EBook Gratis

APRENDIZAJE aframe

Free unaffiliated eBook created from
Stack Overflow contributors.

#aframe

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: aframe

It is an unofficial and free aframe ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official aframe.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con aframe

Observaciones

Esta sección proporciona una descripción general de qué es un marco y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de un marco, y vincular a los temas relacionados. Dado que la Documentación para aframe es nueva, es posible que deba crear versiones iniciales de los temas relacionados.

Versiones

A-Frame 0.x

Versión	Fecha de lanzamiento
0.6	2017-05-25
0.5	2017-02-10
0.4	2016-12-17
0.3	2016-08-18

Versiones heredadas

Versión	Fecha de lanzamiento
0.2	2016-03-26
0.1	2015-12-17

Examples

Empezando

¡A-Frame puede desarrollarse a partir de un archivo HTML simple sin tener que instalar nada! Una excelente manera de probar A-Frame para volver a mezclar el ejemplo de inicio en Glitch, un editor de código en línea que instantáneamente aloja y despliega de forma gratuita. O cree un archivo `.html` e incluya A-Frame en la `head` :

```
<html>
```

```
  <head>
    <script src="https://aframe.io/releases/0.5.0/aframe.min.js"></script>
  </head>
  <body>
    <a-scene>
      <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-box>
      <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
      <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#FFC65D"></a-cylinder>
      <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4" color="#7BC8A4"></a-
plane>
      <a-sky color="#ECECEC"></a-sky>
    </a-scene>
  </body>
</html>
```

Incluir la compilación de JS

Para incluir A-Frame en un archivo HTML, soltamos una etiqueta de `script` apunta a la compilación de CDN:

```
<head>
  <script src="https://aframe.io/releases/0.5.0/aframe.min.js"></script>
</head>
```

Instalar desde npm

También podemos instalar A-Frame a través de npm:

```
$ npm install aframe
```

Entonces podemos incluir A-Frame en nuestra aplicación. Por ejemplo, con Browserify o Webpack:

```
require('aframe');
```

Si usa npm, puede usar angle, una interfaz de línea de comandos para A-Frame. ángulo puede inicializar una plantilla de escena con un solo comando:

```
npm install -g angle && angle initscene
```

Caracteristicas

VR Made Simple

Solo suelta una etiqueta de `script` y `a-scene` . A-Frame manejará la placa de calderas 3D, la configuración de VR y los controles predeterminados. Nada para instalar, no hay pasos de

compilación.

HTML declarativo

HTML es fácil de leer, entender y copiar y pegar. Al estar basado en HTML, A-Frame es accesible para todos: desarrolladores web, entusiastas de la realidad virtual, artistas, diseñadores, educadores, creadores, niños.

VR multiplataforma

Cree aplicaciones de VR para Vive, Rift, Daydream, GearVR y Cardboard con soporte para todos los controladores respectivos. ¿No tienes auriculares o controladores? ¡No hay problema! A-Frame todavía funciona en computadoras de escritorio y teléfonos inteligentes estándar.

Arquitectura Entidad-Componente

A-Frame es un marco poderoso de three.js, que proporciona una estructura de componentes de entidad declarativa, compostable y reutilizable. HTML es solo la punta del iceberg; los desarrolladores tienen acceso ilimitado a JavaScript, API de DOM, three.js, WebVR y WebGL.

Actuación

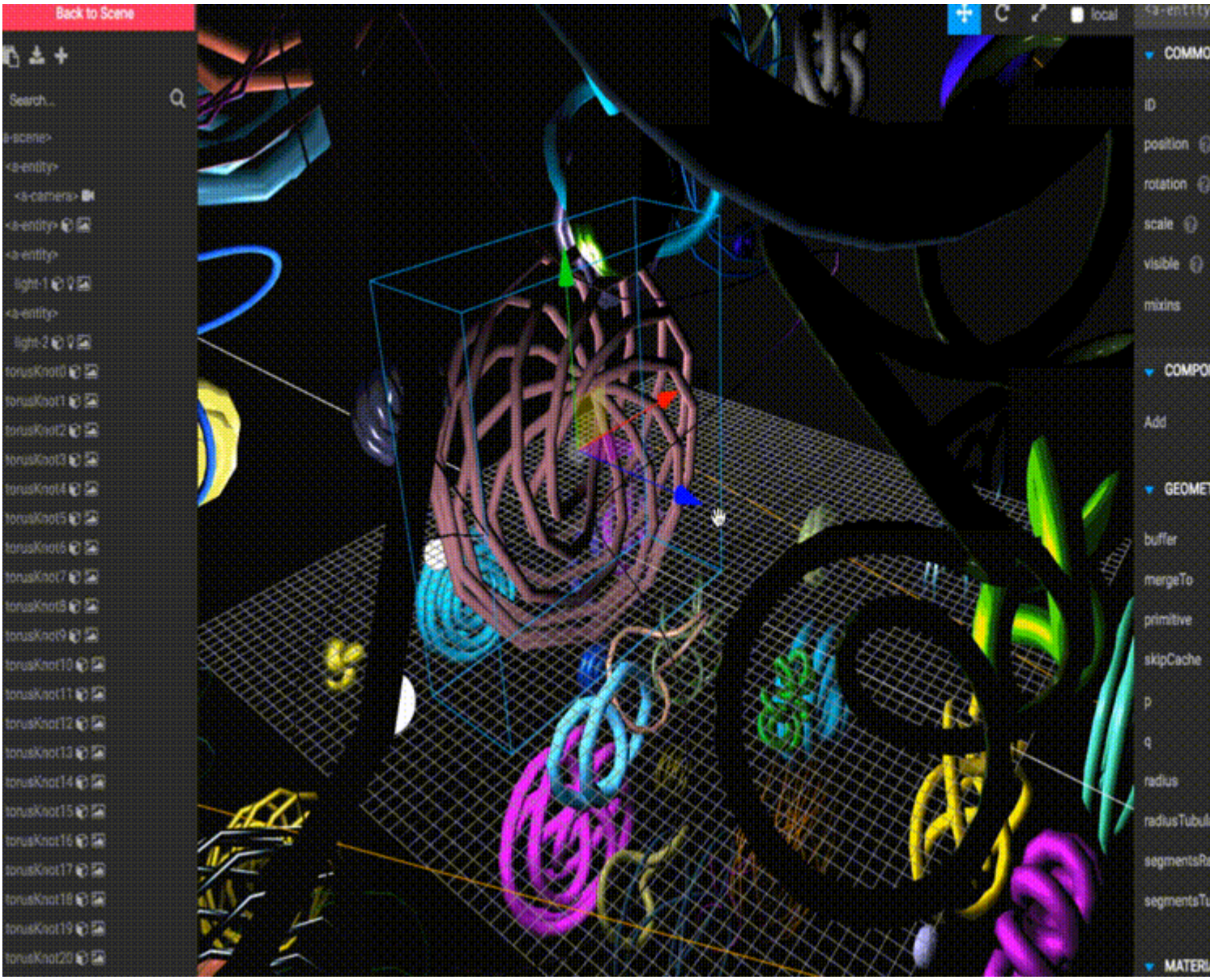
A-Frame está optimizado desde cero para WebVR. Mientras que A-Frame usa el DOM, sus elementos no tocan el motor de diseño del navegador. Todas las actualizaciones de objetos 3D se realizan en memoria con poca sobrecarga en una sola llamada requestAnimationFrame. Para referencia, vea A-Painter, un clon de Pincel de Inclinación construido en A-Frame que se ejecuta como nativo (90+ FPS).

Herramienta agnóstica

Dado que la Web se creó con la noción de HTML, A-Frame es compatible con la mayoría de las bibliotecas, marcos y herramientas, incluyendo React, Preact, Vue.js, Angular, d3.js, Ember.js, jQuery.

Inspector visual

A-Frame proporciona un práctico inspector visual 3D incorporado. Abre cualquier escena de A-Frame, `pulsa ctrl` + `alt` + `i` , y vuela para mirar detrás del capó.



Registro

Tome los componentes poderosos que los desarrolladores han publicado y conéctelos directamente desde HTML. Al igual que en la Tienda de Activos de Unity, el Registro A-Frame recopila y cura estos componentes para un fácil descubrimiento.

Componentes

Empiece a correr con los componentes principales de A-Frame, como geometrías, materiales, luces, animaciones, modelos, difusores de rayos, sombras, audio de posición, texto y controles Vive / Touch / Daydream / GearVR / Cardboard. Vaya aún más lejos con los componentes de la comunidad como sistemas de partículas, física, multiusuario, océanos, montañas, reconocimiento de voz, captura de movimiento, teletransportación, súper manos y realidad aumentada.

Empezando para AR

Para crear aplicaciones de AR en la web, debe agregar una nueva biblioteca llamada AR.js. Primero carga un cuadro A seguido de AR.js.

Newt, debe configurar su escena utilizando la etiqueta A `a-scene` con el `artoolkit` artoolkit agregado. El `sourceType` debe ser tu webcam. La fuente de la cámara de su teléfono inteligente también es compatible con esto.

El `a-marker-camera` marker marca una imagen dentro de la pantalla grabada que representa una imagen. En este caso es `marker.png` . Cuando la cámara detecte este marcador, el cuadro se mostrará en el marcador.

A continuación puede encontrar el código de ejemplo:

```
<script src="https://aframe.io/releases/0.5.0/aframe.min.js"></script>
<script src="https://rawgit.com/jeromeetienne/ar.js/master/aframe/build/aframe-
ar.js"></script>
<script>
    THREEx.ArToolkitContext.baseURL =
'https://rawgit.com/jeromeetienne/ar.js/master/three.js/'
</script>
<body>

    <a-scene artoolkit='sourceType: webcam;'>
        <a-box position='0 0 0.5' material='opacity: 0.5;'></a-box>
        <a-marker-camera preset='marker.png'></a-marker-camera>
    </a-scene>

</body>
```

Lea Empezando con aframe en línea: https://riptutorial.com/es/aframe/topic/10017/empezando-con-aframe

Capítulo 2: Animación

Introducción

Las animaciones y transiciones en A-Frame se definen utilizando el elemento `<a-animation>` como elemento secundario. El sistema se basa aproximadamente en la especificación de animaciones web. A-Frame utiliza tween.js internamente.

Observaciones

Atributos

Aquí hay una visión general de los atributos de animación. Vamos a entrar en más detalle a continuación.

Atributo	Descripción	Valor por defecto
atributo	Atributo para animar. Para especificar un atributo de componente, use la sintaxis `componentName.property` (por ejemplo, `light.intensity`).	rotación
empezar	Nombre del evento a esperar antes de comenzar la animación.	"
retrasar	Retardo (en milisegundos) o nombre del evento para esperar antes de comenzar la animación.	0
dirección	Dirección de la animación (entre `from` y `to`). Uno de `alternate` , `alternateReverse` , `reverse` , `normal` , `reverse` .	normal
dur	Duración en (milisegundos) de la animación.	1000
facilitando	Función facilitadora de la animación. Hay muchos para elegir.	facilitar
fin	Nombre del evento a esperar antes de detener la animación.	"
llenar	Determina el efecto de la animación cuando no está activo en el juego. Uno de `backwards` , `both` , `forwards` , `none` .	hacia adelante
desde	Valor inicial.	Valor actual.
repetir	Repita el conteo o `indefinite` .	0
a	Valor final. Debe ser especificado.	Ninguna

Empezar

El atributo de `begin` define cuando la animación debe comenzar a reproducirse.

Puede ser un *número* , que representa milisegundos que esperar o un *nombre de evento* que esperar. Por ejemplo, podemos definir una animación que espere 2 segundos antes de escalar una entidad.

```
<a-entity>
  <a-animation attribute="scale" begin="2000" to="2 2 2"></a-animation>
</a-entity>
```

O podemos definir una animación que espere a que el elemento padre active un evento llamado `fade` antes de desvanecer una entidad.

```
<a-entity id="fading-cube" geometry="primitive: box" material="opacity: 1">
  <a-animation attribute="material.opacity" begin="fade" to="0"></a-animation>
</a-entity>
```

```
// Trigger an event to begin fading.
document.querySelector('#fading-cube').emit('fade');
```

Dirección

El atributo de `direction` define la forma de animar entre el valor inicial y el valor final.

Cuando definimos una dirección alterna, la animación irá de un lado a otro entre los valores `from` y `to` como un yo-yo. Las direcciones alternas solo tienen efecto cuando repetimos la animación.

Valor	Descripción
alterno	En los ciclos de números pares, anime `from` a `to` . En los ciclos impares, la animación de `to` a `from`
alterno-reverso	En los ciclos con números impares, anime `from` a `to` . En ciclos de números pares, animación de `to` a `from`
normal	Animar a partir `from` a `to` .
marcha atrás	Animar de `to` a `from` .

