

Índice

Animaciones.....	3
¿Cómo juntar diferentes archivos FBX en uno solo?.....	4
Configuración del FBX para poder usarlo en THREE.js.....	11
Plantilla de THREE.js.....	15
XAMPP y Apache.....	20
¿Cómo agregar OBJ a THREE.js?.....	23
¿Cómo agregar FBX a THREE.js?.....	25
¿Cómo activar dos o más animaciones de un FBX?.....	27
Otros métodos y propiedades de la animación en THREE.js.....	31

Animaciones

Para las animaciones hay dos opciones: una es que se realicen desde cero, es decir, que tú pongas el esqueleto de tu personaje y poco a poco lo vayas animando como tú quieras hasta terminar y guardarlo como FBX; la otra opción es usar una página de ayuda, en este caso yo recomiendo la página <https://www.mixamo.com/>.

En Mixamo puedes encontrar modelos ya creados y animaciones complejas. Puedes decidir utilizar uno de los modelos ya existentes, o bien, puedes cargar tu propio modelo y la página te guiará para que acoples el modelo a un esqueleto y puedas asignarle una animación de las ya establecidas.

Cuando ya tengas decidido tu modelo y tus animaciones, deberás descargarlas. El modelo deberá ser descargado en su pose original (pose T).

DOWNLOAD SETTINGS

Format
FBX(.fbx) ▼

Pose
T-pose ▼

CANCEL

DOWNLOAD

Las animaciones deberán ser descargadas en un formato de FBX y con su skin, a 30 frames por segundo y sin reducción de keyframes. Como dato, la reducción de keyframes es básicamente hacer la animación más sencilla, es decir, si tienes varios frames en tu animación que son muy similares o iguales, los va a eliminar porque no son necesarios en la animación.

DOWNLOAD SETTINGS

Format
FBX(.fbx) ▼

Skin
With Skin ▼

Frames per Second
30 ▼

Keyframe Reduction
none ▼

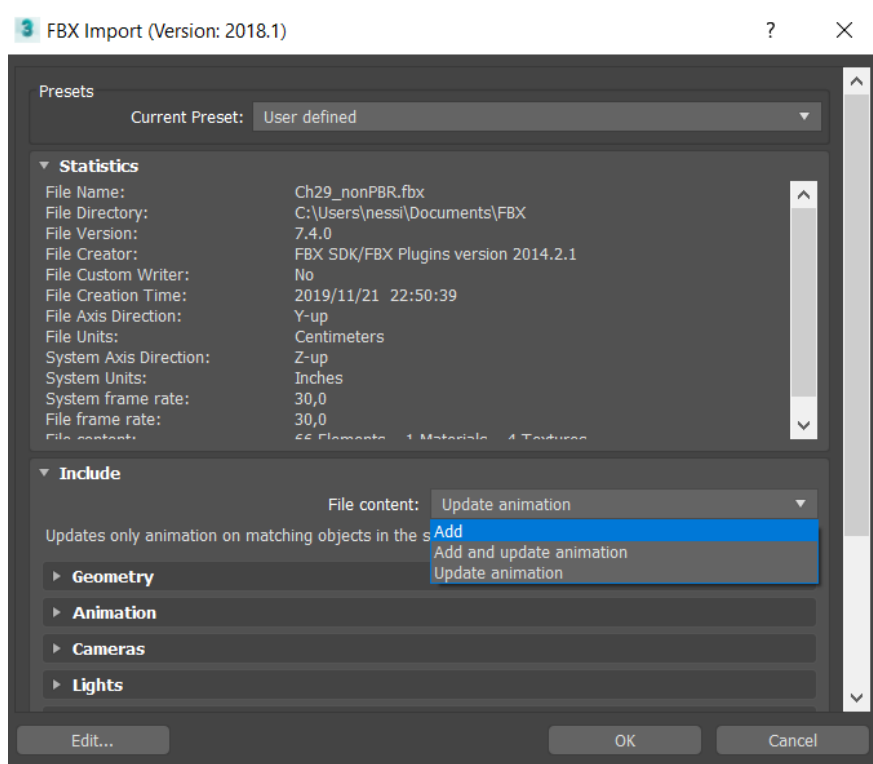
CANCEL

DOWNLOAD

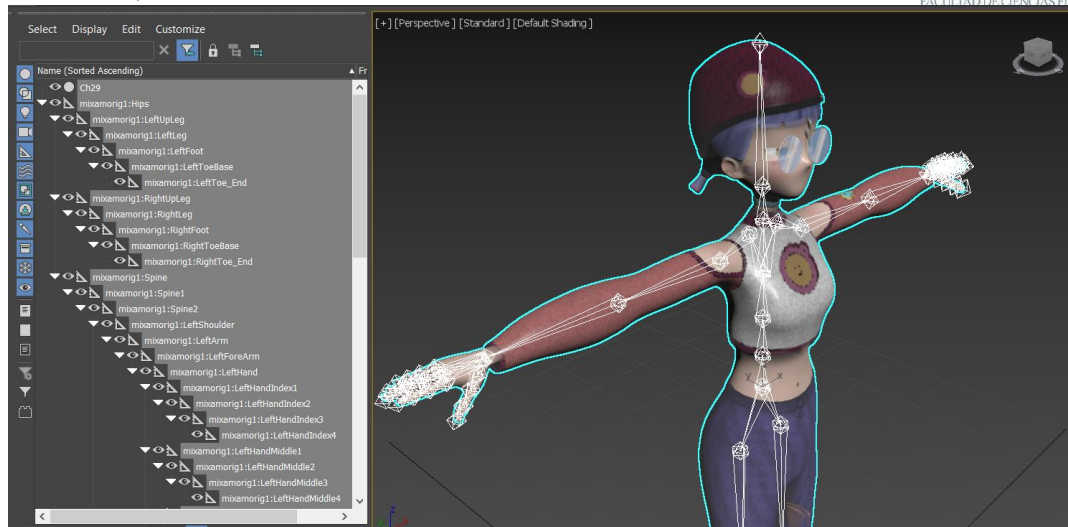
¿Cómo juntar diferentes archivos FBX en uno solo?

Los archivos FBX pueden juntarse desde 3DS Max o desde Maya, sin embargo, encuentro 3DS Max como la opción más sencilla, ya que en Maya el procedimiento es más tardado y en las últimas ediciones se han detectado algunos bugs que entorpecen la animación o simplemente no permite que se exporten de forma correcta. 3DS Max tiene una interfaz más amigable para hacer la combinación de animaciones, y al menos hasta su edición 2018 no se han detectado bugs o problemas complejos.

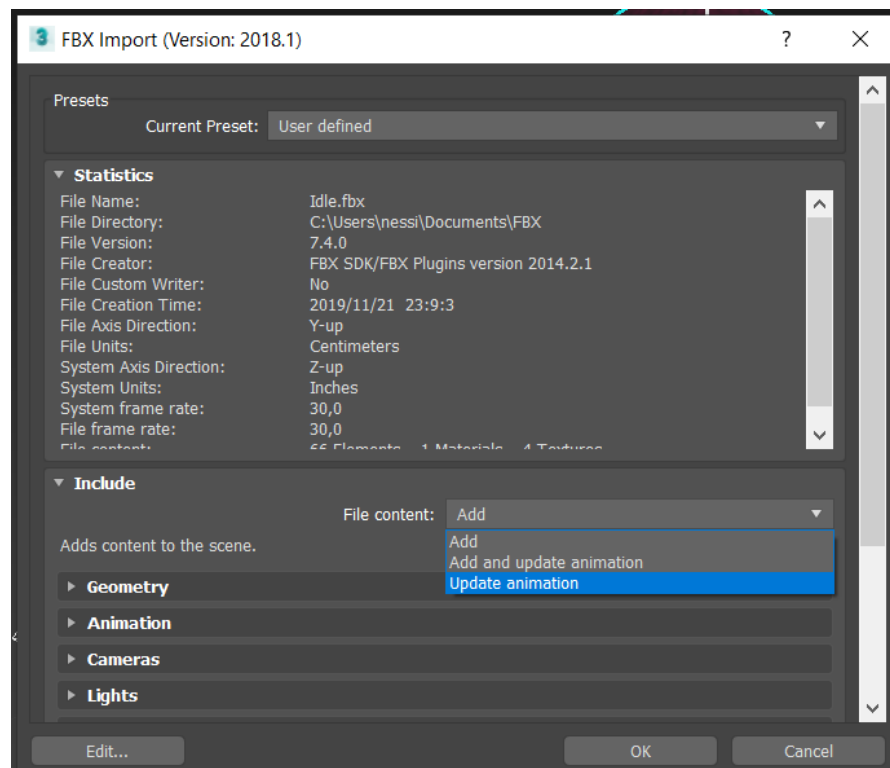
En 3DS Max lo primero que hay que hacer es importar en la escena nuestro modelo en pose T. Al momento de importar, aparecerá esta ventana de opciones, debes asegurarte de que en el apartado de File content esté seleccionada la opción de Add, ya que esta opción le dice al programa que se le añadirán elementos a nuestro modelo. Después se da click en OK.



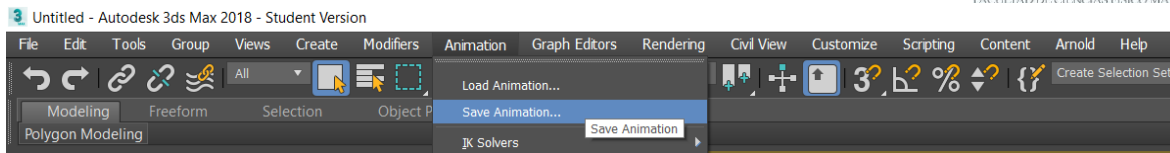
Selecciona absolutamente todas las partes del modelo, incluyendo el rig.



