Three.js model/animation workflow

# From Maya to Three.js

Bringing an animated and textured mesh from Maya into Three.js is a cumbersome and annoying system. But it can be done. I’ve spend the better part of 2 weeks, but I`ve finally solidified the workflow required to make this happen.

First, some caveats. This tutorial is working under the r63 update and assumes you are as well. Things change quickly and rapidly in the three.js world. What I say here may or may not work in your version of three.js

Firstly, the native maya exporter for three.js that ships with the product is useless. Don`t even attempt to use it. The workflow needed to get this working is going to involve creating things in Maya, and then shifting over to Blender for the exporting. If you are OK with working 3D in Blender, then it would be easier for you to skip Maya altogether and use Blender from the get-go. However, for those of you attached to Maya, such as myself, and find it way too frustrating to make the jump to the clunky workflow that Blender offers, here is what needs to be done to bring an object from Maya to blender, then into three.js.

The Idea

We will be using the JSON loader that comes with three.js to do the majority of this work. The JSON format offers a simple and heavily modifiable system to bring meshes into three.js and is, in my opinion, currently the strongest way to bring meshes into a scene. It`s potentially possible to build a script for Maya to do this conversion for you, but that was much too annoying for me to do. Considering how rapidly three.js changes, it will just get broken with time and no upkeep, so I have not done so. Maybe in the future I will, but for now, we will be forced to use the much more up-to-date blender exporter.

What this tutorial goes through

* Getting MORPH ANIMATIONS into three.js (NOT skeletal animations – those were not attempted)
* Getting multiple objects into one mesh with different materials.

In Maya

Begin by modelling, animating, and UV’ing your scene in Maya. Do NOT, I repeat, do NOT put textures or materials on your objects. Materials are the only thing that cannot be transferred over to Blender, so you are going to have to do that part separately in blender(sucks, I know, but it’s not that bad and at least you don’t have to learn ALL of blender).

Once you are done modelling, animating and UV’ing your scene in Maya, you are going to export it to a special format called COLLADA. COLLADA is a cross-product format for holding meshes and animations. The latest version of openCOLLADA can be found on their github at:

<https://github.com/KhronosGroup/OpenCOLLADA/wiki/OpenCOLLADA-Tools>

Download and install their Maya-plugin(Might need a restart to be able to export). Now, you should be able to export the COLLADA format which will be listed in Maya as DAE\_FBX. Choose what you want, hit ‘Export Selection’ or ‘Export All’(for everything in the scene), and export it as a DAE\_FBX.

That’s the Maya part over and done with. We now have to move on to the Blender part.

In Blender

Now go download and install Blender. Grab the blender exporter that comes with blender and install it. This is located in utils\exporters\blender; in the source code that comes with three.js. Read the readme that comes with it to figure out how to install the exporter into blender.

Now, open Blender, go to file->import collada, and import the exported DAE from Maya. Make SURE to click the “import units” option when importing to get the proper scaling.

**Special Note:** If you had materials on your meshes in Maya, Blender will complain at you telling you that the files you had in the textures in your Maya file don’t exist. Note that this complaint comes in the system window, there’s actually no feedback in blender itself (shocking). In blender it will just look like nothing happened when you tried importing. To fix this, you have to complexly remove textures in the exported DAE file. Open it up in your favourite text editor and locate any XML tag for <texture>. They normally look like this:

<texture texture="file1-image" texcoord="CHANNEL0">

<extra>

<technique profile="MAYA">

<wrapU sid="wrapU0">TRUE</wrapU>

<wrapV sid="wrapV0">TRUE</wrapV>

<blend\_mode>NONE</blend\_mode>

</technique>

</extra>

</texture>

Get rid of all references that look like that in the DAE file.

Alternatively you can just remove all materials in Maya itself and re-export it. Both ways work.

If you want materials to show up, you are going to have to make them in Blender OR make them directly in three.js. In many ways, it’s actually much easier to make the materials in three.js, not Blender. But to each his own. Here’s a good way to decide if you want to make materials in Blender or Three.js:

**Blender?**

* If you have a lot of tiny meshes that need their own BASIC texture/colour with NO tiling of the texture, only a simple lambert(or other basic shader), you can use Blender.

**Three.js?**

* If you need your texture tiled at all, don’t even think of doing that in Blender, it needs to be done in three.js.
* If you want normals, bumps, specs, whatnot, do it in three.js it will save you a world of trouble.

Creating materials in Blender is a fairly annoying and tedious process that I haven’t quite got the hang of myself. Once I get a better understanding of it, I’ll update this post right here with the information. For now I only know how to make a lambert with the same texture that I was using when creating UV’s in Maya. So I can get the same basic flat texture.