Facilitando

El atributo de `easing` define la función de suavizado de la animación, que por defecto `ease` . Hay demasiadas funciones de aceleración para enumerar, pero podemos explicarlas implícitamente.

Un valor posible es `linear` . Y las funciones básicas de aceleración son `ease` , `ease-in` , `ease-out` y `ease-in-out`

.

Luego hay más grupos de funciones de flexibilización. Las funciones de suavizado básicas anteriores prefijan cada grupo de funciones de suavizado. Los grupos de funciones de aceleración son `cubic` , `quad` , `quart` , `quint` , `sine` , `expo` , `circ` , `elastic` , `back` y `bounce` .

Por ejemplo, el grupo `cubic` de funciones de aceleración consistiría `ease-cubic` `ease-in-cubic` , `ease-cubic` `ease-in-cubic` , `ease-in-cubic` , `ease-out-cubic` `ease-in-out-cubic` .

Llenar

El atributo de `fill` define el efecto de la animación cuando no está activo en el juego. Piense en el `fill` como qué valores establece la animación en la entidad *antes* y / o *después de* cada ciclo de animación. A continuación se muestran los posibles valores de `fill` y sus efectos.

Valor	Descripción
hacia atrás	Antes comienza la animación, establezca el valor de salida a la `from` valor.
ambos	Combina los efectos de relleno hacia atrás y relleno hacia adelante.
hacia adelante	Una vez finalizada la animación, el valor final se quedará en el `to` valorar. El relleno por defecto.
ninguna	Antes de que comience la animación, establezca el valor inicial en el valor inicial. Una vez finalizada la animación, restablecer el valor al valor inicial.

Repetir

El atributo de `repeat` define con qué frecuencia se repite la animación. Llamamos a cada repetición de la animación un ciclo. Repetir puede ser un número que realiza una cuenta regresiva en cada ciclo de animación hasta que llegue a `0` punto en el que finalizará la animación, o podemos configurar la `repeat` a `indefinite` y la animación se repetirá continuamente hasta que la animación se elimine o se detenga manualmente.

EVENTOS

El elemento `<a-animation>` emite un par de eventos.

Nombre del evento	Descripción
animación	Se emite cuando finaliza la animación. En caso de repeticiones, se emite cuando el recuento de repeticiones llega a `0` . No emitido por repeticiones indefinidas.

Nombre del evento	Descripción
animaciónstart	Se emite inmediatamente cuando la animación comienza a reproducirse.

Examples

Animaciones de ejemplo

Como ejemplo introductorio, para definir una órbita de 5 metros en una entidad sobre el eje Y que toma 10 segundos, podemos compensar la posición y animar la rotación. Esta animación comienza con la rotación inicial sobre el eje Y de 0 grados y gira alrededor de 360 grados. Se define con una duración de 10000 milisegundos, mantiene el valor final en cada ciclo de la animación y se repite infinitamente.

```
<a-entity position="5 0 0" rotation="0 0 0">
  <a-animation attribute="rotation"
               to="0 360 0"
               dur="10000"
               fill="forwards"
               repeat="indefinite"></a-animation>
</a-entity>
```

Animando diferentes tipos de propiedades

El sistema de animación de A-Frame puede animar diferentes tipos de propiedades.

Propiedades de vec3

A-Frame tiene componentes `vec3` estándar (es decir, `position` , `rotation` y `scale`). Estos componentes constan de tres factores: X, Y y Z. Podemos pasar tres números delimitados por espacios a los atributos `from` y `to` tal como los definiríamos en una entidad. En este caso, el sistema de animación asumirá que estamos animando un valor `vec3` .

Por ejemplo, si queremos animar una entidad que va de un lugar a otro, podemos animar el componente de `position` .

```
<a-entity>
  <a-animation attribute="position" from="1 1 1" to="2 4 -8"></a-animation>
</a-entity>
```

Propiedades booleanas

A-Frame tiene componentes estándar que aceptan un solo valor booleano. Los valores booleanos también pueden ser "animados" al voltear el booleano al final de cada ciclo de animación.

Por ejemplo, podemos definir una animación que desactiva la visibilidad de una entidad después de 5 segundos.

```
<a-entity>
  <a-animation attribute="visible" dur="5000" to="false" repeat="indefinite"></a-animation>
</a-entity>
```

Propiedades numéricas

También podemos animar atributos numéricos. Por ejemplo, podemos animar la intensidad de la luz primitiva.

```
<a-light intensity="1">
  <a-animation attribute="intensity" to="3"></a-animation>
</a-light>
```

Propiedades de color

Podemos animar cualquier propiedad componente que tenga un tipo de color. Por ejemplo, podemos animar un cuadro de blanco a rojo.

```
<a-entity id="blushing-cube" geometry="primitive: box">
  <a-animation attribute="material.color" from="white" to="red" dur="1000"></a-animation>
</a-entity>
```

Propiedades de los componentes

Podemos animar una determinada propiedad de un componente multi-propiedad. Para hacerlo, seleccionamos la propiedad del componente usando la sintaxis del punto: `componentName.propertyName` .

Por ejemplo, para animar el radio superior de un cono, podemos seleccionar el valor de `radiusTop` con `geometry.radiusTop` .

```
<a-entity geometry="primitive: cone; radiusTop: 1">
  <a-animation attribute="geometry.radiusTop" to="0.5"></a-animation>
</a-entity>
```

Lea Animación en línea: https://riptutorial.com/es/aframe/topic/10071/animacion--a-animation-

Capítulo 3: Cámara

Introducción

El componente de la cámara define desde qué perspectiva el usuario ve la escena. La cámara generalmente se combina con componentes de controles que permiten que los dispositivos de entrada muevan y giren la cámara.

Sintaxis

- <una cámara de entidad> </a-entity>
- <a-camera> </a-camera>

Parámetros

Propiedad	Descripción
activo	Si la cámara es la cámara activa en una escena con más de una cámara.
lejos	Cámara frustum lejos plano de recorte.
fov	Campo de visión (en grados).
cerca	Cámara frustum cerca de plano de recorte.
usuario altura	Cuánta altura agregar a la cámara cuando no está en modo VR. La cámara predeterminada tiene este ajuste en 1,6 (metros, para representar el nivel promedio de los ojos).
enfocar	Factor de zoom de la cámara.

Observaciones

Cuando no está en modo VR, `userHeight` traduce la cámara a una altura promedio aproximada del nivel del ojo humano. La cámara inyectada tiene este ajuste a 1,6 (metros). Al ingresar a VR, este desplazamiento de altura se *elimina de* manera que usamos la posición absoluta devuelta por el auricular VR. El desplazamiento es conveniente para las experiencias que funcionan tanto dentro como fuera de la realidad virtual, así como para hacer que las experiencias se vean decentes desde una pantalla de escritorio en lugar de recortar el suelo si el auricular descansaba en el suelo.

Al salir de la realidad virtual, la cámara restablecerá su rotación antes de ingresar a la realidad virtual. Esto es así cuando salimos de VR, la rotación de la cámara vuelve a la normalidad para una pantalla de escritorio.

Examples

Cámara por defecto

Una cámara situada a la altura promedio del nivel del ojo humano (1.6 metros o 1.75 yardas o 5.25 pies).

```
<a-entity camera="userHeight: 1.6" look-controls></a-entity>
```

Cambiando la cámara activa

Cuando la propiedad activa se conmuta, el componente notificará al sistema de la cámara para cambiar la cámara actual utilizada por el renderizador:

```
var secondCameraEl = document.querySelector('#second-camera');
secondCameraEl.setAttribute('camera', 'active', true);
```

Fijación de entidades a la cámara

Para colocar las entidades en la cámara de modo que permanezcan a la vista sin importar el aspecto del usuario, puede adjuntar esas entidades como un elemento secundario de la cámara. Los casos de uso pueden ser una pantalla de mano a mano (HUD).

```
<a-entity camera look-controls>
  <a-entity geometry="primitive: plane; height: 0.2; width: 0.2" position="0 0 -1"
            material="color: gray; opacity: 0.5"></a-entity>
</a-entity>
```

Tenga en cuenta que debe usar los HUD con moderación, ya que causan irritación y fatiga ocular en la realidad virtual. Considera integrar los menús en el tejido del mundo mismo. Si crea un HUD, asegúrese de que el HUD esté más en el centro del campo de visión, de modo que el usuario no tenga que forzar la vista para leerlo.

una cámara primitiva

La cámara primitiva determina lo que ve el usuario. Podemos cambiar la ventana gráfica modificando la posición y la rotación de la entidad de la cámara.

Tenga en cuenta que, de forma predeterminada, el origen de la cámara estará en 0 1.6 0 en modo escritorio y 0 0 0 en modo VR. Lea acerca de la propiedad camera.userHeight.

```
<a-scene>
  <a-box></a-box>
  <a-camera></a-camera>
</a-scene>
```

Posicionamiento manual de la cámara

Para colocar la cámara, coloque la posición en una envoltura. No establezca la posición directamente en la cámara primitiva porque los controles anularán rápidamente la posición establecida:

```
<a-entity position="0 0 5">
  <a-camera></a-camera>
</a-entity>
```

Lea Cámara en línea: https://riptutorial.com/es/aframe/topic/10181/camara

Capítulo 4: Componentes

Introducción

En el patrón entidad-componente-sistema, un componente es una porción de datos reutilizable y modular que conectamos en una entidad para agregar apariencia, comportamiento y / o funcionalidad.

En A-Frame, los componentes modifican las entidades que son objetos 3D en la escena. Mezclamos y componemos componentes para construir objetos complejos. Nos permiten encapsular el código three.js y JavaScript en módulos que podemos usar declarativamente desde HTML. Los componentes son aproximadamente análogos a CSS.

Observaciones

Definición de los métodos de manejo del ciclo de vida

Con el esquema siendo la anatomía, los métodos del ciclo de vida son la fisiología; el esquema define la forma de los datos, los métodos del manejador del ciclo de vida *usan* los datos para modificar la entidad. Los manejadores generalmente interactuarán con la **API de** la **entidad** .

Resumen de los métodos

Método	Descripción
en eso	Se llama una vez cuando se inicializa el componente. Se utiliza para configurar el estado inicial y crear instancias de las variables.
actualizar	Se llama tanto cuando el componente se inicializa y cada vez que se actualiza alguna de las propiedades del componente (por ejemplo, a través de *setAttribute*). Se utiliza para modificar la entidad.
retirar	Se llama cuando el componente se elimina de la entidad (por ejemplo, a través de *removeAttribute*) o cuando la entidad se separa de la escena. Se utiliza para deshacer todas las modificaciones anteriores a la entidad.
garrapata	Llamado en cada bucle de render o tick de la escena. Se utiliza para cambios continuos o cheques.
jugar	Se llama cada vez que se reproduce la escena o entidad para agregar cualquier fondo o comportamiento dinámico. También se llama una vez cuando se inicializa el componente. Se utiliza para iniciar o reanudar el comportamiento.
pausa	Se llama cuando la escena o entidad se detiene para eliminar cualquier fondo o comportamiento dinámico. También se llama cuando el

Método	Descripción
	componente se elimina de la entidad o cuando la entidad se separa de la escena. Se utiliza para pausar el comportamiento.
updateSchema	Se llama cada vez que se actualiza alguna de las propiedades del componente. Se puede utilizar para modificar dinámicamente el esquema.

Propiedades de prototipo de componente

Dentro de los métodos, tenemos acceso al prototipo del componente a través de `this` :

Propiedad	Descripción
este.data	Las propiedades del componente analizadas se calculan a partir de los valores predeterminados del esquema, los mixins y los atributos de la entidad.
esto.el	Referencia a la [entidad] [entidad] como un elemento HTML.
esto.el.sceneEl	Referencia a la [escena] [escena] como un elemento HTML.
esto.id	Si el componente puede tener [varias instancias] [múltiples], el ID de la instancia individual del componente (por ejemplo, `foo` from `sound__foo`).

Metodos

.en eso ()

`.init ()` se llama una vez al principio del ciclo de vida del componente. Una entidad puede llamar al controlador de `init` del componente:

- Cuando el componente se establece estáticamente en la entidad en el archivo HTML y la página se carga.
- Cuando el componente se establece en una entidad adjunta a través de `setAttribute` .
- Cuando el componente se establece en una entidad no adjunta, y la entidad se adjunta a la escena a través de `appendChild` .