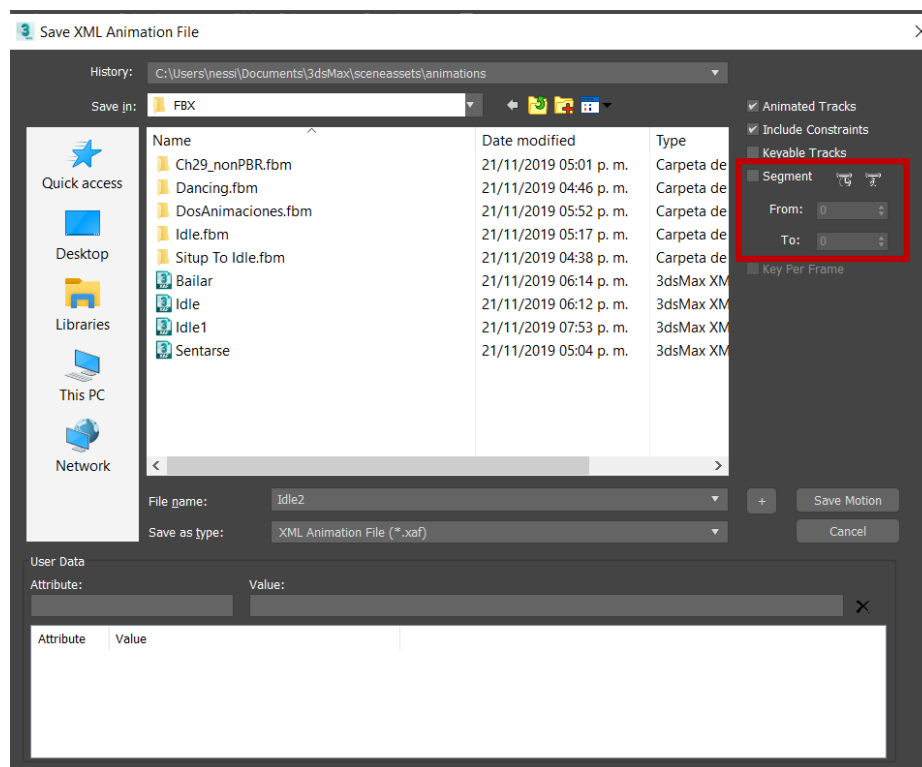
Ahora importaremos una de las animaciones, solo que en la ventana de opciones, seleccionaremos Update animation, que se traduce como actualizar animación, ya que no hay ninguna animación que actualizar, lo que va a hacer es agregar la animación al modelo.



Con la slider de animación confirmamos que la animación se haya colocado al modelo correctamente. Volvemos a seleccionar absolutamente todas las partes del modelo, incluyendo el rig, solo que ahora haremos click en la opción Animation del menú superior, y después haremos click en Save animation...



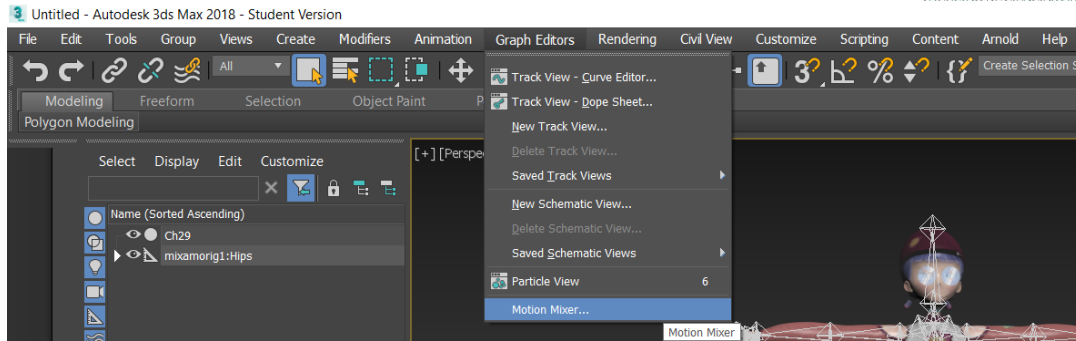
Aparecerá una ventana en donde deberás seleccionar la ubicación en donde guardarás tu animación. El formato en que se guardará es .xaf porque es un archivo en donde solo se guardan las acciones de la animación, no el render como tal. Si solo necesitas una parte de la animación, puedes seleccionar en la parte derecha de la ventana la opción Segment y señalar a partir de qué segundo deseas que inicie la animación y en qué segundo deseas que termine.



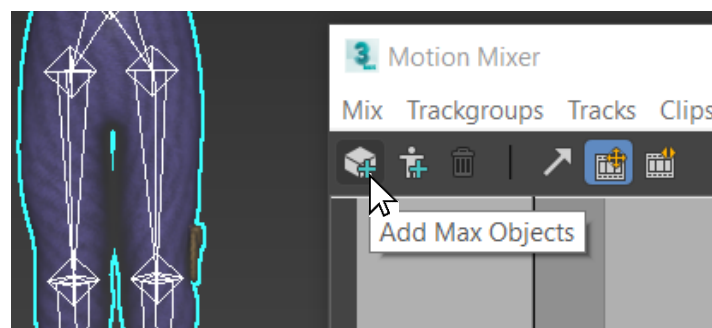
Ya que está guardada la primera animación, cargaremos la segunda. Nuevamente selecciona absolutamente todas las partes del modelo, incluyendo el rig. Importa la nueva animación, y en la ventana de opciones deja colocado el Update animation, como ahora sí existe una animación, lo que hará el Update animation es reemplazar la ya existente con la nueva. Repite los pasos que usaste para guardar la primera animación.

Una vez guardadas las dos animaciones, abrirás un archivo nuevo e importarás tu modelo en pose T, dejando la opción de Add marcada en la ventana de opciones para ésta vez poder agregar ambas animaciones al modelo.

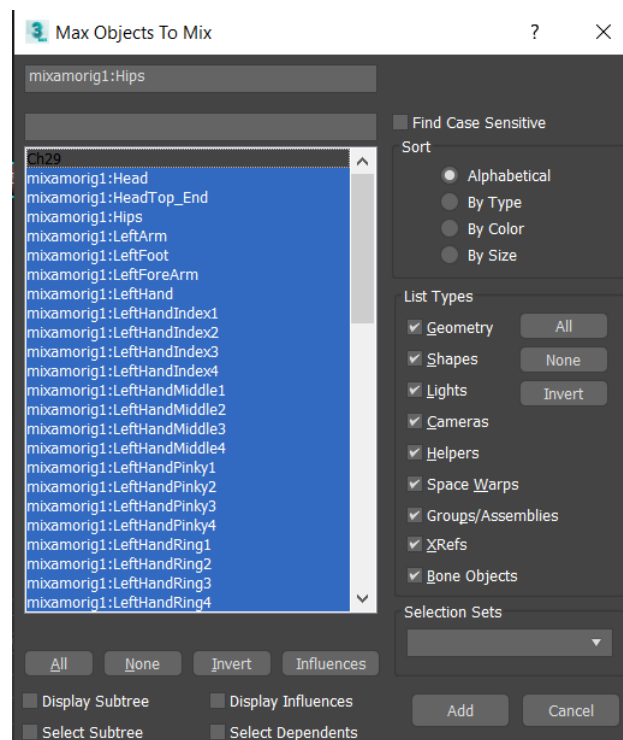
Ahora da click en la opción Graph Editors del menú superior, y después da click en Motion mixer...



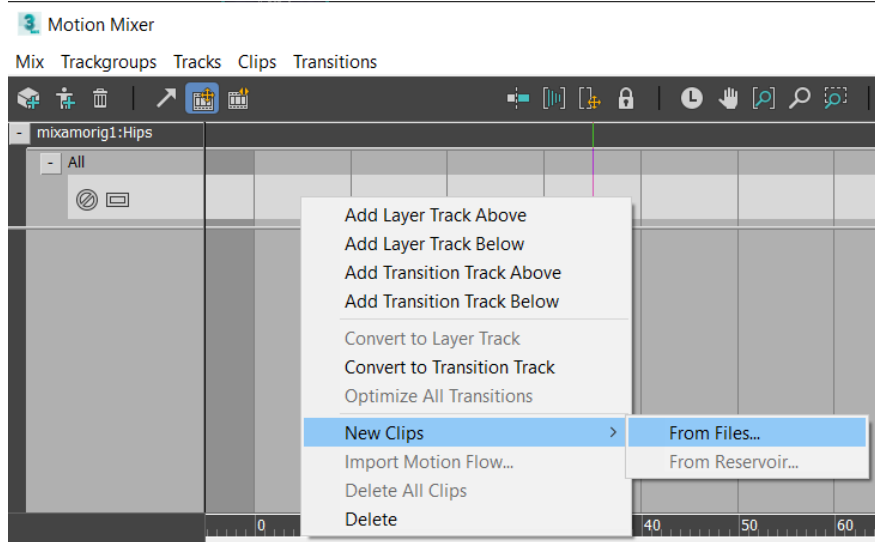
Motion mixer es una ventana que nos ayudará a mezclar las dos animaciones que guardamos previamente. Selecciona absolutamente todas las partes del modelo, incluyendo el rig. Después haz click en el icono de Add max objects en la ventana del Motion mixer.



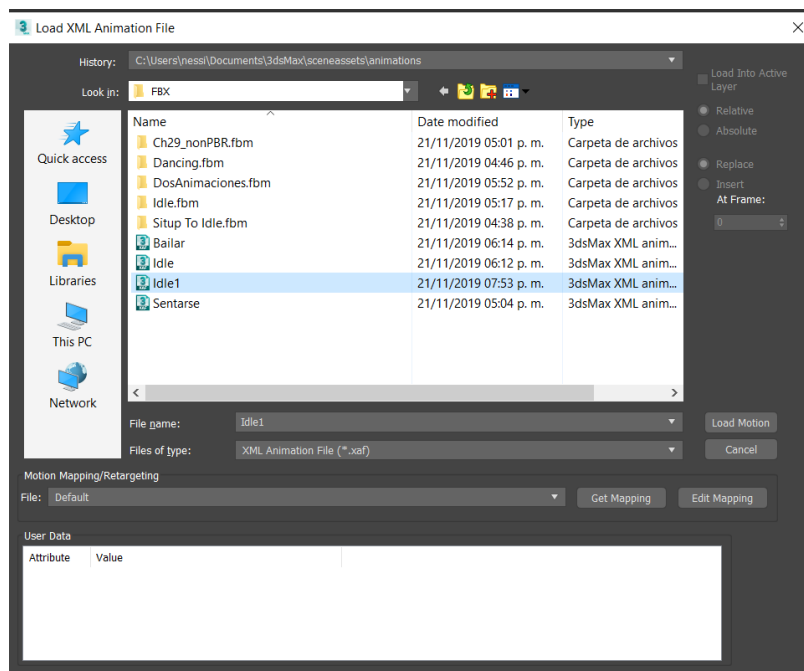
En la nueva ventana de Max objects o mix, vas a deseleccionar cualquier objeto que no sea un rig. En este caso, lo que está en azul es lo que sí está seleccionado, lo que se queda en gris es lo deseleccionado. Las demás opciones se quedan con lo default. Y haces click en Add.



