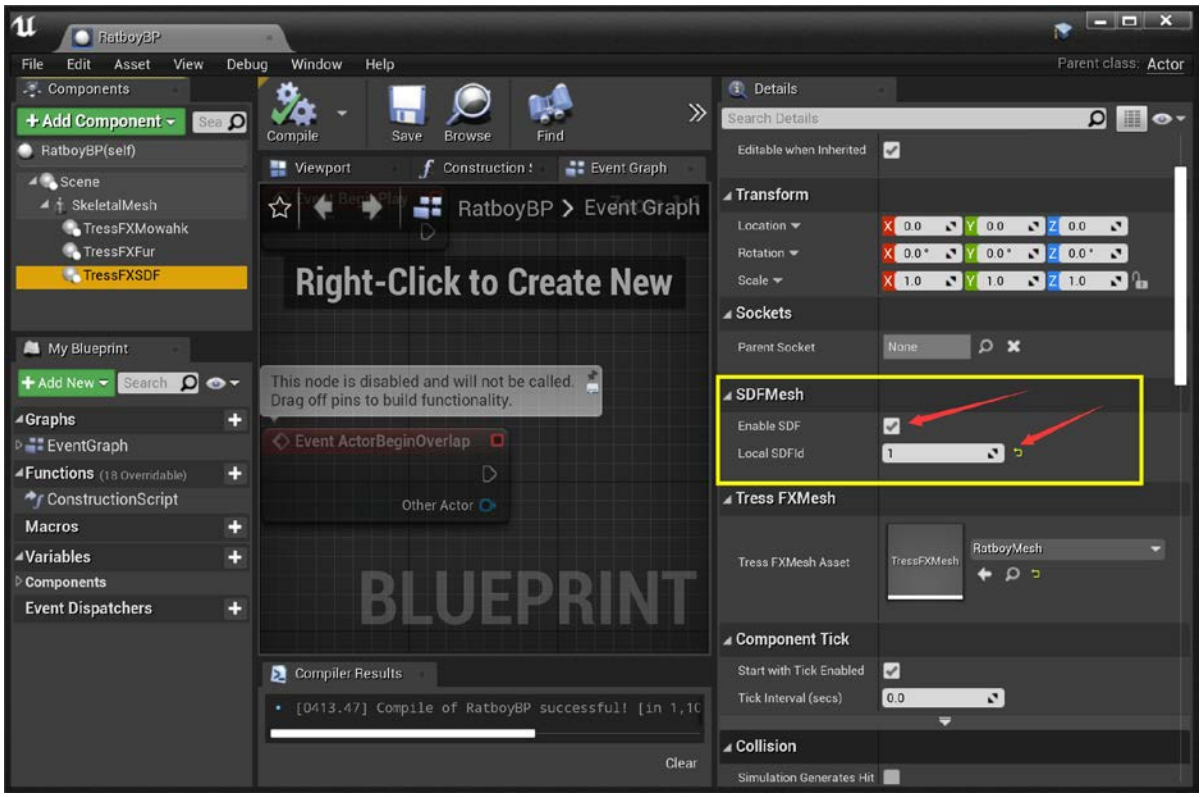
4. Attach the TressFXSDFComponent to the SkeletalMesh.