El controlador de `init` se suele utilizar para:

- Configurar el estado inicial y las variables.
- Métodos de unión
- Adjuntar oyentes de eventos

Por ejemplo, la `init` un componente del cursor establecería variables de estado, métodos de enlace y agregaría escuchas de eventos:

```
AFRAME.registerComponent('cursor', {
  // ...
  init: function () {
    // Set up initial state and variables.
    this.intersection = null;
    // Bind methods.
    this.onIntersection = AFRAME.utils.bind(this.onIntersection, this);
    // Attach event listener.
    this.el.addEventListener('raycaster-intersection', this.onIntersection);
  }
  // ...
});
```

.update (oldData)

`.update (oldData)` cada vez que cambian las propiedades del componente, incluso al comienzo del ciclo de vida del componente. Una entidad puede llamar al controlador de `update` un componente:

- Después de llamar a `init ()` , al comienzo del ciclo de vida del componente.
- Cuando las propiedades del componente se actualizan con `.setAttribute` .

El controlador de `update` se utiliza a menudo para:

- Haga la mayor parte del trabajo para hacer modificaciones a la entidad, utilizando `this.data` .
- Modifique la entidad siempre que cambien una o más propiedades del componente.

Las modificaciones granulares de la entidad pueden realizarse [difingir] [dif] el conjunto de datos actual (`this.data`) con el conjunto de datos anterior antes de la actualización (`oldData`).

A-Frame llama a `.update()` tanto al principio del ciclo de vida de un componente como cada vez que los datos de un componente cambian (por ejemplo, como resultado de `setAttribute`). El controlador de actualización a menudo utiliza `this.data` para modificar la entidad. El controlador de actualización tiene acceso al estado anterior de los datos de un componente a través de su primer argumento. Podemos usar los datos anteriores de un componente para indicar exactamente qué propiedades se modificaron para hacer actualizaciones granulares.

Por ejemplo, la `update` del componente **visible** establece la visibilidad de la entidad.

```
AFRAME.registerComponent('visible', {
  /**
   * this.el is the entity element.
   * this.el.object3D is the three.js object of the entity.
   * this.data is the component's property or properties.
   */
  update: function (oldData) {
    this.el.object3D.visible = this.data;
  }
  // ...
});
```

.retirar ()

`.remove ()` se llama cuando el componente se separa de la entidad. Una entidad puede llamar al controlador de `remove` un componente:

- Cuando el componente se elimina de la entidad a través de `removeAttribute` .
- Cuando la entidad se separa de la escena (por ejemplo, `removeChild`).

El controlador de `remove` se utiliza a menudo para:

- Eliminar, deshacer o limpiar todas las modificaciones del componente a la entidad.
- Separar a los oyentes del evento.

Por ejemplo, cuando se elimina el [componente de luz] [luz], el componente de luz eliminará el objeto de luz que había colocado previamente en la entidad, eliminándolo así de la escena.

```
AFRAME.registerComponent('light', {
  // ...
  remove: function () {
    this.el.removeObject3D('light');
  }
  // ...
});
```

.tick (time, timeDelta)

Se llama a `.tick ()` en cada tic o trama del bucle de renderizado de la escena. La escena llamará al manejador de `tick` un componente:

- En cada cuadro del bucle de render.
- Del orden de 60 a 120 veces por segundo.
- Si la entidad o escena no está en pausa (por ejemplo, el Inspector está abierto).
- Si la entidad todavía está vinculada a la escena.

El controlador de `tick` se utiliza a menudo para:

- Modificar continuamente la entidad en cada marco o en un intervalo.
- Encuesta de condiciones.

El controlador de `tick` se proporciona el tiempo de actividad global de la escena en milisegundos (`time`) y la diferencia de tiempo en milisegundos desde el último fotograma (`timeDelta`). Se pueden usar para interpolación o solo para ejecutar partes del controlador de `tick` en un intervalo establecido.

Por ejemplo, el **componente de controles rastreados** progresará las animaciones del controlador, actualizará la posición y la rotación del controlador y verificará si se presionan los botones.

```
AFRAME.registerComponent('tracked-controls', {
  // ...
  tick: function (time, timeDelta) {
    this.updateMeshAnimation();
    this.updatePose();
```

```
    this.updateButtons();
  }
  // ...
});
```

.pause ()

`.pause ()` cuando la entidad o escena se detiene. La entidad puede llamar al controlador de `pause` un componente:

- Antes de eliminar el componente, antes de llamar al controlador de `remove` .
- Cuando la entidad está en pausa con `Entity.pause ()` .
- Cuando la escena está en pausa con `Scene.pause ()` (por ejemplo, se abre el Inspector).

El controlador de `pause` se utiliza a menudo para:

- Eliminar oyentes de eventos.
- Eliminar cualquier posibilidad de comportamiento dinámico.

Por ejemplo, el **componente de sonido** pausará el sonido y eliminará un detector de eventos que haya reproducido un sonido en un evento:

```
AFRAME.registerComponent('sound', {
  // ...
  pause: function () {
    this.pauseSound();
    this.removeEventListener();
  }
  // ...
});
```

.jugar ()

`.play ()` cuando se reanuda la entidad o la escena. La entidad puede llamar al controlador de `play` un componente:

- Cuando el componente se adjunta por primera vez, después de llamar al controlador de `update` .
- Cuando la entidad se detuvo, pero luego se reanudó con `Entity.play ()` .
- Cuando la escena se detuvo, pero luego se reanudó con `Scene.play ()` .

El controlador de `play` suele utilizarse para:

- Añadir oyentes de eventos.

Por ejemplo, el **componente de sonido** reproducirá el sonido y actualizará el detector de eventos que reproducirá un sonido en un evento:

```
AFRAME.registerComponent('sound', {
  // ...
```

```
  play: function () {
    if (this.data.autoplay) { this.playSound(); }
    this.updateEventListener();
  }
  // ...
});
```

.updateSchema (datos)

`.updateSchema ()` , si está definido, se llama en cada actualización para verificar si el esquema necesita ser modificado dinámicamente.

El controlador `updateSchema` se usa a menudo para:

- Actualice o extienda dinámicamente el esquema, generalmente dependiendo del valor de una propiedad.

Por ejemplo, el **componente de geometría** verifica si la propiedad `primitive` cambió para determinar si actualizar el esquema para un tipo diferente de geometría:

```
AFRAME.registerComponent('geometry', {
  // ...
  updateSchema: (newData) {
    if (newData.primitive !== this.data.primitive) {
      this.extendSchema(GEOMETRIES[newData.primitive].schema);
    }
  }
  // ...
});
```

MÉTODOS DE PROTOTIPO DE COMPONENTES

.flushToDOM ()

Para ahorrar tiempo de CPU en la cadena de caracteres, A-Frame solo actualizará en modo de depuración la representación serializada del componente en el DOM real. Al llamar a flushToDOM () se serializarán manualmente los datos del componente y se actualizará el DOM:

```
document.querySelector('[geometry]').components.geometry.flushToDOM();
```

Examples

Registrar un componente A-Frame personalizado

AFRAME.registerComponent (nombre, definición)

Registrar un componente A-Frame. Debemos registrar los componentes antes de usarlos en cualquier parte. . Significado de un archivo HTML, los componentes deben venir en orden antes .

- **{string} name** - Nombre del componente. La API pública del componente como se representa a través de un nombre de atributo HTML.
- **Definición de {objeto}** - Definición de componente. Contiene esquemas y métodos de manejo del ciclo de vida.

Registrar componente en foo en su archivo js, por ejemplo, foo-component.js

```
AFRAME.registerComponent('foo', {
  schema: {},
  init: function () {},
  update: function () {},
  tick: function () {},
  remove: function () {},
  pause: function () {},
  play: function () {}
});
```

Uso del componente foo en tu escena.

```
<html>
  <head>
    <script src="aframe.min.js"></script>
    <script src="foo-component.js"></script>
  </head>
  <body>
    <a-scene>
      <a-entity foo></a-entity>
    </a-scene>
  </body>
</html>
```

Componente HTML Formulario

Un componente contiene un grupo de datos en forma de una o más propiedades del componente. Los componentes utilizan estos datos para modificar entidades. Considere un componente del motor, podríamos definir propiedades tales como caballos de fuerza o cilindros.

Los atributos HTML representan nombres de componentes y el valor de esos atributos representa datos de componentes.

Componente de propiedad única

Si un componente es un componente de una sola propiedad, lo que significa que sus datos consisten en un solo valor, en HTML, el valor del componente parece un atributo HTML normal:

```
<!-- `position` is the name of the position component. -->
<!-- `1 2 3` is the data of the position component. -->
<a-entity position="1 2 3"></a-entity>
```

Componente Multi-propiedad

Si un componente es un componente de múltiples propiedades, lo que significa que los datos se componen de múltiples propiedades y valores, entonces en HTML, el valor del componente se parece a los estilos de CSS en línea:

```
<!-- `light` is the name of the light component. -->
<!-- The `type` property of the light is set to `point`. -->
<!-- The `color` property of the light is set to `crimson`. -->
<a-entity light="type: point; color: crimson"></a-entity>
```

Definición de objeto de esquema compnent

El esquema es un objeto que define y describe la propiedad o propiedades del componente. Las claves del esquema son los nombres de la propiedad, y los valores del esquema definen los tipos y valores de la propiedad (en el caso de un componente de múltiples propiedades):

Definiendo esquema en tu componente

```
AFRAME.registerComponent('bar', {
  schema: {
    color: {default: '#FFF'},
    size: {type: 'int', default: 5}
  }
}
```

Anular los valores predeterminados del esquema definido

```
<a-scene>
  <a-entity bar="color: red; size: 20"></a-entity>
</a-scene>
```

Esquema de propiedad única

Un componente puede ser un componente de una sola propiedad (que consiste en un valor anónimo) o un componente de múltiples propiedades (que consta de múltiples valores con nombre). A-Frame deducirá si un componente es de una sola propiedad o de múltiples propiedades en función de la estructura del esquema.

El esquema de un componente de una sola propiedad contiene claves de `type` y / o `default` , y los valores del esquema son valores simples en lugar de objetos:

```
AFRAME.registerComponent('foo', {
  schema: {type: 'int', default: 5}
});
```

```
<a-scene>
  <a-entity foo="20"></a-entity>
</a-scene>
```

Tipos de propiedad del esquema de componentes de A-Frame

Los tipos de propiedad definen principalmente cómo el esquema analiza los datos entrantes del DOM para cada propiedad. Los datos analizados estarán disponibles a través de la propiedad de `data` en el prototipo del componente. A continuación se muestran los tipos de propiedad incorporados de A-Frame:

tipo de propiedad	Descripción	Valor por defecto
formación	Analiza los valores separados por comas en la matriz (es decir, `"1, 2, 3" to ['1', '2', '3'])` .	[]
activo	Para direcciones URL que apuntan a activos generales. Puede analizar la URL de una cadena en forma de `url(<url>)` . Si el valor es un selector de ID de elemento (por ejemplo, `#texture`), este tipo de propiedad llamará a `getElementById` y `getAttribute('src')` para devolver una URL. El tipo de propiedad del `asset` puede o no cambiar para manejar XHR o devolver MediaElements directamente (por ejemplo, `<img>` elementos).	''
audio	El mismo análisis que el tipo de propiedad del `asset` . Posiblemente será utilizado por el inspector de fotogramas A para presentar activos de audio.	''
booleano	Analiza la cadena a booleano (es decir, `"false"` a falso, todo lo demás es verdadero).	falso
color	Actualmente no hace ningún análisis. Utilizado principalmente por el inspector de fotograma A para presentar un selector de color. Además, es necesario utilizar el tipo de color para que funcionen las animaciones en color.	#FFF
En t	Llama a `parseInt` (por ejemplo, `"124.5"` a `124`).	0
mapa	El mismo análisis que el tipo de propiedad del `asset` . Posiblemente se utilizará por el inspector de fotograma A para	''

tipo de propiedad	Descripción	Valor por defecto
	presentar activos de textura.	
modelo	El mismo análisis que el tipo de propiedad del `asset` . Posiblemente se usará por el inspector de fotograma A para presentar los activos del modelo.	''
número	Llama a `parseFloat` (por ejemplo, `'124.5'` a `'124.5'`).	0
selector	Llama a `querySelector` (por ejemplo, `"#box"` a `<a-entity id="box">`).	nulo
selectorTodo	Llama a `querySelectorAll` y convierte `NodeList` a `Array` (por ejemplo, `".boxes"` a [<a-entity class = "boxes", ...]),	nulo
cuerda	No hace ningún análisis.	''
vec2	Analiza dos números en un objeto `{x, y}` (por ejemplo, `1 -2` a `{x: 1, y: -2}` .	{x: 0, y: 0}
vec3	Analiza tres números en un objeto `{x, y, z}` (por ejemplo, `1 -2 3` a `{x: 1, y: -2, z: 3}` .	{x: 0, y: 0, z: 0}
vec4	Analiza cuatro números en un objeto `{x, y, z, w}` (p. Ej., `1 -2 3 -4.5` a `{x: 1, y: -2, z: 3, w: -4.5}` .	{x: 0, y: 0, z: 0, w: 0}