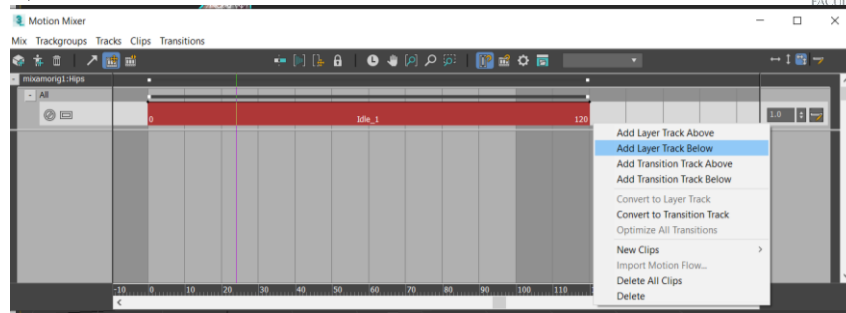
En la ventana de Motion mixer se va a crear una línea de tiempo gris. Haz click derecho en ella, selecciona la opción de New clips, y después la opción de From files...



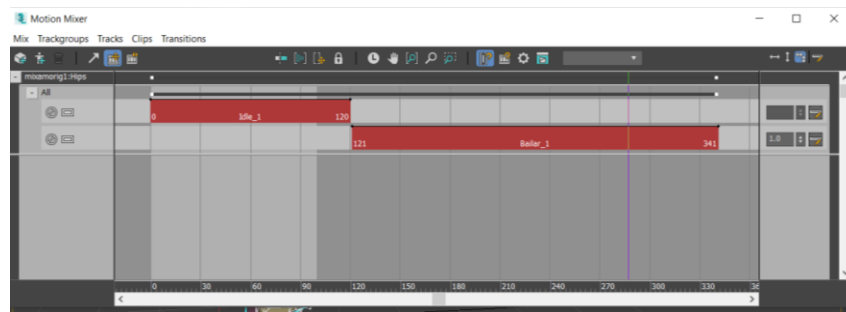
Aquí es donde vas a seleccionar una de las animaciones (.xaf) que guardaste anteriormente, sin modificar las opciones ya pre-establecidas en la ventana. Haz click en load Motion.



La animación debe agregarse como una barra de color rojo al Motion Mixer. Con un click izquierdo puedes arrastlarla y mover su posición en caso de que quieras que empiece o termine antes. Para agregar la segunda animación, da click derecho sobre la línea gris y selecciona Add layer track below.

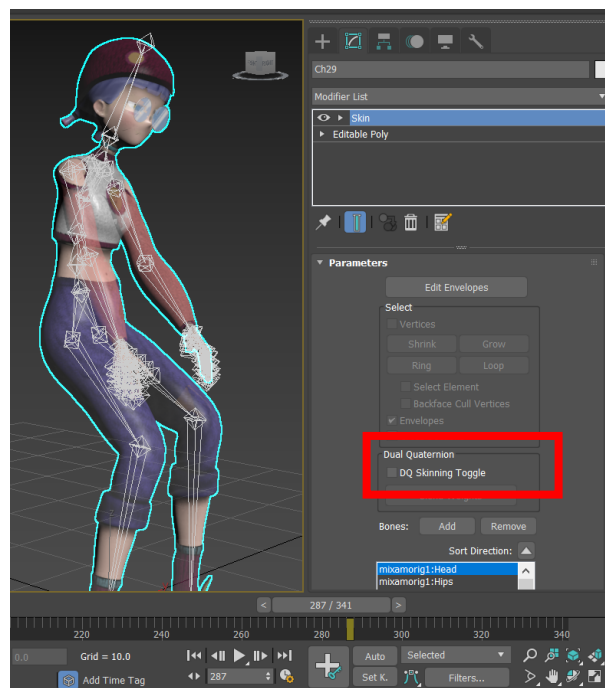


Esto agregará otra línea de tiempo gris debajo de la animación que ya tenemos. Sigue los mismos pasos para cargar la segunda animación. Cuando tengas ambas animaciones, mueve la barra roja de una de ellas para que las animaciones no se superpongan.

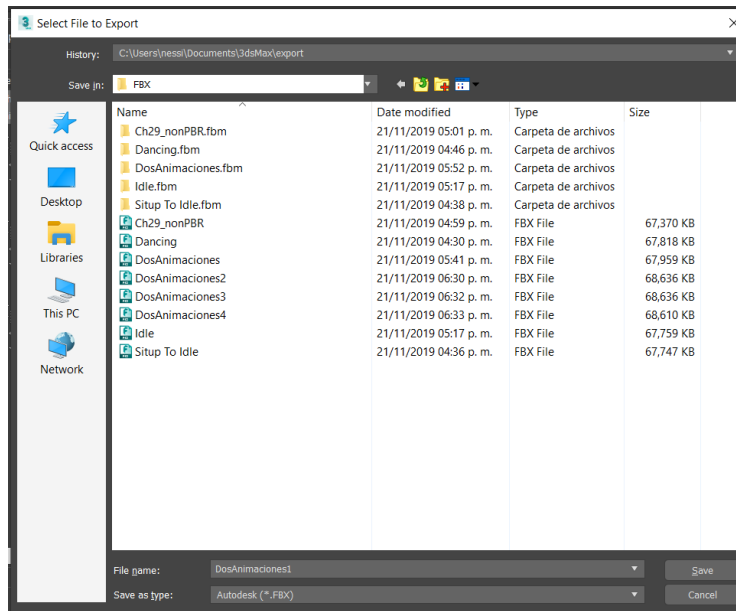


Reproduce la animación para asegurarte de que todo está en orden.

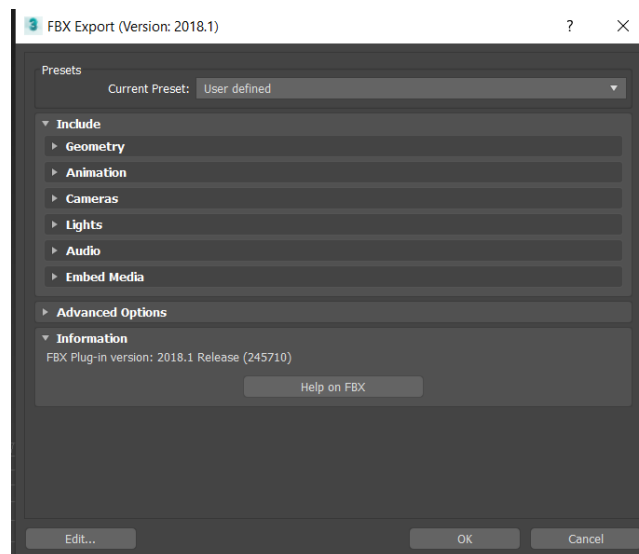
Antes de exportar, selecciona solamente el skin de tu modelo, y a tu derecha en el apartado de Modify, en la sección de Parameters, asegúrate de que la opción DQ Skinning Toggle esté desactivada. Si la opción se deja activada, la skin es deformada (aunque no visiblemente) y no permite un funcionamiento correcto del esqueleto, es decir, la animación no se exportará bien.



Después exporta todo como un FBX.



No muevas ninguna de las opciones que aparecen en la segunda ventana, todo se queda como predeterminado. Da click en OK.

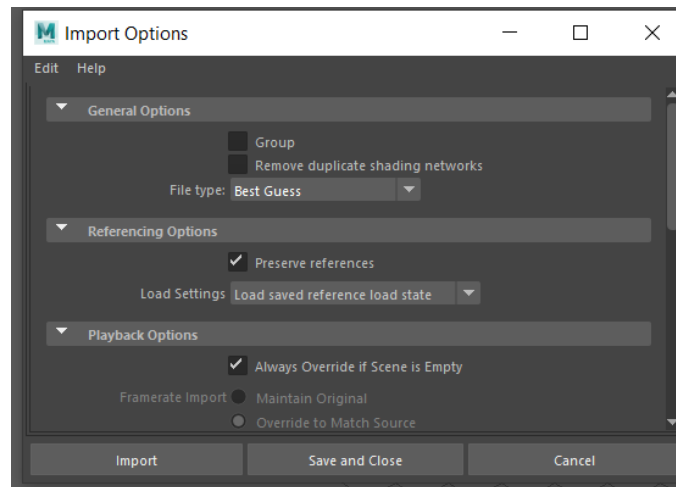


Y ahora tienes tu FBX con múltiples animaciones. Si aún tienes alguna duda, puedes revisar éste video en donde explican todo de manera más visual:
<https://www.youtube.com/watch?v=ztoxsMtH5Bc>

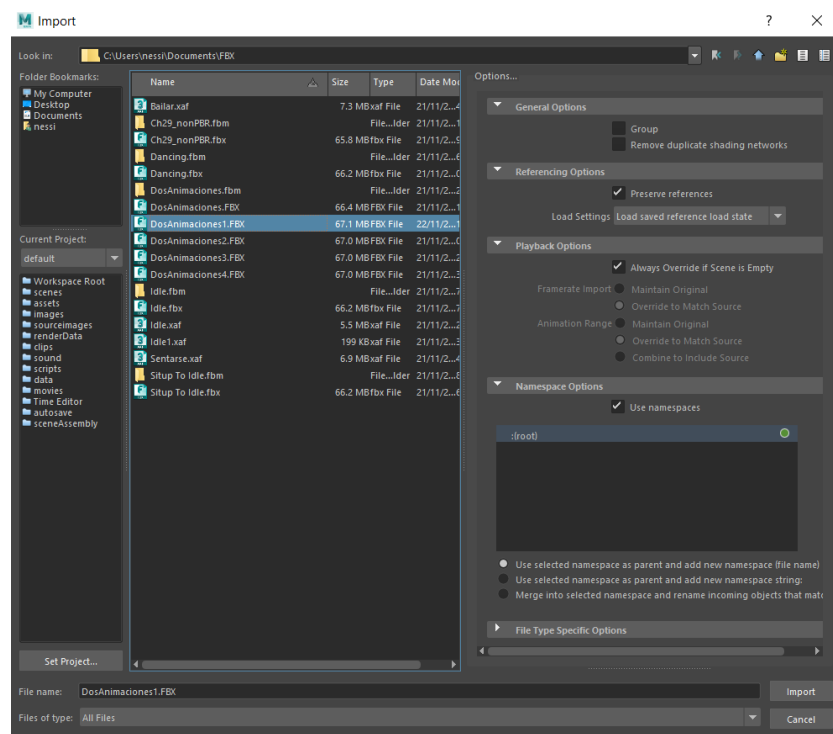
Configuración del FBX para poder usarlo en THREE.js

Para hacer la configuración del archivo FBX, utilizaremos el programa de animación Maya. El programa Maya es conocido por la calidad y fluidez de sus animaciones, aparte de que es uno de los programas más usados para las animaciones de videojuegos, especialmente cuando se trata de animaciones para Unreal, ya que provee más opciones de exportación especializadas para videojuegos.