🡪Information on how to make materials in blender will come here 🡨

Once you’ve done whatever material work you can muster in Blender(or skipped and left for three.js), it’s finally time to get your scene into three.js. In Blender, got to file->export->three.js. Now, there are some options to adhere to:

* **Flip Y/Z:** Hit this button so you get your mesh proper side up in three.js
* **Morph animation:** IF, and I stress IF your mesh is animated, hit this button. If it’s not animated, leave this button unchecked. When you export from Maya it WILL export the entire timeline to morph targets in the DAE, so a static mesh will get a bunch of useless animations that take up a lot of space in the resulting three.js file.

The rest can be left default. You can tinkle with these options if you want to see the various properties.

Now, we can finally head into three.js

In Three.js

Before loading the JSON file, open it up and take a look at it. There are some things to note. If you had more than one object in the file, there will be a separate material for each object that was in the scene. I tell you later how you can use this to your advantage. Take a look at one of the material sections of the JSON file:

"DbgColor" : 15597568,

"DbgIndex" : 1,

"DbgName" : "lambert1.003",

"blending" : "NormalBlending",

"colorAmbient" : [0.3446144155355455, 0.3446144155355455, 0.3446144155355455],

"colorDiffuse" : [0.3446144155355455, 0.3446144155355455, 0.3446144155355455],

"colorSpecular" : [0.5, 0.5, 0.5],

"depthTest" : true,

"depthWrite" : true,

"mapLight" : "brick.png",

"mapLightWrap" : ["repeat", "repeat"],

"shading" : "Lambert",

"specularCoef" : 50,

"transparency" : 1.0,

"transparent" : false,

"vertexColors" : false

The exporter is not always perfect, but the beauty of this system is the ability to manipulate this JSON file after the fact. Here are some things to note:

* Take a look at what texture file it thinks should be linked with the object. It states “mapLight“ is the what channel this brick file should be for. That is wrong. That brick file was meant for the diffuse channel. So you can go ahead and change “mapLight” to “mapDiffuse” and “mapLightWrap” to “mapDiffuseWrap.”
* Look at the name of this file. It’s pretty useless. You can manually change the name of this file, which will be very useful to us later on.

You can mess with this file as much as you want, there are a lot of neat things you can do from here before you import it directly into three.js.

**var** loader = **new** THREE.JSONLoader( **true** );

loader.callbackProgress = callbackProgress;

loader.loadAjaxJSON(

loader, // loader context itself

"../models/myMesh.js", // Model absolute path

meshloader("flingpiece"), // Finished callback function

“../images”, //Texture Directory

callbackProgress // Progress Callback function

);

The above code is the model for loading a JSON file into three.js. There are a few things to note about this system of loading files:

1. It’s asynchronous. It will fire a task that, when finished, will call meshloader. Meshloader should make sure to do anything that would need to be done after this json is finished loading. More detail on meshloader later.
2. The fourth argument takes a directory, which is the place where the images references in the JSON file should be stored.
3. The last argument is a progress function called during the loading stage to help you create load sequences and track the time for loading the file.

The callbackFinished function

function meshloader(fileName){

return function(geometry, materials){

var image = new Image();

image.onload = function () { texture.needsUpdate = true; };

image.src = "models/brick\_poster.png";

var texture = new THREE.Texture( image, new THREE.UVMapping(), THREE.RepeatWrapping, THREE.RepeatWrapping );

texture.repeat.x = 8;

texture.repeat.y = 8;

for(var i = 0; i < materials.length; i++){

if(materials[i].name=="mySpecialMaterial"){

materials[i] = new THREE.MeshLambertMaterial( { map: texture } );

}

}

mesh = new THREE.Mesh( geometry, new THREE.MeshFaceMaterial( materials ) );

}

meshLoaded();

}

Here’s where we can do some very interesting things:

1. **Texture tiling.** You have to do this manually; it can’t be done through the JSON file (as far as I know). As you load the mesh, you can, seperately, load and create a new texture. With this new texture, you can set its THREE.RepeatWrapping for both X and Y, then set its repeat.x and repeat.y values. This will tile the texture to however large you want. You can then create a new material with this texture and assign it to the mesh with:

new THREE.MeshLambertMaterial( { map: texture } );

1. **Changing Materials of multi-material meshes.** Remember that name field in the JSON file? Here’s where it comes to play. The JSON loader will return an array of materials to you that represent the materials that the JSON model had in it. You can loop through those materials, and change specific materials. You can identify them by a name that you can manually set in the JSON file, simply change

"DbgName" : "lambert1.003",

to

"DbgName" : " mySpecialMaterial ",

You can now loop through the materials that the loader has given you and search for this one material, and modify it in any way you want.

The callbackProgress function

**function** callbackProgress( progress, result ) {

**var** bar = 250;

**if** ( progress.total ){

bar = Math.floor( bar \* progress.loaded / progress.total );

}

document.getElementById( "bar" ).style.width = bar + "px";

}

This function will be called thousands of times during a load of a JSON file. You can do whatever you want with it. In this example, it updates a loading bar. Porgress.loaded tells you how far the load is with regards to progress.total, which is the total amount to load.

And Voila! Put that mesh in your scene, and you can have an animated, UV’d, textured mesh that you build in Maya in your three.js scene.