Tipo de propiedad Inferencia

El esquema intentará inferir un tipo de propiedad dado solo un valor predeterminado:

```
schema: {default: 10}  // type: "number"
schema: {default: "foo"}  // type: "string"
schema: {default: [1, 2, 3]}  // type: "array"
```

El esquema establecerá un valor predeterminado si no se proporciona, dado el tipo de propiedad:

```
schema: {type: 'number'}  // default: 0
schema: {type: 'string'}  // default: ''
schema: {type: 'vec3'}  // default: {x: 0, y: 0, z: 0}
```

Tipo de propiedad personalizada

También podemos definir nuestro propio tipo de propiedad o analizador proporcionando una función de `parse` en lugar de un `type` :

```
schema: {
  // Parse slash-delimited string to an array
```

```
  // (e.g., `foo="myProperty: a/b"` to `['a', 'b']`).
  myProperty: {
    default: [],
    parse: function (value) {
      return value.split('/');
    }
  }
}
```

Acceso a los miembros y métodos de un componente

Se puede acceder a los miembros y métodos de un componente a través de la entidad desde el objeto **.components** . Busque el componente en el mapa de componentes de la entidad y tendremos acceso a los componentes internos del componente. Considere este componente de ejemplo:

```
AFRAME.registerComponent('foo', {
  init: function () {
    this.bar = 'baz';
  },
  qux: function () {
    // ...
  }
});
```

Vamos a acceder al miembro de la **barra** y al método **qux** :

```
var fooComponent = document.querySelector('[foo]').components.foo;
console.log(fooComponent.bar);
fooComponent.qux();
```

Lea Componentes en línea: https://riptutorial.com/es/aframe/topic/10068/componentes

Capítulo 5: Controles (componente)

Introducción

Los controladores son vitales para sumergir a las personas en una aplicación de realidad virtual. El potencial de la realidad virtual no se cumple sin ellos, a saber, los controladores que proporcionan seis grados de libertad (6DoF). Con los controladores, las personas pueden acercarse a la escena e interactuar con los objetos con las manos.

A-Frame proporciona componentes para los controladores de todo el espectro como lo admiten sus respectivos navegadores WebVR a través de la API web de Gamepad. Hay componentes para los controladores Vive, Oculus Touch, Daydream y GearVR.

Observaciones

Es posible que debas habilitar las extensiones de gamepad. Podrías hacer eso siguiendo estos pasos:

- **En Chrome:** navega a `chrome://flags`
- **En Firefox:** navega hasta `about:config`
- **En IE:** Vaya al Editor de políticas de grupo en su escritorio
- **En Opera:** navegue a `opera:config`
- **On Edge:** navega hasta `about:flags`

Examples

Controles wasd

El componente wasd-controls controla una entidad con las teclas `W` , `A` , `S` y `D` o las teclas de flecha. El componente `wasd-controls` normalmente se adjunta a una entidad con el componente de cámara.

```
<a-entity camera look-controls wasd-controls></a-entity>
```

Para teclados azerty, puede usar las teclas `Z` , `Q` , `S` y `D`

Mira los controles

El componente look-controles:

- Gira la entidad cuando giramos una pantalla montada en la cabeza VR (HMD).
- Gira la entidad cuando hacemos clic y arrastramos el mouse.
- Gira la entidad cuando tocamos y arrastramos la pantalla táctil.

El componente de controles de apariencia se usa generalmente junto con el componente de la

cámara.

```
<a-entity camera look-controls></a-entity>
```

Advertencias

Si desea crear su propio componente para los controles de aspecto, tendrá que copiar y pegar los bits de seguimiento HMD en su componente. En el futuro, es posible que tengamos un sistema para que las personas creen sus controles más fácilmente.

Añadiendo mirada al cursor.

Para esto necesitas agregar un componente de cursor a tu cámara

```
<a-scene>
  <a-camera>
    <a-cursor></a-cursor>
    <!-- Or <a-entity cursor></a-entity> -->
  </a-camera>
</a-scene>
```

Más información se puede encontrar en el tema del cursor (componente) .

Controles de mano

A-Frame 0.x 0.3

A-Frame proporciona una implementación para admitir múltiples tipos de controladores 6DoF (Vive, Oculus Touch) a través del componente de controles manuales. El componente de control manual es principalmente para los controladores 6DoF, ya que está orientado a las interacciones a escala de la sala, como agarrar objetos. El componente de controles manuales funciona sobre los controladores Vive y Oculus Touch de la siguiente manera:

- Configuración de los componentes vive-controls y oculus-touch-controls
- Anulando los modelos de controlador con un modelo de mano simple
- Asignación de eventos específicos de Vive y Oculus Touch a eventos y gestos de mano (p. Ej., Agarre hacia abajo y desencadenamiento para minima)

Para agregar el componente de controles manuales:

```
<a-entity hand-controls="left"></a-entity>
<a-entity hand-controls="right"></a-entity>
```

Desafortunadamente, aún no hay un componente de controlador 3DoF que resuma bien todos los tipos de controladores 3DoF (es decir, Daydream, GearVR). Podríamos crear un controlador

personalizado que funcione con ambos controladores. Sería bastante fácil de cubrir ya que los controladores 3DoF no ofrecen mucho potencial para la interacción (es decir, solo el seguimiento rotativo con un panel táctil).

Los controles manuales dan manos rastreadas (utilizando un modelo prescrito) con gestos animados. los controles manuales envuelven los componentes de controles de vida y controles de oculus-touch, que a su vez envuelven el componente de controles controlados. El componente proporciona eventos adicionales y maneja animaciones y poses de mano.

```
<a-entity hand-controls="left"></a-entity>
<a-entity hand-controls="right"></a-entity>
```

Controles de seguimiento

A-Frame 0.x 0.3

El componente de controles con seguimiento es el componente del controlador base de A-Frame que proporciona la base para todos los componentes del controlador de A-Frame. El componente de control de seguimiento:

- Agarra un objeto Gamepad de la API de Gamepad dado un ID o prefijo.
- Aplica la posición (posición y orientación) de la API de Gamepad para leer el movimiento del controlador.
- Busca cambios en los valores de los botones del objeto del Gamepad para proporcionar eventos cuando se presionan o tocan los botones y cuando se cambian el eje y los `axischanged` `buttonchanged` (es decir, el `buttonchanged` `axischanged` , el `buttonchanged` , el `buttondown` , el `buttonup` , el `touchstart` , el `touchend`).

Todos los componentes del controlador de A-Frame se construyen sobre el componente de `tracked-controls` al:

- Configuración del componente de controles rastreados en la entidad con el ID de Gamepad adecuado (por ejemplo, Oculus Touch (Derecha)). Por ejemplo, el componente vive-controls hace

  ```
  el.setAttribute('tracked-controls', {idPrefix: 'OpenVR'})
  ```

  `tracked-controls` se conectarán al objeto Gamepad adecuado para proporcionar una pose y eventos para la entidad.
- Resumen de los eventos proporcionados por los controles de seguimiento. los eventos de control de seguimiento son de bajo nivel; Nos resultaría difícil decir qué botones se presionaron basándose solo en esos eventos porque tendríamos que conocer las asignaciones de botones de antemano. Componentes controlador puede saber las asignaciones de antemano por sus respectivos controladores y proporcionar más eventos semánticos tales como `triggerdown` o `xbuttonup` .

- Proporcionando un modelo. `tracked-controls` solo no proporciona ninguna apariencia. Los componentes del controlador pueden proporcionar un modelo que muestra comentarios visuales, gestos y animaciones cuando se pulsan o se tocan los botones. Los siguientes componentes del controlador solo se activan si detectan que el controlador se encuentra y se ve como conectado en la API de Gamepad.

Los `tracked-controls` interfaces de componentes con controladores de seguimiento. los controles rastreados utilizan la API de Gamepad para manejar los controladores rastreados, y se abstraen mediante el componente de controles manuales, así como los componentes de controles de vida y controles de oculus-touch. Este componente elige el controlador apropiado, aplica la pose a la entidad, observa el estado de los botones y emite los eventos apropiados.

Tenga en cuenta que debido a los cambios específicos del navegador, la API de Gamepad puede devolver los controladores Vive con los valores de identificación de *"OpenVR Gamepad"* o *"OpenVR Controller"* , por lo que se `idPrefix` usar `idPrefix` para los controladores Vive / OpenVR.

```
<a-entity tracked-controls="controller: 0; idPrefix: OpenVR"></a-entity>
```

Controladores 3Dof y 6Dof

Añadiendo Controladores 3DoF

Los controladores con 3 grados de libertad (3DoF) están limitados al seguimiento rotacional. Los controladores 3DoF no tienen un seguimiento posicional, lo que significa que no podemos llegar ni mover nuestra mano hacia adelante y hacia atrás o hacia abajo. Tener un controlador con solo 3DoF es como tener una mano y una muñeca sin un brazo. Lea más sobre los grados de libertad para VR.

Los componentes del controlador 3DoF proporcionan un seguimiento rotativo, un modelo predeterminado que coincide con el hardware de la vida real y eventos para abstraer las asignaciones de botones. Los controladores para Google Daydream y Samsung GearVR tienen 3DoF, y ambos admiten solo un controlador por una mano.

A-Frame 0.x 0.6

Controladores de ensueño

El componente de controles de Daydream interactúa con los controladores de Google Daydream. Envuelve el componente de controles rastreados al agregar asignaciones de botones, eventos y un modelo de controlador de Daydream que resalta los botones tocados y / o presionados (trackpad).

Haga coincidir el controlador Daydream si está presente, independientemente de la mano.

```
<a-entity daydream-controls></a-entity>
```

Haga coincidir el controlador de Daydream si está presente y para la mano especificada.

```
<a-entity daydream-controls="hand: left"></a-entity>
<a-entity daydream-controls="hand: right"></a-entity>
```

Controladores GearVR

El componente gearvr controla los interfaces con los controladores Samsung / Oculus Gear VR. Envuelve el componente de controles rastreados al agregar asignaciones de botones, eventos y un modelo de controlador de Gear VR que resalta los botones tocados y / o presionados (trackpad, disparador).

```
<!-- Match Gear VR controller if present, regardless of hand. -->
<a-entity gearvr-controls></a-entity>
<!-- Match Gear VR controller if present and for specified hand. -->
<a-entity gearvr-controls="hand: left"></a-entity>
<a-entity gearvr-controls="hand: right"></a-entity>
```

Agregando Controladores 6DoF

Los controladores con 6 grados de libertad (6DoF) tienen un seguimiento tanto rotacional como posicional. A diferencia de los controladores con 3DoF que están restringidos a la orientación, los controladores con 6DoF pueden moverse libremente en el espacio 3D. 6DoF nos permite avanzar, detrás de nuestras espaldas, mover nuestras manos a través de nuestro cuerpo o cerca de nuestra cara. Tener 6DoF es como la realidad donde tenemos ambas manos y brazos. 6DoF también se aplica a los auriculares y rastreadores adicionales (por ejemplo, pies, accesorios). Tener 6DoF es un mínimo para proporcionar una experiencia de VR realmente inmersiva.

Los componentes del controlador 6DoF proporcionan un seguimiento completo, un modelo predeterminado que coincide con el hardware de la vida real y eventos para abstraer las asignaciones de botones. HTC Vive y Oculus Rift with Touch proporcionan 6DoF y controladores para ambas manos. HTC Vive también ofrece rastreadores para rastrear objetos adicionales en el mundo real en VR.

A-Frame 0.x 0.3

Controladores de vive

El componente vive-controls se conecta con los controladores / varitas de HTC Vive. Envuelve el componente de controles rastreados al agregar asignaciones de botones, eventos y un modelo de controlador Vive que resalta los botones presionados (disparador, control, menú, sistema) y trackpad.

```
<a-entity vive-controls="hand: left"></a-entity>
<a-entity vive-controls="hand: right"></a-entity>
```

A-Frame 0.x 0.5

Controladores táctiles Oculus

El componente oculus-touch-controles se interconecta con los controladores Oculus Touch. Envuelve el componente de controles rastreados al agregar asignaciones de botones, eventos y un modelo de controlador táctil.

```
<a-entity oculus-touch-controls="hand: left"></a-entity>
<a-entity oculus-touch-controls="hand: right"></a-entity>
```

Control del mouse

Los controles del mouse solo se admiten fuera del modo VR y podrían usarse para juegos sin un HMD. Para obtener más información sobre los controles del mouse, puede encontrar en el ejemplo del cursor del mouse .

```
<a-scene>
  <a-entity camera look-controls mouse-cursor>
</a-scene>
```

Lea Controles (componente) en línea: https://riptutorial.com/es/aframe/topic/10112/controles--componente-

Capítulo 6: cursores

Introducción

El componente del cursor nos permite interactuar con entidades haciendo clic y mirando.