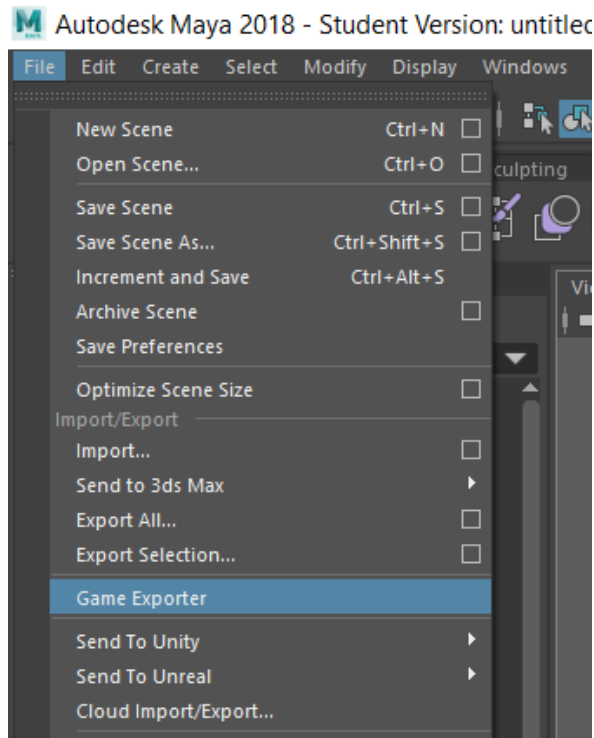
Primero hay que importar el archivo FBX. En la ventana de opciones para importar, se quedan las opciones por default. Da click en Import.



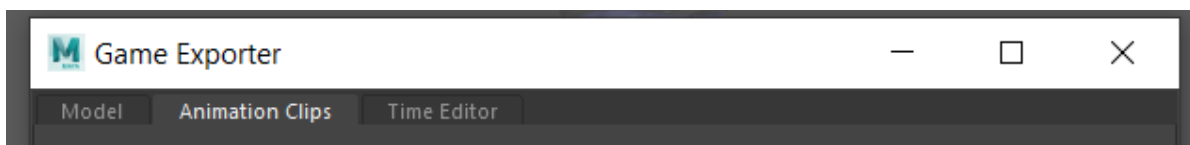
En la segunda ventana, también se quedan las opciones por default. Selecciona tu FBX y da click en Import.



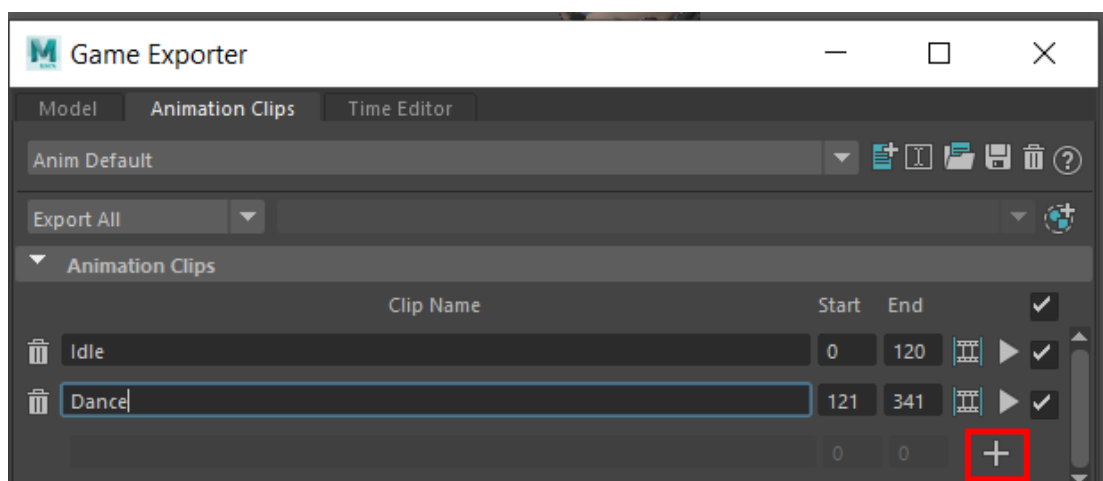
Ahora, vamos a exportar el archivo, pero no como generalmente exportamos las animaciones, sino con el Game Exporter, que es especial para las animaciones de videojuegos.



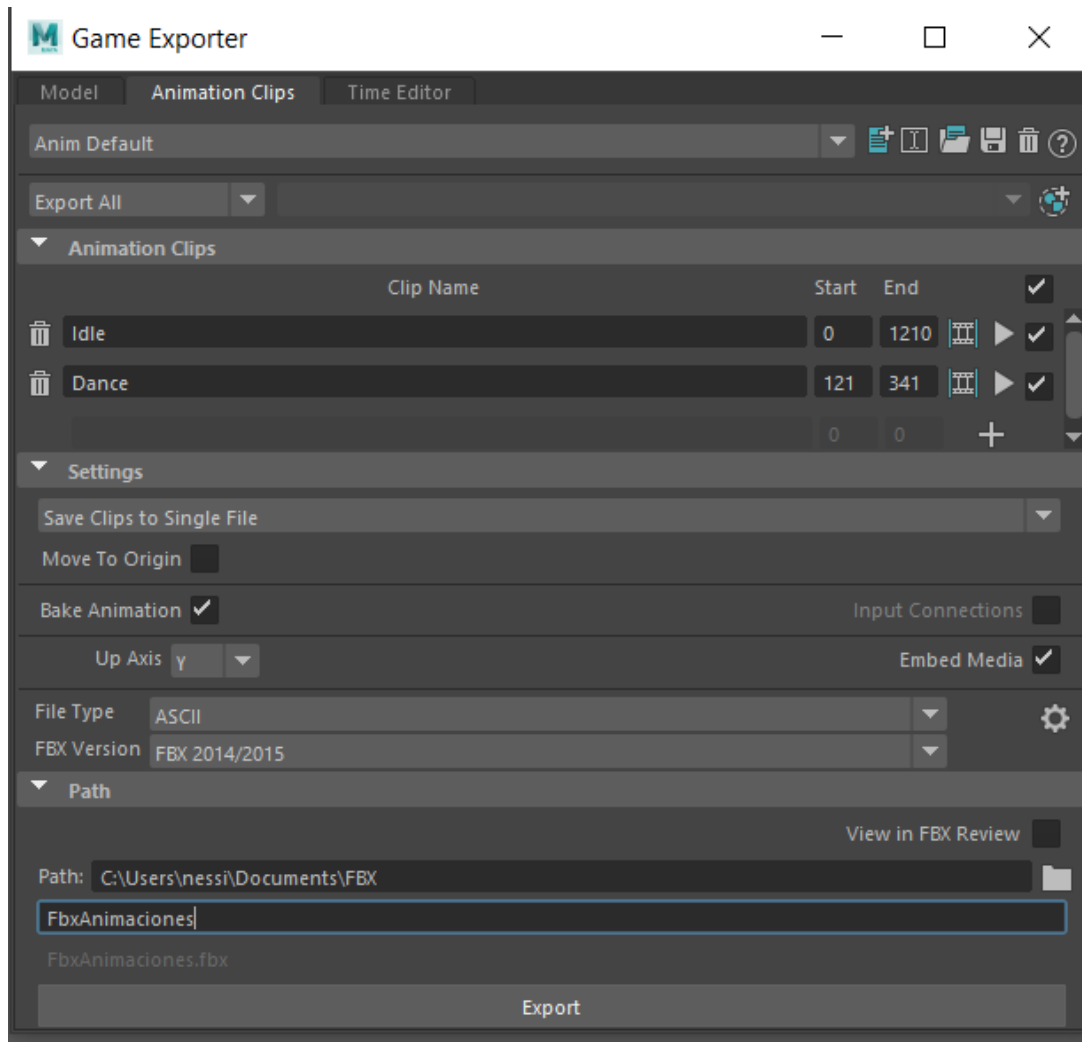
Una vez en la ventana de Game Exporter, haremos click en la pestaña superior de Animation Clips.



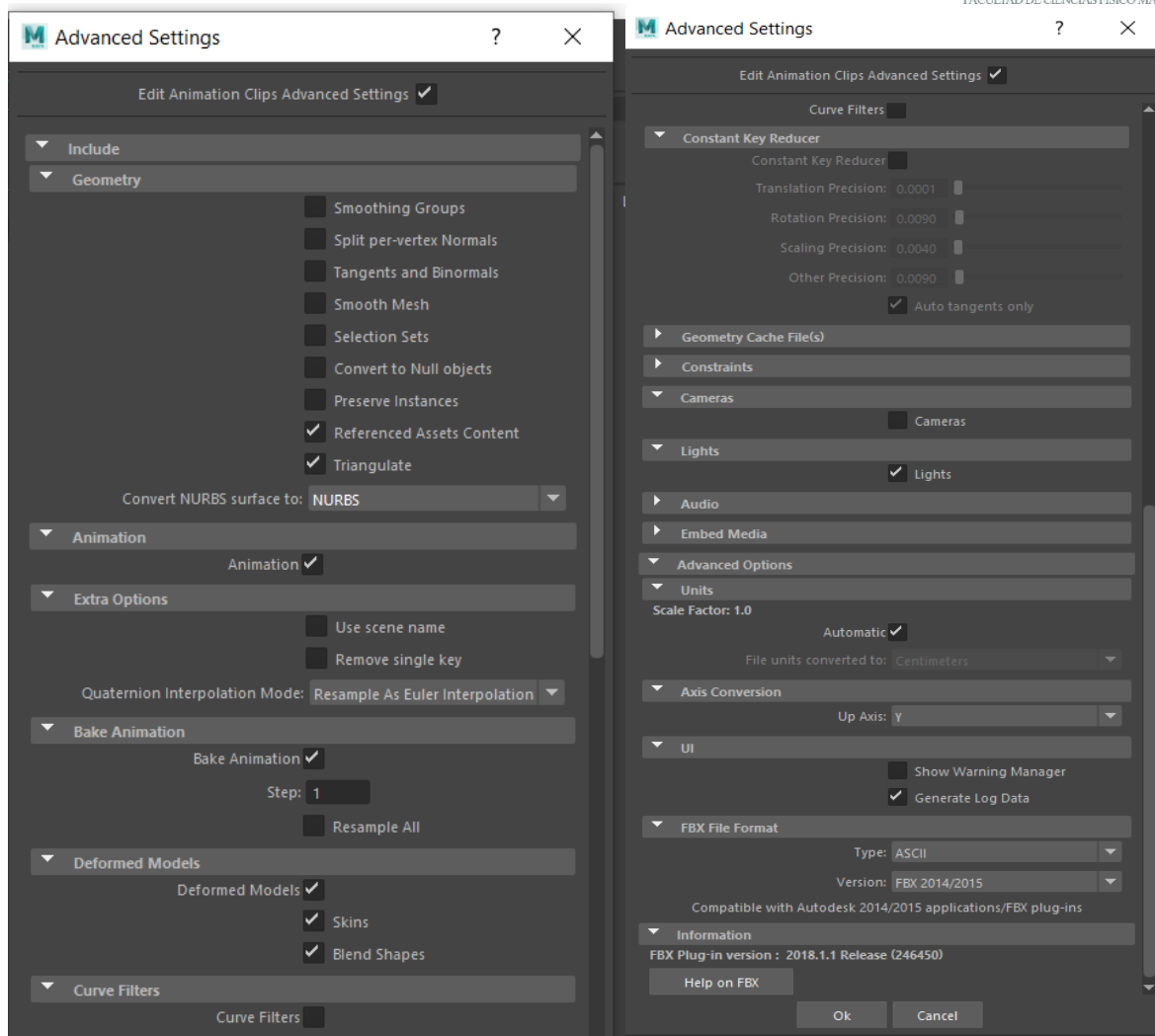
En el apartado de Animation Clips, vamos a agregar tantos clips como tantas animaciones tengamos. En mi caso, tengo dos animaciones, por lo que agregaré dos clips. A cada clip le asignas un nombre, en qué frame inicia y en qué frame termina.



