FAQ

Sistemas Gráficos

Grado en Ingeniería Informática

Curso 2019/2020

Pregunta 1 de 2

En el ejercico 6 he cargado el modelo pero cuando lo voy a usar me sale un error de que alguna variable o atributo no está definido.

Respuesta

El problema es que los navegadores cuando cargan archivos del disco, dado que es algo que puede emplear más tiempo, lo hacen en paralelo mientras continúa la ejecución del archivo javascript en curso. Por lo tanto, si en esa ejecución se llega a una línea donde se referencia algo que aún no se ha definido dará dicho error.

Veámoslo con el ejemplo de la carga de un modelo desde disco. Según cómo de alta sea la resolución del modelo se puede llevar un tiempo y sufrir ese problema.

```
1 class Modelo extends THREE.Object3D {
2   constructor() {
3     super();
4
5     var that = this;
6     var materialLoader = new THREE.MTLLoader();
7     var objectLoader = new THREE.OBJLoader();
8     materialLoader.load ('porsche911/911.mtl',
9         function (materials) {
10          objectLoader.setMaterials (materials);
11          objectLoader.load ('porsche911/Porsche_911_GT2.obj',
12            function (object) {
13              that.modelo = object;
14              that.add (that.modelo);
15            }, null, null);
16        });
17    this.modelo.position.y = 5;
18  }
19 }
```

En ese ejemplo, la línea 17 dará un error. El código que empieza en la línea 10 no se va a ejecutar hasta que el archivo de materiales no se haya cargado, y el código que empieza en la línea 13 no se va a ejecutar hasta que el archivo del modelo no se haya cargado.

Mientras se producen esas cargas de archivos el código del constructor ha seguido su ejecución en paralelo y se ejecutará la línea 17 antes de que se haya ejecutado la 13. Por lo tanto tendremos un error del tipo de que no existe `position` para `undefined`.

Por ello, en las transparencias de ese tema os decía, *El modelo se manipula operando con this. Por ejemplo:* `this.position.x = 5;`

Es decir, se está creando un nodo Object3D (this) y en algún momento se le va a colgar como hijo un subgrafo (el modelo cargado) cuya raíz la vamos a referenciar con this.modelo.

Pero precisamente por eso, porque el modelo es hijo de this, podemos aplicar todas las transformaciones que hagan falta a this que el modelo tambien se va a transformar de la misma manera. Vamos, el funcionamiento normal de las transformaciones en un modelo jerárquico o en un grafo de escena.

Cuando el modelo esté cargado, el grafo formado y se renderice se verá transformado tal cual se espera.

Por lo tanto, la solución sería sustituir la línea 17 del ejemplo por esta:

```
this.position.y = 5;
```

Los modelos cargados los manipularemos actuando sobre su nodo padre y no sobre el nodo raíz del subgrafo del modelo.

No obstante os puede surgir la siguiente pregunta.

Pregunta 2 / 2

¿Y no hay una forma de detener la ejecución para esperar a que se cargue el modelo?

Respuesta

Sí. Se puede indicar que un determinado método no se ejecute hasta que otros hayan finalizado.

Veamos el ejemplo de antes modificado.

```
1 class Modelo extends THREE.Object3D {
2   constructor() {
3     super();
4
5     var diferidos = [];
6     diferidos.push(this,cargaDiferidaModelo());
7     $.when.apply(this, diferidos).done( ()=>this.manipularModelo());
8   }
9
10  cargaDiferidaModelo() {
11    var metodoDiferido = $.Deferred();
12
13    var that = this;
14    var materialLoader = new THREE.MTLLoader();
15    var objectLoader = new THREE.OBJLoader();
16    materialLoader.load('porsche911/911.mtl',
17        function(materials) {
18          objectLoader.setMaterials(materials);
19          objectLoader.load('porsche911/Porsche_911_GT2.obj',
20            function(object) {
21              that.modelo = object;
22              that.add(that.modelo);
23              metodoDiferido.resolve();
24            }, null, null);
25        });
26    return metodoDiferido.promise();
27  }
28
29  manipularModelo() {
30    this.modelo.position.y = 5;
31  }
32 }
```

Las instrucciones que implican cargar archivos se han encapsulado en un método, que he llamado `cargaDiferidaModelo()`

En dicho método se define y usa una variable para controlar esa sincronización que andamos buscando.

En la línea 11 se declara y define.

En la línea 26 se devuelve aplicándole el metodo `promise()`.

Y en la línea 23, la última línea de la última función que se ha ejecutado asíncronamente, se llama al método `resolve()` de esa variable de sincronización.

Por otro lado, el código que debe esperar a que finalice el anterior también se ha encapsulado en otro método, que he llamado `manipularModelo()`.

Ya solo nos queda, llamar a `cargaDiferidaModelo()` y decirle a `manipularModelo()` que lo espere.

Son las líneas 5 y 6 del constructor. Donde se crea una lista para almacenar las *promesas* que nos devuelvan los métodos que van a tener ejecuciones asíncronas. Método que está siendo llamado en la propia línea 6.

Y la línea 7, donde le decimos que cuando se *resuelvan* las *promesas* de la lista, se ejecute el método que tiene como parámetro.

En definitiva, this.manipularModelo() no se va a ejecutar hasta que se ejecute la línea 23 y se resuelva la promesa que se hizo en la línea 26.

Último comentario

Para el caso concreto del ejercicio 6, veo mejor la primera solución. Esta segunda explicación la he dado porque en la práctica 2 siempre hay gente que necesita algún tipo de sincronización de este tipo. Que algún método espere a que se cargen texturas, modelos, etc. y ya he aprovechado una consulta que me han hecho para explicaros esto.