Sintaxis

- <a-entity cursor = ""> </a-cursor>
- <a-cursor> </a-cursor>

Parámetros

Propiedad	Descripción
fusible	Si el cursor está basado en fusibles. Valor predeterminado: `false` en el escritorio, `true` en el móvil
fuseTimeout	Cuánto tiempo se debe esperar (en milisegundos) antes de activar un evento de clic basado en fusibles. Valor por defecto: 1500

Observaciones

El cursor es una aplicación específica del componente raycaster en el sentido de que

- Escucha los clics del mouse y los fusibles basados en la mirada.
- Captura solo la primera entidad intersectada
- Emite eventos especiales del mouse y el mouse (por ejemplo, relacionados con el mouse arriba / abajo / ingresar / salir)
- Tiene más estados para flotar.

Cuando el mouse hace clic, la entidad visible más cercana que interseca el cursor, si existe, emitirá un evento de clic. Tenga en cuenta que el componente del cursor solo aplica el comportamiento de emisión de rayos. Para proporcionar una forma o apariencia al cursor, puede aplicar los componentes de geometría y material.

Eventos

Evento	Descripción
hacer clic	Se emite tanto en el cursor como en la entidad intersectada si se hace clic en una entidad actualmente intersectada (ya sea con el mouse o con un

Evento	Descripción
	fusible).
fusionando	Se emite tanto en el cursor como en la entidad intersectada cuando el cursor basado en fusible comienza la cuenta atrás.
ratón hacia abajo	Se emite tanto en el cursor como en la entidad intersecada (si existe) en la posición de ratón en el elemento del lienzo.
mouseenter	Se emite tanto en el cursor como en la entidad intersecada (si existe) cuando el cursor se interseca con una entidad.
mouseleave	Se emite tanto en el cursor como en la entidad intersecada (si existe) cuando el cursor ya no se interseca con la entidad previamente intersectada.
mouseup	Se emite tanto en el cursor como en la entidad intersecada (si existe) en mouseup en el elemento de lienzo.

Examples

Cursor predeterminado

Por ejemplo, podemos crear un cursor en forma de anillo fijo al centro de la pantalla. Para fijar el cursor en la pantalla para que el cursor esté siempre presente, sin importar dónde miremos, lo colocamos como un elemento secundario de la entidad de cámara activa. Lo colocamos frente a la cámara colocándolo en el eje Z negativo. Cuando el cursor hace clic en el cuadro, podemos escuchar el evento clic.

```
<a-entity camera>
  <a-entity cursor="fuse: true; fuseTimeout: 500"
            position="0 0 -1"
            geometry="primitive: ring; radiusInner: 0.02; radiusOuter: 0.03"
            material="color: black; shader: flat">
  </a-entity>
</a-entity>

<a-entity id="box" cursor-listener geometry="primitive: box" material="color: blue">
</a-entity>
```

```
// Component to change to random color on click.
AFRAME.registerComponent('cursor-listener', {
  init: function () {
    var COLORS = ['red', 'green', 'blue'];
    this.el.addEventListener('click', function (evt) {
      var randomIndex = Math.floor(Math.random() * COLORS.length);
      this.setAttribute('material', 'color', COLORS[randomIndex]);
      console.log('I was clicked at: ', evt.detail.intersection.point);
    });
  }
});
```

Interacciones basadas en la mirada con el cursor Componente

Primero repasaremos las interacciones basadas en la mirada. Las interacciones basadas en la mirada se basan en girar la cabeza y mirar los objetos para interactuar con ellos. Este tipo de interacción es para auriculares sin controlador. Incluso con un controlador de solo rotación (Daydream, GearVR), la interacción sigue siendo similar. Debido a que A-Frame proporciona controles de arrastre con el mouse de manera predeterminada, se puede usar la función basada en la mirada en el escritorio para previsualizar la interacción arrastrando la rotación de la cámara.

Para agregar una interacción basada en la mirada, necesitamos agregar o implementar un componente. A-Frame viene con un componente de cursor que proporciona interacción basada en la mirada si está conectado a la cámara:

1. Defina explícitamente la entidad. Anteriormente, A-Frame proporcionaba la cámara predeterminada.
2. Agregue `a-cursor` entidad de `a-cursor` como un elemento secundario debajo de la entidad de cámara.
3. Opcionalmente, configure el raycaster usado por el cursor.

```
<a-scene>
  <a-camera>
    <a-cursor></a-cursor>
    <!-- Or <a-entity cursor></a-entity> -->
  </a-camera>
</a-scene>
```

primitiva de un cursor

El cursor primitivo es una retícula que permite hacer clic e interactividad básica con una escena en dispositivos que no tienen un controlador manual. El aspecto predeterminado es una geometría de anillo. El cursor suele colocarse como un hijo de la cámara.

```
<a-scene>
  <a-camera>
    <a-cursor></a-cursor>
  </a-camera>
  <a-box></a-box>
</a-scene>
```

Cursor basado en fusible

También conocido como cursor basado en la mirada. Si configuramos el cursor para que esté basado en fusibles, el cursor activará un clic si el usuario observa una entidad durante un período de tiempo determinado. Imagine un láser atado a la cabeza del usuario, y el láser se extiende hacia la escena. Si el usuario mira a una entidad el tiempo suficiente (es decir, fuseTimeout), entonces el cursor activará un clic.

La ventaja de las interacciones basadas en fusibles para la realidad virtual es que no requiere dispositivos de entrada adicionales que no sean los auriculares. El cursor basado en fusible está

destinado principalmente a las aplicaciones de Google Cardboard. La desventaja de las interacciones basadas en fusibles es que requiere que el usuario gire mucho la cabeza.

Configurando el cursor a través del componente Raycaster

El cursor se construye sobre y depende del componente raycaster. Si queremos personalizar las partes de raycasting del cursor, podemos hacerlo cambiando las propiedades del componente raycaster. Digamos que queremos establecer una distancia máxima, verificar las intersecciones con menos frecuencia y establecer qué objetos se pueden hacer clic:

```
<a-entity cursor raycaster="far: 20; interval: 1000; objects: .clickable"></a-entity>
```

Añadiendo comentarios visuales

Para agregar información visual al cursor para mostrar cuando el cursor está haciendo clic o fusionándose, podemos usar el sistema de animación. Cuando el cursor se cruza con la entidad, emitirá un evento, y el sistema de animación recogerá el evento con el atributo de inicio:

```
<a-entity cursor="fuse: true; fuseTimeout: 500"
          position="0 0 -1"
          geometry="primitive: ring"
          material="color: black; shader: flat">
  <a-animation begin="click" easing="ease-in" attribute="scale"
               fill="backwards" from="0.1 0.1 0.1" to="1 1 1"></a-animation>
  <a-animation begin="cursor-fusing" easing="ease-in" attribute="scale"
               fill="forwards" from="1 1 1" to="0.1 0.1 0.1"></a-animation>
</a-entity>
```

Cursor del ratón

Nota: para este ejemplo, debe agregar un paquete npm externo.

Si desea utilizar el cursor del mouse de su computadora, debe agregar un `aframe-mouse-cursor-component` . Después si debes incluir el script usando este código:

```
import 'aframe';
import 'aframe-mouse-cursor-component';

// or this

require('aframe');
require('aframe-mouse-cursor-component');
```

Y en tu cámara necesitas agregar el componente del `mouse-cursor` del `mouse-cursor` .

```
<a-scene>
  <a-entity camera look-controls mouse-cursor>
</a-scene>
```

Lea cursores en línea: https://riptutorial.com/es/aframe/topic/10180/cursores

Capítulo 7: Entidades

Introducción

A-Frame representa una entidad a través del elemento `<a-entity>` . Como se define en el patrón entidad-componente-sistema, las entidades son objetos de marcador de posición a los que conectamos componentes para proporcionarles apariencia, comportamiento y funcionalidad.

Sintaxis

- `<a-entity> // Consider the entity below. By itself, it has no appearance, behavior, or functionality. It does nothing:`
- `<a-entity geometry="primitive: box" material="color: red"> // We can attach components to it to make it render something or do something. To give it shape and appearance, we can attach the geometry and material components:`
- `<a-entity geometry="primitive: box" material="color: red" light="type: point; intensity: 2.0"> // Or to make it emit light, we can further attach the light component:`

Parámetros

Parámetro	Detalles
componentes	`<a-entity>.components` es un objeto de componentes adjuntos a la entidad. Esto nos da acceso a los componentes de la entidad, incluidos los datos, el estado y los métodos de cada componente.
está jugando	Si la entidad está activa y jugando. Si pausamos la entidad, entonces la reproducción se vuelve falsa.
object3D	`<a-entity>.object3D` es una referencia a la representación de tres.js Object3D de la entidad. Más específicamente, object3D será un objeto de `THREE.Group` que puede contener diferentes tipos de `THREE.Object3D` como cámaras, mallas, luces o sonidos:
object3DMap	El object3DMap de una entidad es un objeto que da acceso a los diferentes tipos de THREE.Object3Ds (por ejemplo, cámara, mallas, luces, sonidos) que los componentes han establecido.
escenaEl	Una entidad tiene una referencia a su elemento de escena.

Observaciones

Metodos

addState (stateName)

addState empujará un estado en la entidad. Esta emitirá el evento **stateadded,** y podemos comprobar el estado de lata su existencia utilizando **.is:**

```
entity.addEventListener('stateadded', function (evt) {
  if (evt.detail.state === 'selected') {
    console.log('Entity now selected!');
  }
});
entity.addState('selected');
entity.is('selected');  // >> true
```

---

emitir (nombre, detalle, burbujas)

emit emite un evento DOM personalizado en la entidad. Por ejemplo, podemos emitir un evento para activar una animación:

```
<a-entity>
  <a-animation attribute="rotation" begin="rotate" to="0 360 0"></a-animation>
</a-entity>
```

```
entity.emit('rotate');
```

También podemos pasar detalles del evento o datos como segundo argumento:

```
entity.emit('collide', { target: collidingEntity });
```

El evento burbujeará por defecto. Podemos decirle que no burbujee pasando falso por burbuja:

```
entity.emit('sink', null, false);
```

---

flushToDOM (recursivo)

flushToDOM serializará manualmente los datos de los componentes de una entidad y actualizará el DOM.

---

getAttribute (componentName)

getAttribute recupera datos de componentes analizados (incluidos mixins y valores predeterminados).

```
// <a-entity geometry="primitive: box; width: 3">
entity.getAttribute('geometry');
// >> {primitive: "box", depth: 2, height: 2, translate: "0 0 0", width: 3, ...}
entity.getAttribute('geometry').primitive;
// >> "box"
entity.getAttribute('geometry').height;
// >> 2
```

```
entity.getAttribute('position');
// >> {x: 0, y: 0, z: 0}
```

Si componentName no es el nombre de un componente registrado, getAttribute se comportará como lo haría normalmente:

```
// <a-entity data-position="0 1 1">
entity.getAttribute('data-position');
// >> "0 1 1"
```

---

getDOMAttribute (componentName)

getDOMAttribute recupera solo los datos de componentes analizados que están definidos explícitamente en el DOM o mediante **setAttribute** . Si **componentName** es el nombre de un componente registrado, **getDOMAttribute** devolverá solo los datos del componente definidos en el HTML como un objeto analizado. **getDOMAttribute** para componentes es la forma parcial de **getAttribute** ya que los datos de componentes devueltos no incluyen mixins aplicados o valores predeterminados:

Compare la salida del ejemplo anterior de **getAttribute** :

```
// <a-entity geometry="primitive: box; width: 3">
entity.getDOMAttribute('geometry');
// >> { primitive: "box", width: 3 }
entity.getDOMAttribute('geometry').primitive;
// >> "box"
entity.getDOMAttribute('geometry').height;
// >> undefined
entity.getDOMAttribute('position');
// >> undefined
```

---

getObject3D (tipo)

getObject3D busca un hijo **TRES. Objeto 3D** referenciado por **tipo** en **object3DMap** .

```
AFRAME.registerComponent('example-mesh', {
  init: function () {
     var el = this.el;
     el.getOrCreateObject3D('mesh', THREE.Mesh);
     el.getObject3D('mesh');  // Returns THREE.Mesh that was just created.
  }
});
```

---

getOrCreateObject3D (tipo, Constructor)

Si la entidad no tiene un **THREE.Object3D** registrado en el **tipo** , **getOrCreateObject3D** registrará un **THREE.Object3D** instanciado utilizando el **Constructor** pasado. Si la entidad tiene un **THREE.Object3D** registrado en **tipo** , **getOrCreateObject3D** actuará como **getObject3D** :

```
AFRAME.registerComponent('example-geometry', {
  update: function () {
    var mesh = this.el.getOrCreateObject3D('mesh', THREE.Mesh);
    mesh.geometry = new THREE.Geometry();
  }
});
```

---

pausa ()

pause () detendrá cualquier comportamiento dinámico definido por animaciones y componentes. Cuando pausamos una entidad, detendrá sus animaciones y llamará a **Component.pause ()** en cada uno de sus componentes. Los componentes deciden implementar lo que sucede en la pausa, que a menudo es eliminar los escuchas de eventos. Una entidad llamará a pausa () en sus entidades secundarias cuando pausemos una entidad.

```
// <a-entity id="spinning-jumping-ball">
entity.pause();
```

Por ejemplo, el componente look-controls on pause eliminará los controladores de eventos que escuchan las entradas.