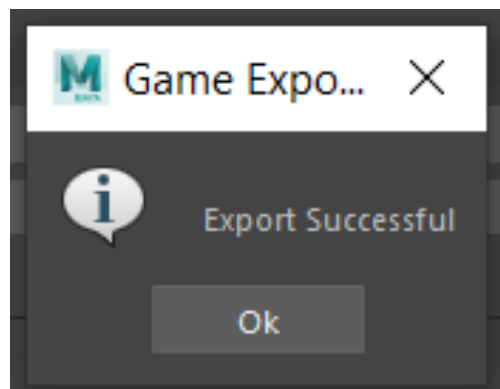
En lo que resta de la ventana, coloca la siguiente configuración, sin olvidar que en Path irá la carpeta en donde quieres guardar tu FBX.



Antes de exportar, da click en el engranaje que se encuentra en el apartado de Settings. Se abrirá una segunda ventana donde colocarás la siguiente configuración, cuando termines, da click en OK.



Ahora puedes dar click en Export. Al finalizar el proceso debe aparecer una ventana de que todo se hizo correctamente.



Plantilla de THREE.js

Para comenzar, explicaré de manera breve el cómo hay que hacer una plantilla de THREE.js utilizando Sublime Text.

Hay que tener una base de html:

```
View Goto Tools Project Preferences Help
index1.html x index.html
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Usar FBX en THREE.js</title>
5 </head>
6 <body>
7
8 </body>
9 </html>
```

Dentro de <head> vamos a colocar las librerías que son necesarias para que THREE.js pueda funcionar, así como otras librerías para poder cargar OBJ y FBX en el archivo.

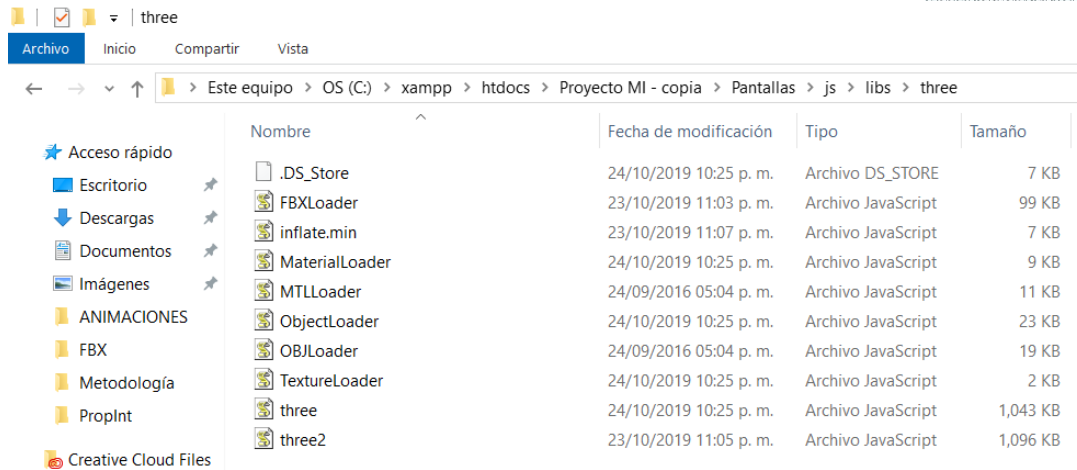
La primera librería es prácticamente jQuery, y se puede descargar desde <https://jquery.com/>

La segunda es THREE.js y se puede descargar desde <https://github.com/mrdoob/three.js/blob/dev/build/three.js>

La librería de MTLLoader y OBJLoader nos ayudarán a poder utilizar los archivos OBJ, mientras que FBXLoader e inflate nos ayudarán con los archivos FBX. Las librerías Loaders las podemos descargar desde <https://github.com/mrdoob/three.js/tree/dev/examples/js/loaders>, mientras que inflate lo podemos conseguir en <https://github.com/mrdoob/three.js/tree/dev/examples/js/libs>

Recuerda que estas librerías deben estar en tus carpetas del programa, en mi caso, las copié y pegué en la ruta js/libs/three/.

```
View Goto Tools Project Preferences Help
index1.html x
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Usar FBX en THREE.js</title>
5
6   <script type="text/javascript" src="js/libs/jquery/jquery-2.1.4.min.js"></script>
7   <script type="text/javascript" src="js/libs/three/three2.js"></script>
8   <script type="text/javascript" src="js/libs/three/MTLLoader.js"></script>
9   <script type="text/javascript" src="js/libs/three/FBXLoader.js"></script>
10  <script type="text/javascript" src="js/libs/three/OBJLoader.js"></script>
11  <script type="text/javascript" src="js/libs/three/inflate.min.js"></script>
12
13 </head>
14 <body>
15
16 </body>
17 </html>
```



Dentro de un `<script>` colocaremos un `document ready`, el cual nos ayuda a saber cuándo la página está lista o completamente cargada. El código dentro del `document ready` solo correrá una vez que el DOM esté listo para JavaScript.

También de una vez, agregaremos la función `render`. La función `render` es en la que se harán las animaciones.

```
View Goto Tools Project Preferences Help

index1.html index.html x

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Usar FBX en THREE.js</title>
5
6   <script type="text/javascript" src="js/libs/jquery/jquery-2.1.4.min.js"></script>
7   <script type="text/javascript" src="js/libs/three/three2.js"></script>
8   <script type="text/javascript" src="js/libs/three/MTLLoader.js"></script>
9   <script type="text/javascript" src="js/libs/three/FBXLoader.js"></script>
10  <script type="text/javascript" src="js/libs/three/OBJLoader.js"></script>
11  <script type="text/javascript" src="js/libs/three/inflate.min.js"></script>
12
13  <script type="text/javascript">
14    $(document).ready(function() {
15
16    }
17
18    function render() {
19
20    }
21  }
22 </script>
23
24 </head>
25 <body>
26
27 </body>
28 </html>
```

Fuera de la función `document ready`, colocaremos nuestras primeras variables globales, que son prácticamente los tres componentes básicos de la escena:

La escena, que es la colección de objetos de `THREE.js`.

El renderer, que se encarga de dibujar todo lo que está en una escena.

Y la cámara, que nos permite ver todo lo que dibuja el renderer.

View Goto Tools Project Preferences Help

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Usar FBX en THREE.js</title>
5
6    <script type="text/javascript" src="js/libs/jquery/jquery-2.1.4.min.js"></script>
7    <script type="text/javascript" src="js/libs/three/three2.js"></script>
8    <script type="text/javascript" src="js/libs/three/MTLLoader.js"></script>
9    <script type="text/javascript" src="js/libs/three/FBXLoader.js"></script>
10   <script type="text/javascript" src="js/libs/three/OBJLoader.js"></script>
11   <script type="text/javascript" src="js/libs/three/inflate.min.js"></script>
12
13   <script type="text/javascript">
14
15     var scene;
16     var renderer;
17     var camera;
18
19     $(document).ready(function() {
20

```

Ahora podemos colocar todos los elementos básicos en el document ready. Recordemos que, si no hay un canvas, prácticamente no se puede hacer nada en THREE.js.

View Goto Tools Project Preferences Help

```

19   $(document).ready(function() {
20
21     //Tamaño del canvas
22     var canvasSize = {
23       width: window.innerWidth,
24       height: window.innerHeight
25     }
26
27     //Inicializar el renderer
28     renderer = new THREE.WebGLRenderer();
29
30     //Con lo que se limpia la pantalla
31     renderer.setClearColor( new THREE.Color(1,1,1) );
32
33     renderer.setSize(
34       canvasSize.width,
35       canvasSize.height
36     );
37
38     //Inicializar la camara
39     camera = new THREE.PerspectiveCamera(
40       //campo de visión / field of view
41       75,
42       //Relación aspecto
43       canvasSize.width / canvasSize.height,
44       //Que tan cerca se va a ver
45       0.1,
46       //Que tan lejos se va a ver
47       100
48     );
49
50     //Inicializamos la escena
51     scene = new THREE.Scene();
52
53     //Agregamos la etiqueta canvas dentro del DIV
54     $('#scene-section').append(renderer.domElement);
55
56     //Mandamos a llamar la función render
57     render();
58
59   }
60
61   function render() {
62
63   }
64
65 </script>
66
67 </head>
68 <body>
69
70   <div id="scene-section"/>
71
72 </body>
73 </html>

```

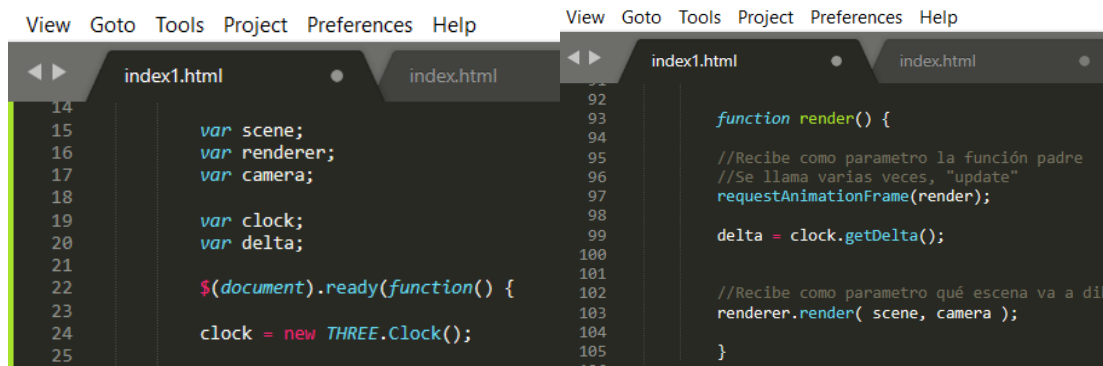

Ahora colocamos los elementos básicos de la función render.