5. Set TressFXMeshAsset.

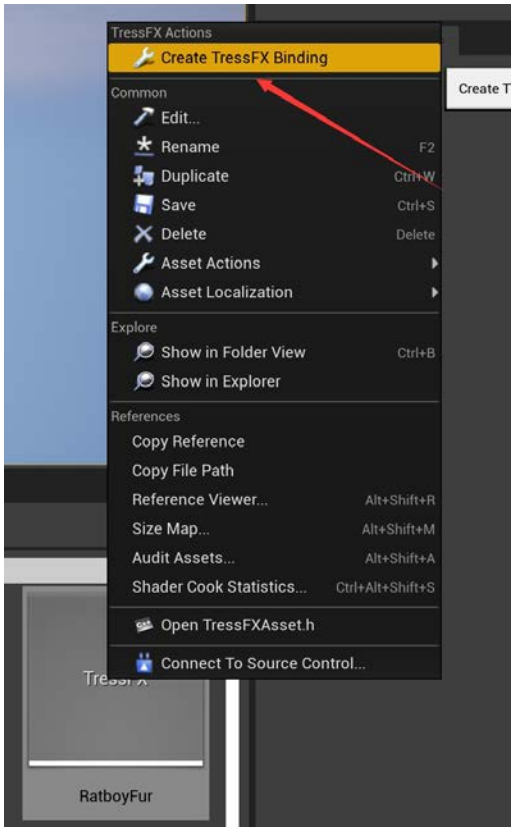


6. Set EnableSDF and LocalSDFId.

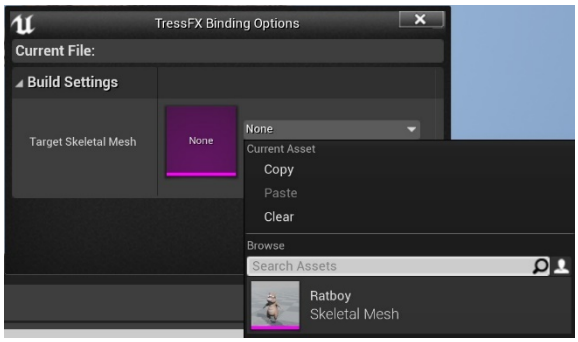


Using UE4 TressFX 5.0 Triangle based Skinning

1. Select TressFXAsset and right-click to create a TressFXBindingAsset.

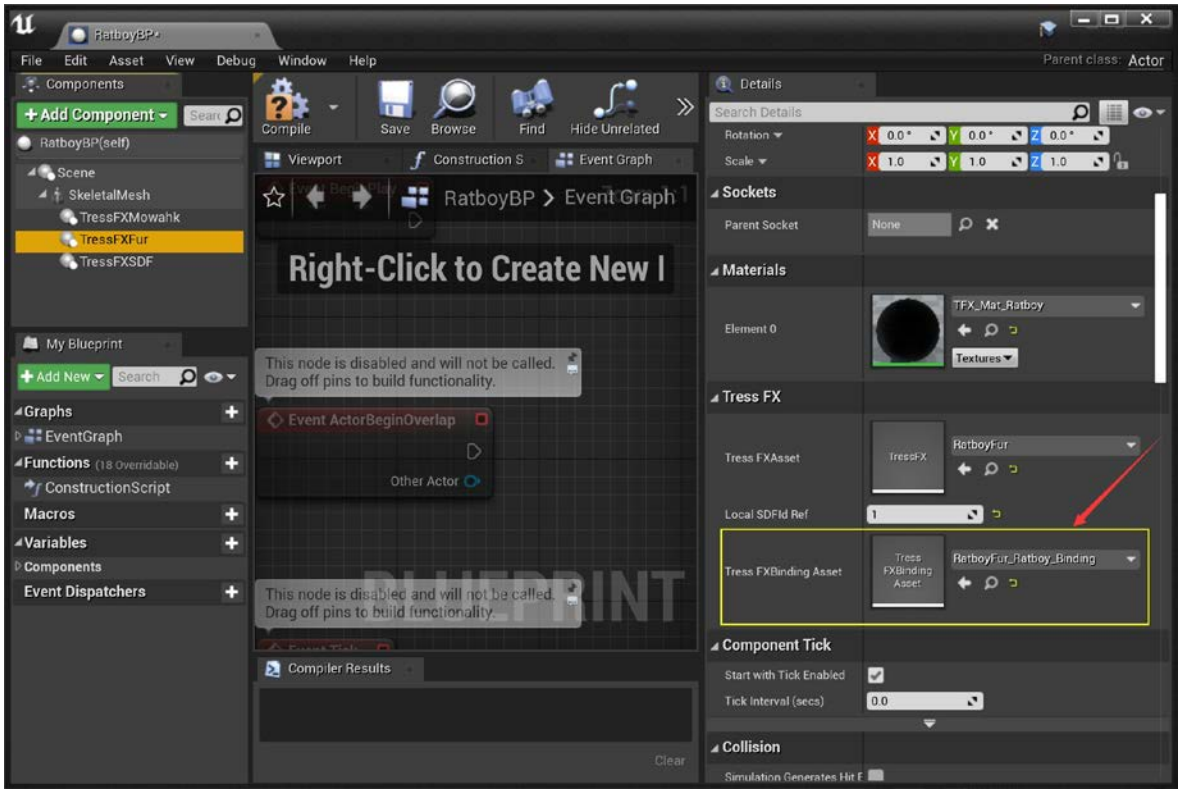


2. Select the right SkeletalMesh.



3. Generate the *_Binding Asset.

4. Set the *_Binding Asset to the corresponding TressFXComponent.



DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD’s products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

AMD, the AMD Arrow logo, AMD Radeon, and combinations thereof are trademarks of Advanced Micro Devices, Inc.

Microsoft, Windows, DirectX, and combinations thereof are either trademarks or registered trademarks of Microsoft Corporation in the US and/or other countries. Vulkan and the Vulkan logo are registered trademarks of the Khronos Group Inc.

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.