---

jugar ()

play () iniciará cualquier comportamiento dinámico definido por animaciones y componentes. Esto se llama automáticamente cuando el DOM adjunta una entidad. Cuando una entidad **juega ()** , la entidad llama a **play ()** en sus entidades secundarias.

```
entity.pause();
entity.play();
```

Por ejemplo, el componente de sonido en la reproducción comenzará a reproducir el sonido.

---

setAttribute (componentName, value, [propertyValue | clobber])

Si **componentName** no es el nombre de un componente registrado o si el componente es un componente de una sola propiedad, **setAttribute se** comporta como lo haría normalmente:

```
entity.setAttribute('visible', false);
```

Aunque si **componentName** es el nombre de un componente registrado, puede manejar un análisis especial para el valor. Por ejemplo, el **componente de posición** es un **componente de** una sola propiedad, pero su analizador de tipo de propiedad le permite tomar un objeto:

```
entity.setAttribute('position', { x: 1, y: 2, z: 3 });
```

---

setObject3D (tipo, obj)

setObject3D registrará el **obj** pasado, un **THREE.Object3D** , como **tipo** debajo del **object3DMap de** la entidad. A-Frame agrega **obj** como hijo del objeto **raíz3D** de la entidad.

```
AFRAME.registerComponent('example-orthogonal-camera', {
  update: function () {
    this.el.setObject3D('camera', new THREE.OrthogonalCamera());
  }
});
```

---

removeAttribute (componentName, propertyName)

Si **componentName** es el nombre de un componente registrado, junto con la eliminación del atributo del DOM, **removeAttribute** también separará el componente de la entidad, invocando el método de **eliminación del** ciclo de vida del componente.

```
entity.removeAttribute('goemetry');  // Detach the geometry component.
entity.removeAttribute('sound');  // Detach the sound component.
```

Si se proporciona **propertyName** , **removeAttribute** restablecerá el valor de propiedad de esa propiedad especificada por **propertyName** al valor predeterminado de la propiedad:

```
entity.setAttribute('material', 'color', 'blue');  // The color is blue.
entity.removeAttribute('material', 'color');  // Reset the color to the default value, white.
```

---

removeObject3D (tipo)

removeObject3D elimina el objeto especificado por **tipo** de **THREE.Group** de la entidad y, por lo tanto, de la escena. Esto actualizará el **object3DMap de** la entidad, estableciendo el valor de la clave de **tipo** en **nulo** . Esto generalmente se llama desde un componente, a menudo dentro del controlador de eliminación:

```
AFRAME.registerComponent('example-light', {
  update: function () {
    this.el.setObject3D('light', new THREE.Light());
    // Light is now part of the scene.
    // object3DMap.light is now a THREE.Light() object.
  },
  remove: function () {
    this.el.removeObject3D('light');
    // Light is now removed from the scene.
    // object3DMap.light is now null.
  }
});
```

---

removeState (stateName)

removeState sacará un estado de la entidad. Esto emitirá el evento **stateremoved** , y podemos verificar el estado de su eliminación mediante **.is** :

```
entity.addEventListener('stateremoved', function (evt) {
  if (evt.detail.state === 'selected') {
    console.log('Entity no longer selected.');
  }
});
entity.addState('selected');
entity.is('selected');  // >> true
entity.removeState('selected');
entity.is('selected');  // >> false
```

EVENTOS

Nombre del evento	Descripción
adjunto al niño	Una entidad hijo se adjuntó a la entidad.
separado del niño	Una entidad hijo fue separada de la entidad.
componente cambiado	Uno de los componentes de la entidad fue modificado.
componenteinit	Uno de los componentes de la entidad fue inicializado.
componerse	Uno de los componentes de la entidad fue eliminado.
cargado	La entidad ha adjuntado e inicializado sus componentes.
object3dset	THREE.Object3D se configuró en la entidad usando setObject3D (nombre). El detalle del evento contendrá el nombre utilizado para establecer en el object3DMap.
pausa	La entidad ahora está inactiva y en pausa en términos de comportamiento dinámico.
jugar	La entidad ahora está activa y jugando en términos de comportamiento dinámico.
declarado	La entidad recibió un nuevo estado.
stateremoved	La entidad ya no tiene un cierto estado.
esquema cambiado	Se cambió el esquema de un componente.

DETALLES DEL EVENTO

A continuación se muestra lo que contiene el detalle del evento para cada evento:

Nombre del evento	Propiedad	Descripción
adjunto al niño	el	Referencia al elemento hijo adjunto.
componente cambiado	nombre	Nombre del componente que tuvo sus datos modificados.
	carné de identidad	ID del componente que tuvo sus datos modificados.
	nuevos datos	Nuevos datos del componente, luego de su modificación.
	Datos antiguos	Datos anteriores del componente, antes de su modificación.
componente inicializado	nombre	Nombre del componente que se inicializó.
	carné de identidad	ID del componente que tuvo sus datos modificados.
	datos	Datos del componente.
componerse	nombre	Nombre del componente que fue eliminado.
	carné de identidad	ID del componente que se eliminó.
declarado	estado	El estado que se adjuntó (cadena).
stateremoved	estado	El estado que fue separado (cadena).
esquema cambiado	componente	Nombre del componente que tuvo su esquema cambiado.

Examples

Escuchar los cambios de componentes

Podemos usar el evento de cambio de **componente** para escuchar los cambios en la entidad:

```
entity.addEventListener('componentchanged', function (evt) {
  if (evt.detail.name === 'position') {
    console.log('Entity has moved from',
      evt.detail.oldData, 'to', evt.detail.newData, '!');
  }
```

```
});
```

Escuchando los elementos del niño que están unidos y separados

Podemos usar los eventos **adjuntos** y **separados del niño** para escuchar cuando la escena adjunta o separa una entidad:

```
entity.addEventListener('child-attached', function (evt) {
  if (evt.detail.el.tagName.toLowerCase() === 'a-box') {
    console.log('a box element has been attached');
  };
});
```

Datos de componentes de múltiples propiedades de la entidad (setAttribute)

Actualización de datos de componentes de propiedades múltiples

Para actualizar los datos del componente para un componente de múltiples propiedades, podemos pasar el nombre de un componente registrado como el nombre de **componente** , y pasar un objeto de propiedades como el **valor** . Una cadena también es aceptable (p. Ej., **Escriba: punto; distancia: 30**), pero los objetos guardarán A-Frame en algunos trabajos de análisis:

```
// Only the properties passed in the object will be overwritten.
entity.setAttribute('light', {
  type: 'spot',
  distance: 30,
  intensity: 2.0
});
```

O para actualizar propiedades individuales para un componente de múltiples propiedades, podemos pasar el nombre del componente registrado como el nombre de **componente** , un nombre de propiedad como el segundo argumento y el valor de la propiedad para establecer como el tercer argumento:

```
// All previous properties for the material component (besides the color)  will be unaffected.
entity.setAttribute('material', 'color', 'crimson');
```

Tenga en cuenta que los tipos de propiedad de matriz se comportan de forma única:

- Las matrices son mutables. Se asignan por referencia para que los cambios en las matrices sean visibles por el componente.
- Las actualizaciones de las propiedades de tipo de matriz no activarán el método de actualización del componente ni emitirán eventos.

Actualización de datos de componentes de propiedades

múltiples

Si se pasa **verdadero** como tercer argumento a **.setAttribute** , las propiedades no especificadas se restablecerán y se borrarán:

```
// All previous properties for the light component will be removed and overwritten.
entity.setAttribute('light', {
  type: 'spot',
  distance: 30,
  intensity: 2.0
}, true);
```

Recuperando una Entidad

Simplemente podemos recuperar una entidad utilizando las API de DOM.

```
<a-entity id="mario"></a-entity>
```

```
var el = document.querySelector('#mario');
```

Recuperando componentes de una entidad

Por ejemplo, si quisiéramos agarrar el objeto de cámara u objeto material de una entidad de tres.js, podríamos llegar a sus componentes

```
var camera = document.querySelector('a-entity[camera]').components.camera.camera;
var material = document.querySelector('a-entity[material]').components.material.material;
```

O si un componente expone una API, podemos llamar a sus métodos:

```
document.querySelector('a-entity[sound]').components.sound.pause();
```

Lea Entidades en línea: https://riptutorial.com/es/aframe/topic/10066/entidades--a-entity-

Capítulo 8: Escena

Introducción

Una escena está representada por el elemento `<a-scene>` . La escena es el objeto raíz global, y todas las entidades están contenidas dentro de la escena.

La escena hereda de la clase Entidad, por lo que hereda todas sus propiedades, sus métodos, la capacidad de adjuntar componentes y el comportamiento para esperar a todos sus nodos secundarios (por ejemplo, `<a-assets>` y `<a-entity>`) para cargar antes de iniciar el bucle de render.

Parámetros

Parámetro	Detalles
comportamientos	Conjunto de componentes con métodos de tick que se ejecutarán en cada fotograma.
cámara	Cámara activa de three.js.
lona	Referencia al elemento canvas.
ismobile	Si el entorno es detectado o no para ser móvil.
object3D	TRES.Escena de objeto.
renderizador	Activo THREE.WebGLRenderer.
renderStarted	Si la escena está representando.
efecto	Renderizador para VR creado al pasar un renderizador activo a THREE.VREffect.
sistemas	Sistemas instanciados.
hora	Tiempo de actividad global de la escena en segundos.

Observaciones

Metodos

Nombre	Descripción
enterVR	Cambia al renderizado estéreo y coloca el contenido en el auricular. Debe

Nombre	Descripción
	llamarse dentro de un controlador de eventos generado por el usuario, como `click` . la primera vez que una página entra en VR.
exitVR	Cambia al renderizador mono y deja de presentar contenido en el auricular.
recargar	Revertir la escena a su estado original.

EVENTOS

Nombre	Descripción
enter-vr	El usuario ha introducido VR y los auriculares han comenzado a presentar contenido.
salida-vr	El usuario ha salido de la realidad virtual y los auriculares han dejado de presentar contenido.
cargado	Todos los nodos se han cargado.
renderstart	El bucle de render ha comenzado.

Examples

Adjuntar componentes de escena

Los componentes se pueden adjuntar a la escena, así como las entidades. A-Frame se envía con algunos componentes para configurar la escena:

Componente	Detalles
incrustado	Eliminar estilos de pantalla completa del lienzo.
niebla	Añadir niebla.
atajos de teclado	Alternar atajos de teclado.
inspector	Inyectar el inspector de fotograma a.
estadísticas	Alternar estadísticas de rendimiento.
vr-mode-ui	Alternar la interfaz de usuario para entrar y salir de la realidad virtual.
depurar	Habilita la serialización de componente a DOM.