```
View Goto Tools Project Preferences Help
index1.html index.html x
61
62
63     function render() {
64
65         //Recibe como parametro la función padre
66         //Se llama varias veces, "update"
67         requestAnimationFrame(render);
68
69
70         //Recibe como parametro qué escena va a dibujar, y la cámara que se va a utilizar
71         renderer.render( scene, camera );
72
73     }
74
75 </script>
76
77 </head>
78 <body>
79
80     <div id="scene-section"/>
81
82 </body>
83 </html>
```

Después agregaremos elementos de iluminación dentro de la función document ready, antes de mandar a llamar la función de render.

```
View Goto Tools Project Preferences Help
index1.html index.html
56
57         //Iluminación
58         var ambient = new THREE.AmbientLight(
59             //Color
60             new THREE.Color(1, 1, 1),
61             //Intensidad
62             1.0
63         );
64
65         var directional = new THREE.DirectionalLight(
66             //Color
67             new THREE.Color(1, 1, 0),
68             //Intensidad
69             0.4
70         );
71
72         //Posición de la luz direccional
73         directional.position.set(0, 0, 1);
74
75         //Ambos tipos de iluminación se agregan a la escena
76         scene.add(ambient);
77         scene.add(directional);
78
79
80
81         //Mandamos a llamar la función render
82         render();
83
84     }
85
```

Agregamos otras dos variables globales que nos serán de ayuda para tener un buen seguimiento del tiempo en la escena: clock y delta. Estas dos variables son muy útiles respecto a casos de animación, toman los segundos para que las animaciones se vean fluidas.

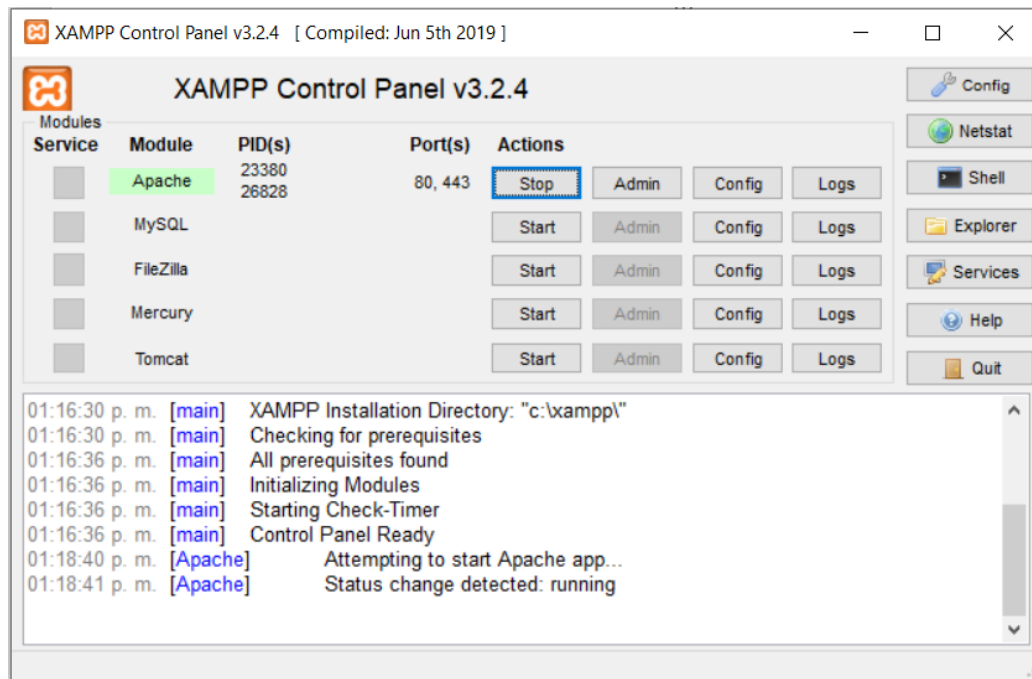