Usando escenas incrustadas

Si desea usar contenido de WebVR mezclado con contenido HTML, por ejemplo, cuando está creando un contenido de clave de exhibición extendido, puede usar la etiqueta `embedded` . Cuando está usando esto, es posible mirar a su alrededor dentro del contenido de 360 ° usando el giroscopio de su teléfono inteligente o hacer clic y arrastrar en la computadora.

```
<script src="https://aframe.io/releases/0.5.0/aframe.min.js"></script>
<div class="vrcontent">
  <a-scene embedded>
    <a-assets>
      <img id="sky" src="https://c1.staticflickr.com/5/4248/34705881091_37b5cf2d28_o.jpg"
alt="" />
    </a-assets>

    <a-sky src="#sky"></a-sky>
  </a-scene>
</div>

<div class="overlay">
  <button class="calltoaction">Click me!</button>
</div>

<div class="content">
  <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Deleniti animi aliquid
architecto quibusdam ipsum, debitis dolor mollitia. Quidem, cumque quos porro doloribus iure
dolore illum, qui rem asperiores unde laboriosam.Dolorum tempora quam eveniet ea recusandae
deserunt, velit similique. Cum sunt rerum beatae officiis qui sed molestiae et ullam quasi,
harum maxime vel, aspernatur quidem molestias. Provident quae illo harum?Sunt expedita,
repellat saepe vel accusamus odio. Alias, obcaecati harum earum inventore asperiores quaerat,
sit autem nostrum. Sunt illo numquam, temporibus pariatur optio nam, expedita necessitatibus
aliquid nemo maxime nisi. Praesentium corporis, ea sunt asperiores, recusandae animi, rem
doloribus, possimus cum laudantium libero. Maiores a, iusto aspernatur reiciendis ratione sunt
nisi, rem, quasi temporibus ullam non. Neque repellat facilis illo.Quibusdam reiciendis sunt
tempora fuga deleniti, molestias temporibus doloremque. Nam sed consequatur consectetur ut
tempora a nesciunt, perspiciatis dolorem reprehenderit modi enim at veritatis, excepturi
voluptate quod, voluptatibus voluptas. Cum.Debitis, nesciunt, repellat voluptatem sapiente
incidunt quidem asperiores reprehenderit vero quisquam placeat sunt voluptatibus velit.
Consectetur atque voluptates, repellendus facere sequi ea totam quia quis non incidunt.
Soluta, aut, provident. Eos sequi itaque dolorem atque ex id maiores dolor eaque libero iste
deserunt ea voluptate minima cum laboriosam, qui animi, fuga suscipit necessitatibus vero,
autem blanditiis, totam nulla. Quo, et. Quisquam commodi voluptatum dolorem aspernatur,
distinctio et ullam laborum laboriosam quo nisi, praesentium quaerat ab excepturi. Illum harum
doloremque, accusantium, beatae culpa assumenda laboriosam, quos mollitia aperiam dolorem
praesentium minus!</p>
</div>
```

Depurar

El componente de depuración permite la serialización de componente a DOM.

```
<a-scene debug></a-scene>
```

Serialización componente a DOM

De forma predeterminada, por razones de rendimiento, A-Frame no actualiza el DOM con datos de componentes. Si abrimos el inspector DOM del navegador, veremos que solo los nombres de los componentes (y no los valores) están visibles.

```
<a-entity geometry material position rotation></a-entity>
```

A-Frame almacena los datos del componente en la memoria. Actualizar el DOM lleva tiempo de CPU para convertir datos de componentes internos a cadenas. Si queremos ver la actualización de DOM para fines de depuración, podemos adjuntar el componente de depuración a la escena. Los componentes buscarán un componente de depuración habilitado antes de intentar serializar el DOM. Entonces podremos ver los datos del componente en el DOM:

```
<a-entity geometry="primitive: box" material="color: red" position="1 2 3" rotation="0 180
0"></a-entity>
```

Asegúrese de que este componente no esté activo en la producción.

Serialización manual a DOM

Para serializar manualmente a DOM, use Entity.flushToDOM o Component.flushToDOM:

```
document.querySelector('a-entity').components.position.flushToDOM();  // Flush a component.
document.querySelector('a-entity').flushToDOM();  // Flush an entity.
document.querySelector('a-entity').flushToDOM(true);  // Flush an entity and its children.
document.querySelector('a-scene').flushToDOM(true);  // Flush every entity.
```

Ejecución de scripts de contenido en la escena

La forma recomendada es escribir un componente y adjuntarlo al elemento de la escena.
La escena y sus hijos serán inicializados antes de este componente.

```
AFRAME.registerComponent('do-something', {
  init: function () {
    var sceneEl = this.el;
  }
});
```

```
<a-scene do-something></a-scene>
```

Si, por alguna razón en particular, prefiere no escribir un componente dedicado, debe esperar a que la escena termine de inicializarse y adjuntarse:

```
var scene = document.querySelector('a-scene');

if (scene.hasLoaded) {
  run();
} else {
  scene.addEventListener('loaded', run);
```

```
}

function run () {
  var entity = scene.querySelector('a-entity');
  entity.setAttribute('material', 'color', 'red');
}
```

Lea Escena en línea: https://riptutorial.com/es/aframe/topic/10069/escena--a-scene-

Capítulo 9: luz (componente)

Introducción

El componente de luz define la entidad como una fuente de luz. La luz afecta a todos los materiales que no han especificado un modelo de sombreado plano con sombreador: plano. Tenga en cuenta que las luces son computacionalmente caras, debemos limitar el número de luces en una escena.

Sintaxis

- <a-entity light = "color: #AFA; intensidad: 1.5" position = "- 1 1 0"> </a-entity>
- <a-light type = "point" color = "blue" position = "0 5 0"> </a-light>

Parámetros

Parámetros	Detalles
tipo	Uno de ambiente, direccional, hemisferio, punto, mancha.
color	Color claro.
intensidad	Fuerza de la luz.

Examples

Ambiente

Las luces ambientales afectan globalmente a todas las entidades en la escena. Las propiedades de color e intensidad definen las luces ambientales. Además, la posición, la rotación y la escala no tienen efecto en las luces ambientales.

Recomendamos tener algún tipo de luz ambiental tal que las áreas sombreadas no sean completamente negras e imitar la iluminación indirecta.

```
<a-entity light="type: ambient; color: #CCC"></a-entity>
```

Direccional

Las luces direccionales son como una fuente de luz que está infinitamente lejos, pero que brilla desde una dirección específica, como el sol. Por lo tanto, la posición absoluta no tiene un efecto sobre la intensidad de la luz en una entidad. Podemos especificar la dirección usando el componente de posición.

El siguiente ejemplo crea una fuente de luz que brilla desde la parte superior izquierda en un ángulo de 45 grados. Tenga en cuenta que como solo importa el vector, la posición = "- 100 100 0" y la posición = "- 1 1 0" son iguales.

```
<a-entity light="type: directional; color: #EEE; intensity: 0.5" position="-1 1 0"></a-entity>
```

Podemos especificar la dirección de la luz direccional con su orientación creando una entidad secundaria a la que se dirige. Por ejemplo, apuntando hacia abajo su eje -Z:

```
<a-light type="directional" position="0 0 0" rotation="-90 0 0" target="#directionaltarget">
    <a-entity id="directionaltarget" position="0 0 -1"></a-entity>
</a-light>
```

Hemisferio

Las luces del hemisferio son como una luz ambiental, pero con dos colores diferentes, uno desde arriba (color) y otro desde abajo (color de fondo). Esto puede ser útil para escenas con dos colores de iluminación distintos (por ejemplo, un campo de hierba bajo un cielo gris).

```
<a-entity light="type: hemisphere; color: #33C; groundColor: #3C3; intensity: 2"></a-entity>
```

Propiedad	Descripción	Valor por defecto
color de fondo	Color claro desde abajo.	#fff

Punto

Las luces puntuales, a diferencia de las luces direccionales, son omnidireccionales y afectan a los materiales dependiendo de su posición y distancia. Los "me gusta" son como una bombilla. Cuanto más se acerca la bombilla a un objeto, mayor es la iluminación del objeto.

```
<a-entity light="type: point; intensity: 0.75; distance: 50; decay: 2"
      position="0 10 10"></a-entity>
```

Propiedad	Descripción	Valor por defecto
decaer	La cantidad de luz se atenúa a lo largo de la distancia de la luz.	1.0
distancia	Distancia donde la intensidad se convierte en 0. Si la distancia es 0, entonces la luz del punto no decae con la distancia.	0.0

Lugar

Las luces puntuales son como luces puntuales en el sentido de que afectan a los materiales dependiendo de su posición y distancia, pero las luces puntuales no son omnidireccionales.

Principalmente arrojan luz en una dirección, como la Bat-Signal.

```
<a-entity light="type: spot; angle: 45"></a-entity>
```

Propiedad	Descripción	Valor por defecto
ángulo	Máxima extensión de la luz puntual desde su dirección (en grados).	60
decaer	La cantidad de luz se atenúa a lo largo de la distancia de la luz.	1.0
distancia	Distancia donde la intensidad se convierte en 0. Si la distancia es 0, entonces la luz del punto no decae con la distancia.	0.0
penumbra	Porcentaje del cono de foco que se atenúa debido a la penumbra.	0.0
objetivo	elemento que el punto debe señalar. configúrelo en nulo para transformar el foco por orientación, apuntando a su eje -Z.	nulo

Iluminación por defecto

Por defecto, las escenas de fotograma A inyectan iluminación predeterminada, una luz ambiental y una luz direccional. Estas luces predeterminadas son visibles en el DOM con el atributo data-aframe-default-light. Cada vez que agregamos luces, A-Frame elimina las luces predeterminadas de la escena.

```
<!-- Default lighting injected by A-Frame. -->
<a-entity light="type: ambient; color: #BBB"></a-entity>
<a-entity light="type: directional; color: #FFF; intensity: 0.6" position="-0.5 1 1"></a-
entity>
```

Lea luz (componente) en línea: https://riptutorial.com/es/aframe/topic/10078/luz--componente-

Capítulo 10: mezcla-modelo (componente)

Introducción

Componente del modelo de mezcla Carga un modelo JSON de formato three.js que contiene una combinación de animaciones esqueléticas utilizando **THREE.BlendCharacter** . Esto se utiliza principalmente para representar los controladores de mano y Vive.

Sintaxis

- `<a-entity blend-model="#a-asset-item-selector"></a-entity>`

Observaciones

VALORES

Tipo	Descripción
selector	Selector a un `<a-asset-item>`
cuerda	`url()` - ruta de acceso cerrada a un archivo JSON

EVENTOS

Nombre del evento	Descripción
modelo cargado	El modelo JSON fue cargado en la escena.

Examples

Ejemplo de uso de `blend-model`

Podemos cargar el modelo apuntando con la ID a una que especifica el src a un archivo:

```
<a-scene>
  <a-assets>
    <!-- At first we load skeletal animation blending JSON as asset -->
    <a-asset-item id="hand" src="/path/to/hand.json"></a-asset-item>
  </a-assets>
  <!-- Now we can use that asset with blend-model-->
  <a-entity blend-model="#hand"></a-entity>
</a-scene>
```

Lea mezcla-modelo (componente) en línea: https://riptutorial.com/es/aframe/topic/10073/mezcla-modelo--componente-

Capítulo 11: Mixins

Introducción

Los mixins proporcionan una forma de componer y reutilizar los conjuntos de propiedades de componentes utilizados comúnmente. Se definen utilizando el elemento `<a-mixin>` y se colocan en `<a-assets>` . Los mixins deben establecerse con un id, y cuando una entidad establece ese id como su atributo mixin, la entidad absorberá todos los atributos del mixin.

Examples

Ejemplo de uso de mixins

```
<a-scene>
  <a-assets>
    <a-mixin id="red" material="color: red"></a-mixin>
    <a-mixin id="blue" material="color: blue"></a-mixin>
    <a-mixin id="cube" geometry="primitive: box"></a-mixin>
  </a-assets>
  <a-entity mixin="red cube"></a-entity>
  <a-entity mixin="blue cube"></a-entity>
</a-scene>
```

La entidad con cubo rojo tomará las propiedades de la mezcla roja y la mezcla del cubo en ese orden. Lo mismo ocurre con el cubo azul. Conceptualmente, las entidades anteriores se expanden a:

```
<a-entity material="color: red" geometry="primitive: box"></a-entity>
<a-entity material="color: blue" geometry="primitive: box"></a-entity>
```

Fusionar propiedades de componentes

Las propiedades de un componente de múltiples propiedades se fusionarán si están definidas por varios mixins y / o la entidad. Por ejemplo:

```
<a-scene>
  <a-assets>
    <a-mixin id="box" geometry="primitive: box"></a-mixin>
    <a-mixin id="tall" geometry="height: 10"></a-mixin>
    <a-mixin id="wide" geometry="width: 10"></a-mixin>
  </a-assets>
  <a-entity mixin="wide tall box" geometry="depth: 2"></a-entity>
</a-scene>
```

Todas las propiedades del componente de geometría se fusionarán ya que se incluyen como mixins y se definen en la entidad. La entidad sería entonces equivalente a:

```
<a-entity geometry="primitive: box; height: 10; depth: 2; width: 10"></a-entity>
```

Orden y Precedencia

Cuando una entidad incluye varios mixins que definen las mismas propiedades del componente, el mixin más a la derecha tiene prioridad. En el siguiente ejemplo, la entidad incluye mixins `red` y `blue` , y como la mixin `blue` se incluye en último lugar, el color final del cubo será azul.

```
<a-scene>
  <a-assets>
    <a-mixin id="red" material="color: red"></a-mixin>
    <a-mixin id="blue" material="color: blue"></a-mixin>
    <a-mixin id="cube" geometry="primitive: box"></a-mixin>
  </a-assets>

  <a-entity mixin="red blue cube"></a-entity>
</a-scene>
```

Si una entidad define una propiedad que ya está definida por una combinación, la definición de la entidad tendrá prioridad. En el siguiente ejemplo, la entidad incluye mixins tanto `red` como `blue` y también define un color verde. Dado que la entidad define directamente su propio color, el color final del cubo será verde.

```
<a-scene>
  <a-assets>
    <a-mixin id="red" material="color: red"></a-mixin>
    <a-mixin id="blue" material="color: blue"></a-mixin>
    <a-mixin id="cube" geometry="primitive: box"></a-mixin>
  </a-assets>

  <a-entity mixin="red blue cube" material="color: green"></a-entity>
</a-scene>
```

Lea Mixins en línea: https://riptutorial.com/es/aframe/topic/10072/mixins--a-mixin-

Capítulo 12: modelo gltf (componente)

Introducción

El componente del modelo gltf permite utilizar modelos 3D en el formato glTF con A-Frame. glTF es un estándar de Khronos para modelos 3D eficientes de escena completa y está optimizado para su uso en la web.