```
View Goto Tools Project Preferences Help
index1.html index.html
14
15     var scene;
16     var renderer;
17     var camera;
18
19     var clock;
20     var delta;
21
22     $(document).ready(function() {
23
24         clock = new THREE.Clock();
25
View Goto Tools Project Preferences Help
index1.html index.html
92
93     function render() {
94
95         //Recibe como parametro la función padre
96         //Se llama varias veces, "update"
97         requestAnimationFrame(render);
98
99         delta = clock.getDelta();
100
101
102         //Recibe como parametro qué escena va a dibujar
103         renderer.render( scene, camera );
104
105     }
106
```

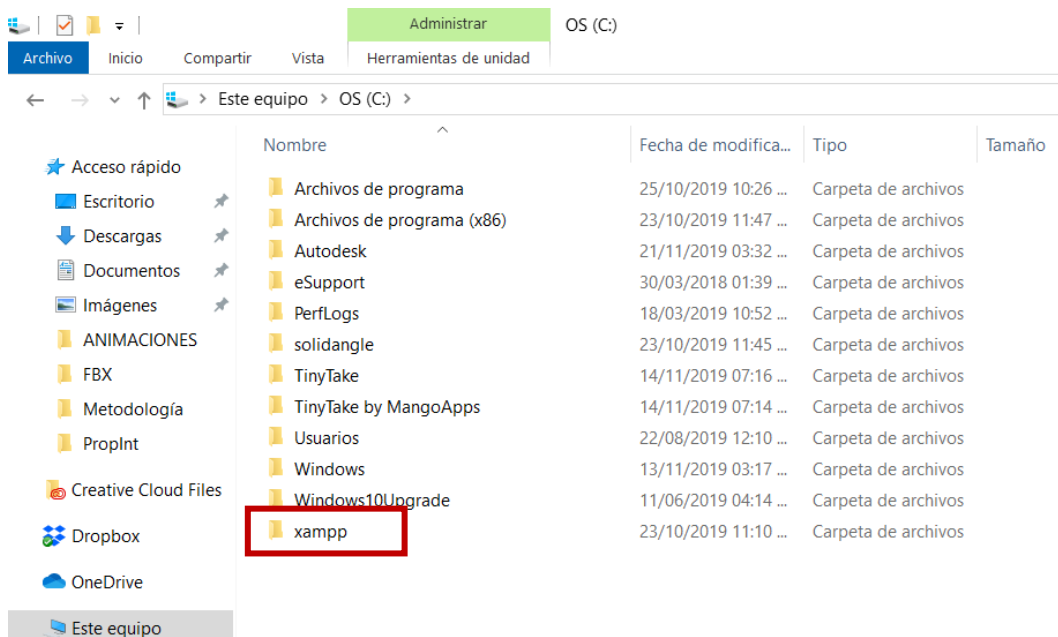
Puedes cambiar el color del renderer.setClearColor(new THREE.Color(1, 1, 1)); solo para verificar que el canvas esté funcionando.

XAMPP y Apache

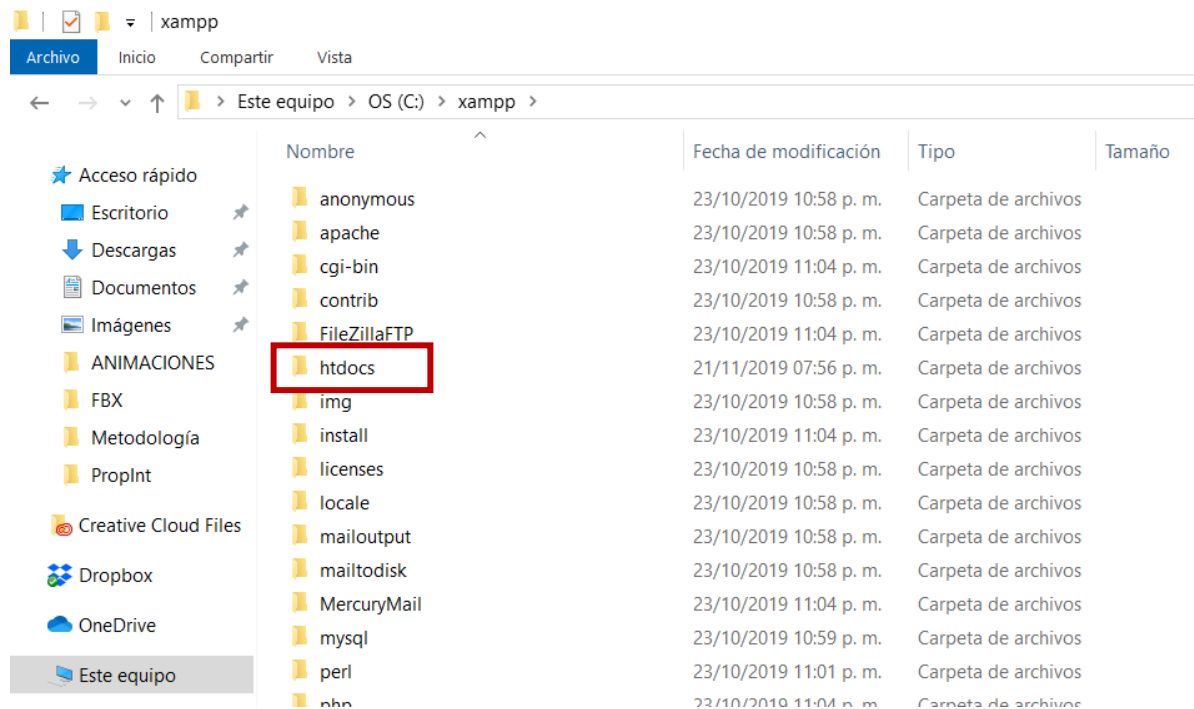
Para el siguiente paso que será cargar los OBJ y los FBX, hay que tener en cuenta que deberás contar con XAMPP y activar Apache, ya que los modelos externos a las figuras básicas de THREE.js no pueden ser vistos directo en el navegador sin una configuración.



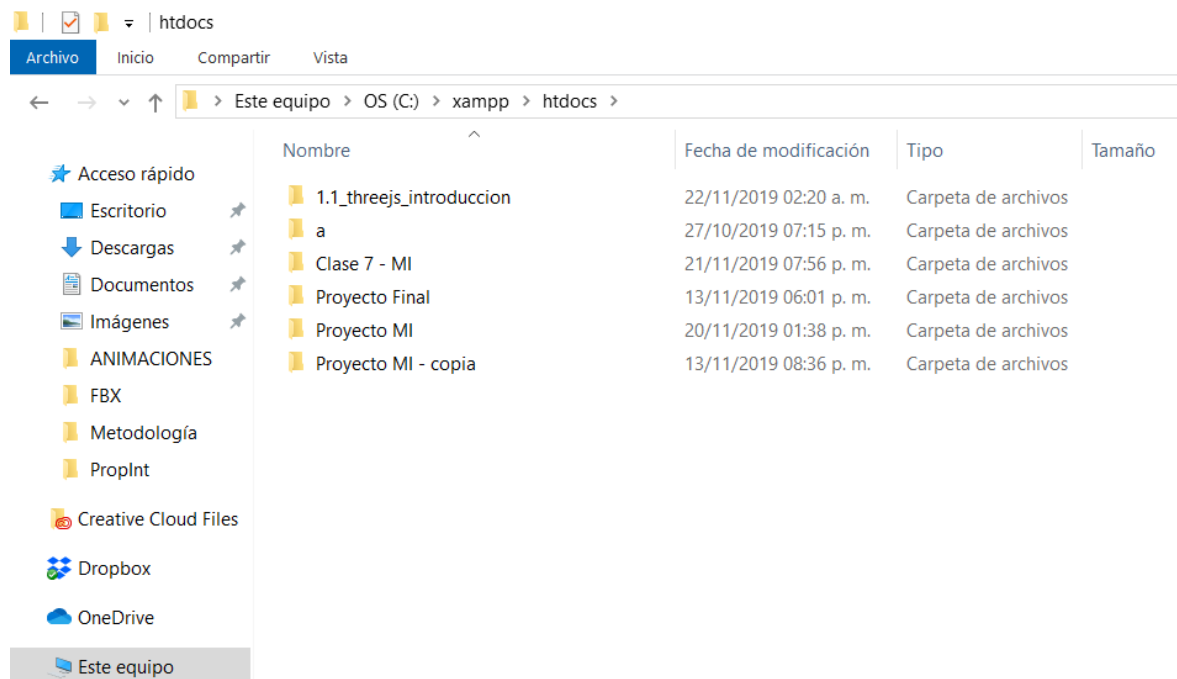
Los archivos de tu proyecto los colocarás en una carpeta especial de XAMPP. Primero debes dirigirte a tu equipo y hacer abrir la carpeta de XAMPP.



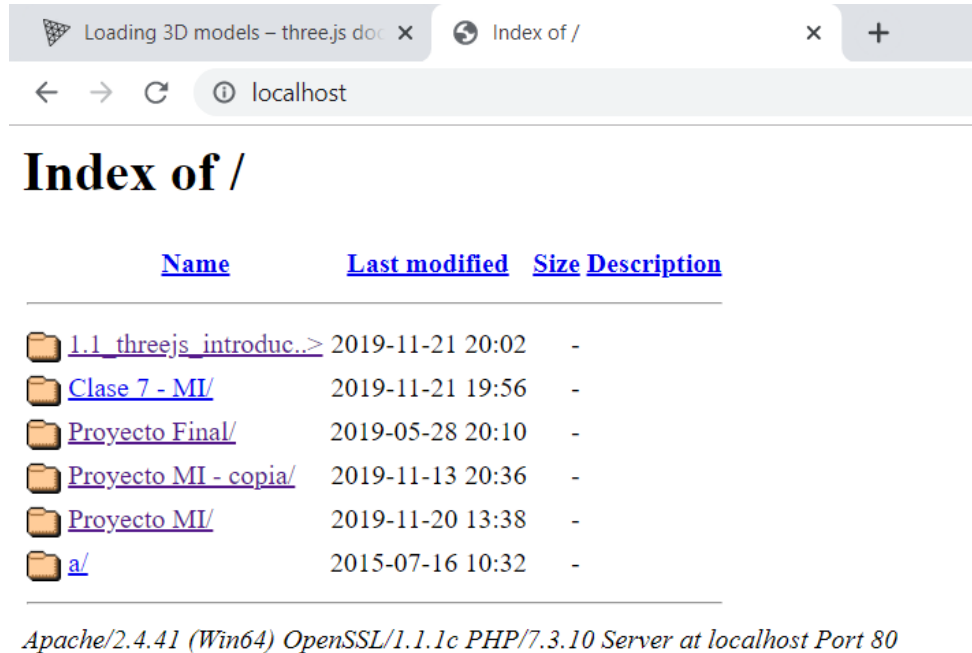
Una vez ahí, abrirás la carpeta de htdocs.



En esa carpeta puedes colocar tu proyecto con el que cargarás OBJ y/o FBX.



La plantilla ahora deberás correrla desde el local host. Abre tu navegador y coloca localhost en el buscador, esto te llevará a donde se encuentran tus carpetas de proyectos (dentro de htdocs). Haz doble click sobre el archivo de tu proyecto, y listo.



Index of /

Name	Last modified	Size	Description
1.1_threejs_introduc..>	2019-11-21 20:02	-	
Clase 7 - MI/	2019-11-21 19:56	-	
Proyecto Final/	2019-05-28 20:10	-	
Proyecto MI - copia/	2019-11-13 20:36	-	
Proyecto MI/	2019-11-20 13:38	-	
a/	2015-07-16 10:32	-	

Apache/2.4.41 (Win64) OpenSSL/1.1.1c PHP/7.3.10 Server at localhost Port 80

¿Cómo agregar OBJ a THREE.js?

Ya que la plantilla está lista y podemos abrirla desde el navegador, podemos comenzar a agregar los elementos, iniciando por los OBJ, que son los modelos más sencillos por no tener animación, dentro de la función document ready.

```
View Goto Tools Project Preferences Help
index1.html index.html
79
80 //Ambos tipos de iluminación se agregan a la escena
81 scene.add(ambient);
82 scene.add(directional);
83
84
85 // Importar Modelos OBJ
86 loadOBJWithMTL(
87     //Carpeta en donde está el modelo
88     'Modelos/',
89     //El archivo obj del modelo
90     'Gota4.obj',
91     //El archivo mtl del modelo
92     'Gota4.mtl',
93     (miObjetoYaCargado)=>{
94         //Posición del objeto en x, y, z
95         miObjetoYaCargado.position.z= 0;
96         miObjetoYaCargado.position.x= 0;
97         miObjetoYaCargado.position.y= 0;
98
99         //Escala del objeto en x, y, x
100         miObjetoYaCargado.scale.set (0.4, 0.4, 0.4);
101
102         //Se le asigna un nombre al objeto
103         miObjetoYaCargado.name = 'Gota';
104
105         //Se agrega el objeto a escena
106         scene.add(miObjetoYaCargado);
107     }
108 );
109
110
111 //Mandamos a llamar la función render
112 render();
113
114 }
115
```

Fuera de la función document ready se hace una función para leer el archivo mtl.

```
View Goto Tools Project Preferences Help
index1.html index.html
104
105 //Se agrega el objeto a escena
106 scene.add(miObjetoYaCargado);
107 }
108 );
109
110 //Mandamos a llamar la función render
111 render();
112
113 }
114
115 //Cargar mtl
116 function loadOBJWithMTL(path, objFile, mtlFile, onLoadCallback) {
117     var mtlLoader= new THREE.MTLLoader();
118     mtlLoader.setPath(path);
119
120     mtlLoader.load(mtlFile,(material)=>{
121         //primero cargas materiales, luego el objeto, luego se lo aplicamos
122         //Este bloque se ejecuta cuando el material ya este cargado
123         //material parametro va a ser el material cargado
124         var objLoader = new THREE.OBJLoader();
125         objLoader.setPath(path);
126         objLoader.setMaterials(material);
127         objLoader.load(objFile,(object3D)=>{
128             //Este bloque se ejecuta cuando el OBJ ya este cargado
129             onLoadCallback(object3D);
130         });
131     });
132 }
133
134
```

Prácticamente con eso debería cargar un modelo OBJ, sin embargo, hay veces en que el modelo aparece en color negro o gris, esto se debe a que el mtl no se exportó de manera correcta.

La mejor manera de corregirlo es agregando el archivo de la textura al mtl y colocando los valores de Ka y Kd como en la siguiente imagen:

```
View Goto Tools Project Preferences Help
juego.html x Gota4.mtl
1 newmtl moontear:material_01713A59
2 illum 4
3 Ka 0.2000 0.2000 0.2000
4 Kd 0.8000 0.8000 0.8000
5 Tf 1.00 1.00 1.00
6 map_Kd zelda_gi_moonte_0.png
7 map_Ka zelda_gi_moonte_0.png
8 Ni 1.50
9 Ks 0.20 0.20 0.20
10 Ns 43.22
11
12
```

¿Cómo agregar FBX a THREE.js?

Para cargar un modelo FBX, el procedimiento no es tan distinto al de un OBJ. Primero agregaremos algunas variables globales que nos serán de ayuda para cargar el FBX.

Var mixers = []; éste arreglo de mixers nos servirá para tener un control sobre las animaciones.

Var action, action2; nos permitirán ejecutar las animaciones que deseamos.

```
View Goto Tools Project Preferences Help
index1.html indexhtml
14
15     var scene;
16     var renderer;
17     var camera;
18
19     var clock;
20     var delta;
21
22     var mixers = [];
23     var action, action2;
24
25     $(document).ready(function() {
26
27         clock = new THREE.Clock();
```

Agregaremos el código para cargar un FBX dentro de la función document ready, antes de mandar a llamar la función render.

```
View Goto Tools Project Preferences Help
index1.html indexhtml
113
114
115     var loader = new THREE.FBXLoader();
116     loader.load('Modelos/DosAnimaciones3.fbx', function (personaje){
117
118         //AnimationMixer es un reproductor de animaciones
119         personaje.mixer = new THREE.AnimationMixer(personaje);
120
121         //El reproductor de animaciones de nuestro modelo
122         //entra al arreglo mixers
123         mixers.push(personaje.mixer);
124
125         //En esta parte es donde nos sirve el haber preparado el archivo FBX
126         //en Maya, al haberlo exportado como un export de videojuego
127         //internamente el archivo separa las animaciones y así
128         //podemos decir que una es la animación 0 y otra es la animación 1
129         //Si tuvieramos más de dos animaciones, los numeros irían aumentando
130         action = personaje.mixer.clipAction(personaje.animations[0]);
131         action2 = personaje.mixer.clipAction(personaje.animations[1]);
132
133         //Aplicamos play a ambas animaciones
134         action.play();
135         action2.play();
136
137         //Le asignamos peso a una de las animaciones
138         //la de mayor peso es la que se va a reproducir
139         //NOTA: el peso menor es 0 y el mayor es 1
140         //se puede combinar pesos, por ejemplo
141         //ponerle a ambos 0.5, entonces las animaciones
142         //se mezclarían
143         action.weight = 0;
144         action2.weight = 1;
145
146         //Le damos la posición que queramos en x, y, z
147         personaje.position.z = -10;
148         personaje.position.x = -10;
149         personaje.position.y = -5;
150
151         //Le asignamos una escala en x, y, z
152         personaje.scale.set (0.08, 0.08, 0.08);
153
154         //Le asignamos un nombre al modelo
155         personaje.name = 'Monita';
156
157         //Agregamos el personaje a escena
158         scene.add(personaje);
159
160     });
161
162
163     //Mandamos a llamar la función render
164     render();
165
166 }
167
```


Para que se pueda realizar la animación del FBX, haremos una actualización de los frames en el render.

```
View Goto Tools Project Preferences Help

index1.html index.html

187
188
189     function render() {
190
191         //Recibe como parametro la función padre
192         //Se llama varias veces, "update"
193         requestAnimationFrame(render);
194
195         delta = clock.getDelta();
196
197         //Mide el arreglo de mixers, mientras no esté vacío
198         //lo va a recorrer y hará el update de animaciones
199         if (mixers.length > 0){
200             for (var i = 0; i < mixers.length; i++){
201                 mixers[i].update(delta);
202             }
203         }
204
205
```

Y prácticamente eso es todo para que un FBX cargue en THREE.js. Éste tipo de animaciones son muy útiles para personajes de videojuegos, ya que pasan de una animación estática a una animación caminando, o de caminando a corriendo.

¿Cómo activar dos o más animaciones de un FBX?

Mostraré como se pueden activar dos animaciones, una por default y la otra mediante una tecla.

Primero agregaremos dos variables globales más:

Var KEYS = {}; para cuando debamos detectar las teclas presionadas.

Var flag = false; para la validación de qué animación debe hacer cuando una tecla esté presionada.

```
View Goto Tools Project Preferences Help
index1.html index.html
15     var scene;
16     var renderer;
17     var camera;
18
19     var clock;
20     var delta;
21
22     var mixers = [];
23     var action, action2;
24
25     var KEYS = {};
26     var flag = false;
27
28     $(document).ready(function() {
29         clock = new THREE.Clock();
```

Para las teclas colocaremos el siguiente código dentro de la función document ready, antes de llamar la función de render. Éste código nos ayuda a detectar si las teclas están siendo presionadas.

```
View Goto Tools Project Preferences Help
index1.html index.html
164
165
166     $(document).on("keydown", function(evt){
167         KEYS[String.fromCharCode(evt.which)] = true;
168         // KEYS["A"] = true
169     });
170
171     $(document).on("keyup", function(evt){
172         var key = String.fromCharCode(evt.which);
173         KEYS[key] = false;
174     });
175
176     //Mandamos a llamar la función render
177     render();
178
179 }
180
```

Hay que comentar la parte de código en donde le estamos asignando un peso a cada animación para que no lo tome como default.

```
116
117
118   var loader = new THREE.FBXLoader();
119   loader.load('Modelos/DosAnimaciones3.fbx', function (personaje){
120
121       //AnimationMixer es un reproductor de animaciones
122       personaje.mixer = new THREE.AnimationMixer(personaje);
123
124       //El reproductor de animaciones de nuestro modelo
125       //entra al arreglo mixers
126       mixers.push(personaje.mixer);
127
128       //En esta parte es donde nos sirve el haber preparado el archivo FBX
129       //en Maya, al haberlo exportado como un export de videojuego
130       //internamente el archivo separa las animaciones y así
131       //podemos decir que una es la animación 0 y otra es la animación 1
132       //Si tuvieramos más de dos animacioness, los numeros irían aumentando
133       action = personaje.mixer.clipAction(personaje.animations[0]);
134       action2 = personaje.mixer.clipAction(personaje.animations[1]);
135
136       //Aplicamos play a ambas animaciones
137       action.play();
138       action2.play();
139
140       //Le asignamos peso a una de las animaciones
141       //la de mayor peso es la que se va a reproducir
142       //NOTA: el peso menor es 0 y el mayor es 1
143       //se puede combinar pesos, por ejemplo
144       //ponerle a ambos 0.5, entonces las animaciones
145       //se mezclarían
146
147       //action.weight = 0;
148       //action2.weight = 1;
149
150       //Le damos la posición que queramos en x, y, z
151       personaje.position.z = -10;
152       personaje.position.x = -10;
153       personaje.position.y = -5;
154
155       //Le asignamos una escala en x, y, z
156       personaje.scale.set (0.08, 0.08, 0.08);
157
158       //Le asignamos un nombre al modelo
159       personaje.name = 'Monita';
160
161       //Agregamos el personaje a escena
162       scene.add(personaje);
163
164   });
165
```

Después modificaremos la actualización de la animación en el render, y utilizaremos la variable flag. Si flag es verdadera, reproducirá la animación dos, si flag es falsa reproducirá la animación uno.

```
View Goto Tools Project Preferences Help
index1.html index.html
202
203     function render() {
204
205         //Recibe como parametro la función padre
206         //Se llama varias veces, "update"
207         requestAnimationFrame(render);
208
209         delta = clock.getDelta();
210
211         //Mide el arreglo de mixers, mientras no esté vacío
212         //lo va a recorrer y hará el update de animaciones
213         if (mixers.length > 0){
214             for (var i = 0; i < mixers.length; i++){
215                 mixers[i].update(delta);
216             }
217             if (flag) {
218                 //mixers[0].update(3.9);
219                 action.weight = 0;
220                 action2.weight = 1;
221
222                 flag=false;
223             }
224             else{
225                 action.weight = 1;
226                 action2.weight = 0;
227             }
228         }
229
230         //Recibe como parametro qué escena va a dibujar, y la cámara que se va a utilizar
231         renderer.render( scene, camera );
232
233     }
234
235 }
```

Después de la función render, crearemos una función para cuando se presione una de las teclas. Si, por ejemplo, quieres que tu personaje también avance en el mundo, puedes mandar a llamar tu modelo por el nombre que le diste en un inicio, y dentro de la misma tecla a presionar colocar:

```
if (personajeMueve.position.x > -15) {
    personajeMueve.position.x-= 10*delta; }
```

Éste bloque de código nos dice, que si la posición en x del modelo es mayor a -15, moverá en x al modelo 10 espacios por delta. El if nos sirve para poner límites en el movimiento, en caso de que no quieras que tu modelo salga de cierto cuadro en la pantalla.

```
juego.html
377     //Render
378
379     function moverPersonaje(delta){
380
381         var personajeMueve = scene.getObjectByName('Monita');
382
383         ///JUGADOR 1
384         if (KEYS["A"]) { //a
385
386             flag=true;
387
388             if(personajeMueve.position.x > -15){
389                 personajeMueve.position.x-= 10*delta;
390             }
391         }
392     }
393
394 }
```

Y ahora mandamos a llamar esa función dentro del render.

```
View Goto Tools Project Preferences Help
index1.html index.html
202
203 function render() {
204
205     //Recibe como parametro la función padre
206     //Se llama varias veces, "update"
207     requestAnimationFrame(render);
208
209     delta = clock.getDelta();
210
211     //Mide el arreglo de mixers, mientras no esté vacío
212     //lo va a recorrer y hará el update de animaciones
213     if (mixers.length > 0){
214         for (var i = 0; i < mixers.length; i++){
215             mixers[i].update(delta);
216         }
217         if (flag) {
218             //mixers[0].update(3.9);
219             action.weight = 0;
220             action2.weight = 1;
221
222             flag=false;
223         }
224         else{
225             action.weight = 1;
226             action2.weight = 0;
227         }
228     }
229     moverPersonaje(delta);
230
231     //Recibe como parametro qué escena va a dibujar, y la cámara que se va a utilizar
232     renderer.render( scene, camera );
233
234
235
236 }
```

Y listo, ahora cada vez que se presione la tecla A, flag será verdadero y se reproducirá la segunda animación, así como también el modelo avanzará de posición en x en el entorno. En cambio, cuando se deje de presionar la tecla A, flag será falso y se reproducirá la primera animación, así como el modelo dejará de avanzar.

Puedes agregar más animaciones en tu FBX y aplicar el mismo procedimiento con otras teclas, solo asegúrate de dar el peso correcto a las animaciones que quieras reproducir.

Otros métodos y propiedades de la animación en THREE.js

A parte del método `.play()`; que nos sirve para que se reproduzca la animación, existen algunos otros como `.setDuration(tiempoEnSegundos)` que sirve para hacer que la animación vaya más lento o más rápido. También está `.startAt(tiempoEnSegundos)` que puede funcionar como un delay. Hay una gran cantidad de métodos y propiedades con los que se pueden jugar para crear efectos interesantes en las animaciones. Aquí hay dos páginas de documentación al respecto:

- <https://threejs.org/docs/#api/en/animation/AnimationMixer>
- <https://threejs.org/docs/#api/en/animation/AnimationAction>

Las animaciones en FBX son muy útiles para juegos que necesitan muchas animaciones, como caminar, correr, atacar, brincar... Y es aún más útil que todas éstas animaciones se encuentren en un solo FBX, eso evita que el programa tenga que estar leyendo múltiples archivos, disminuye la cantidad de código por cada archivo y es más fácil hacer las transiciones de una animación a otra.

El dominar el tema solo es cuestión de práctica.