Sintaxis

- `<a-entity gltf-model="url(https://cdn.rawgit.com/KhronosGroup/glTF-Sample-Models/9176d098/1.0/Duck/glTF/Duck.gltf)"></a-entity>`
- `<a-entity gltf-model="#duck"></a-entity>`

Parámetros

Parámetro	Detalles
`url(...)`	cargará el modelo glTF envuelto desde la URL envuelta en `url()`
`#example`	cargará el `<a-asset-item>` con el `example` ID

Examples

Cargando un modelo glTF a través de URL

```
<a-scene>
  <a-entity gltf-model="url(https://cdn.rawgit.com/KhronosGroup/glTF-Sample-Models/9176d098/1.0/Duck/glTF/Duck.gltf)" position="0 0 -5"></a-entity>
</a-scene>
```

Cargando un modelo gltf a través del sistema de activos

```
<a-scene>
  <a-assets>
    <a-asset-item id="duck" src="https://cdn.rawgit.com/KhronosGroup/glTF-Sample-Models/9176d098/1.0/Duck/glTF/Duck.gltf"></a-asset-item>
  </a-assets>

  <a-entity gltf-model="#duck" position="0 0 -5"></a-entity>
</a-scene>
```

Lea modelo gltf (componente) en línea: https://riptutorial.com/es/aframe/topic/10758/modelo-gltf--componente-

Capítulo 13: Primitivas

Introducción

Las primitivas son solo `<a-entity>` bajo el capó. Esto significa que las primitivas tienen la misma API que las entidades, como posicionamiento, rotación, escalado y conexión de componentes. A-Frame proporciona un puñado de elementos como `<a-box>` o `<a-sky>` llamados primitivos que envuelven el patrón entidad-componente para que resulte atractivo para los principiantes. . Los desarrolladores también pueden crear sus propios primitivos.

Observaciones

Bajo el capó

Los primitivos actúan como una capa de conveniencia (es decir, azúcar sintáctica) principalmente para los recién llegados. Tenga en cuenta por ahora que las primitivas están `<a-entity>` s debajo del capó que:

- Tener un nombre semántico (por ejemplo, `<a-box>`)
- Tener un paquete preestablecido de componentes con valores predeterminados
- Asignar o asignar atributos HTML a los datos de [componente]

Los primitivos son similares a los prefabs en Unity . Algunas publicaciones sobre el patrón entidad-componente-sistema se refieren a ellas como ensamblajes . Se abstrae la API de componente de entidad central para:

- Pre-componer componentes útiles junto con los valores predeterminados prescritos
- Actuar como una forma abreviada de tipos de entidades complejas pero comunes (por ejemplo, `<a-sky>`)
- Proporciona una interfaz familiar para principiantes, ya que A-Frame lleva el HTML en una nueva dirección

Bajo el capó, este `<a-box>` primitivo:

```
<a-box color="red" width="3"></a-box>
```

representa esta forma entidad-componente:

```
<a-entity geometry="primitive: box; width: 3" material="color: red"></a-entity>
```

`<a-box>` defecto la propiedad `geometry.primitive` to `box` . Y la primitiva asigna el atributo de `width` HTML a la propiedad `geometry.width` subyacente, así como el atributo de `color` HTML a la propiedad `material.color` subyacente.

Examples

Registro de un primitivo

Podemos registrar nuestras propias primitivas (es decir, registrar un elemento) utilizando `AFRAME.registerPrimitive(name, definition)` . `definition` es un objeto de JavaScript que define estas propiedades:

Propiedad	Descripción
defectoComponentes	Objeto que especifica los componentes por defecto de la primitiva. Las claves son los nombres de los componentes y los valores son los datos predeterminados de los componentes.
mapeos	Objeto que especifica la asignación entre el nombre de atributo HTML y los nombres de propiedad del componente. Siempre que se actualice el nombre del atributo HTML, la primitiva actualizará la propiedad del componente correspondiente. La propiedad del componente se define mediante una sintaxis de puntos `${componentName}.${propertyName}` .

Por ejemplo, a continuación se muestra el registro de A-Frame para la primitiva `<a-box>` :

```
var extendDeep = AFRAME.utils.extendDeep;

// The mesh mixin provides common material properties for creating mesh-based primitives.
// This makes the material component a default component and maps all the base material
properties.
var meshMixin = AFRAME.primitives.getMeshMixin();

AFRAME.registerPrimitive('a-box', extendDeep({}, meshMixin, {
  // Preset default components. These components and component properties will be attached to
the entity out-of-the-box.
  defaultComponents: {
    geometry: {primitive: 'box'}
  },

  // Defined mappings from HTML attributes to component properties (using dots as delimiters).
  // If we set `depth="5"` in HTML, then the primitive will automatically set
`geometry="depth: 5"`.
  mappings: {
    depth: 'geometry.depth',
    height: 'geometry.height',
    width: 'geometry.width'
  }
}));
```

Que podemos usar entonces

```
<a-box depth="1.5" height="1.5" width="1.5"></a-box>
```

representa esta forma entidad-componente:

```
<a-entity geometry="primitive: box; depth: 1.5; height: 1.5; width:1.5;"></a-entity>
```

Lea Primitivas en línea: https://riptutorial.com/es/aframe/topic/10074/primitivas

Capítulo 14: Raycasters (componente)

Introducción

El componente raycaster realiza pruebas generales de intersección con un raycaster. La difusión de rayos es el método de extender una línea desde un origen hacia una dirección, y verificar si esa línea se cruza con otras entidades. El componente raycaster es una envoltura encima de three.js raycaster. Comprueba las intersecciones en un cierto intervalo contra una lista de objetos, y emitirá eventos en la entidad cuando detecte intersecciones o desatasque las intersecciones (es decir, cuando el raycaster ya no esté).

Parámetros

Parámetro	Detalles
lejos	Distancia máxima bajo la cual se devuelven las entidades resultantes. No puede ser más bajo que cerca.
intervalo	Número de milisegundos para esperar entre cada prueba de intersección. Un número más bajo es mejor para actualizaciones más rápidas. Un número más alto es mejor para el rendimiento.
cerca	Distancia mínima a partir de la cual se devuelven las entidades resuilting. No puede ser inferior a 0.
objetos	Selector de consultas para seleccionar qué objetos probar para la intersección. Si no se especifica, todas las entidades serán probadas.
recursivo	Comprueba todos los hijos de los objetos si se establece. De lo contrario, solo comprueba las intersecciones con objetos raíz.

Observaciones

Eventos

Nombre	Detalles
`raycaster-intersected`	Emitido en la entidad intersectada. La entidad se está cruzando con un raycaster. El detalle del evento contendrá el, la entidad de difusión de rayos y la intersección, un objeto que contiene datos detallados sobre la intersección.
`raycaster-intersected-`	Emitido en la entidad intersectada. La entidad ya no se intersecta con un

Nombre	Detalles
`cleared`	raycaster. El detalle del evento contendrá el, la entidad de emisión de rayos.
`raycaster-intersection`	Emitido en la entidad de emisión de rayos. Raycaster se está cruzando con una o más entidades. Los detalles del evento contendrán els, una matriz con las entidades intersecadas, e intersecciones, una matriz de objetos que contienen datos detallados sobre las intersecciones.
`raycaster-intersection-cleared`	Emitido en la entidad de emisión de rayos. Raycaster ya no se intersecta con una entidad. El detalle del evento contendrá el, la entidad anteriormente intersectada.

Miembro

Miembro	Descripción
`intersectedEls`	Entidades que actualmente se cruzan con el raycaster.
`objects`	Objetos three.js para probar las intersecciones. Serán scene.children si no se especifica la propiedad de los objetos.
`raycaster`	Objeto raycaster three.js.

Methode

Método	Descripción
`refreshObjects`	Actualiza la lista de objetos basados en la propiedad de objetos para probar la intersección.

Examples

Estableciendo el Origen y la Dirección del Raycaster

El raycaster tiene un origen, donde comienza su rayo, y una dirección, donde va el rayo.

El origen del raycaster se encuentra en la posición de la entidad raycaster. Podemos cambiar el origen del transmisor de rayos estableciendo el componente de posición de la entidad de transmisor de rayos (o entidades primarias de la entidad de transmisor de rayos).

La dirección del transmisor de rayos está en "frente" de la entidad del transmisor de rayos (es decir, 0 0 -1, en el eje Z negativo). Podemos cambiar la dirección del transmisor de rayos

estableciendo el componente de rotación de la entidad de transmisor de rayos (o entidades primarias de la entidad de transmisor de rayos).

Por ejemplo, aquí está aplicando un transmisor de rayos a lo largo de una bala girada:

```
<!-- Bullet, rotated to be parallel with the ground. -->
<a-entity id="bullet" geometry="primitive: cylinder; height: 0.1" rotation="-90 0 0">
  <!-- Raycaster, targets enemies, made to be as long as the bullet, positioned to the start
of the bullet, rotated to align with the bullet. -->
  <a-entity raycaster="objects: .enemies; far: 0.1" position="0 -0.5 0" rotation="90 0 0"></a-
entity>
</a-entity>
```

Entidades de lista blanca para probar la intersección

Por lo general, no queremos probar todo en la escena en busca de intersecciones (por ejemplo, para colisiones o clics). Las intersecciones selectivas son buenas para el rendimiento, ya que limitan el número de entidades a probar para la intersección, ya que la prueba de intersección es una operación que se ejecutará más de 60 veces por segundo.

Para seleccionar o elegir las entidades que queremos probar para la intersección, podemos usar la propiedad objetos. Si esta propiedad no está definida, entonces el raycaster probará todos los objetos en la escena en busca de intersección. objetos toma un valor selector de consulta:

```
<a-entity raycaster="objects: .clickable" cursor></a-entity>
<a-entity class="clickable" geometry="primitive: box" position="1 0 0"></a-entity>
<a-entity class="not-clickable" geometry="primitive: sphere" position="-1 0 0"></a-entity>
```

Lea Raycasters (componente) en línea: https://riptutorial.com/es/aframe/topic/10036/raycasters--componente-

Capítulo 15: Sistema

Introducción

Un sistema, del patrón entidad-componente-sistema, proporciona alcance global, servicios y administración a las clases de componentes. Proporciona API públicas (métodos y propiedades) para las clases de componentes. Se puede acceder a un sistema a través del elemento de escena, y puede ayudar a los componentes a interactuar con la escena global.

Por ejemplo, el sistema de la cámara gestiona todas las entidades con el componente de la cámara, controlando qué cámara es la cámara activa.

Parámetros

Parámetro	Detalles
datos	Datos proporcionados por el esquema disponible a través de manejadores y métodos.
el	Referencia a `<a-scene>`
esquema	Se comporta igual que los esquemas de componentes. Analiza los datos.

Observaciones

Metodos

Un sistema, como un componente, define los manejadores de ciclo de vida. También puede definir métodos destinados a ser API pública.

Método	Descripción
en eso	Se llama una vez cuando se inicializa el sistema. Se utiliza para inicializar.
pausa	Llamado cuando la escena se detiene. Se utiliza para detener el comportamiento dinámico.
jugar	Llamado cuando la escena comienza o se reanuda. Se utiliza para iniciar el comportamiento dinámico.
garrapata	Si está definido, se llamará en cada tic del bucle de renderizado de la escena.

Examples

Registro de un sistema

Un sistema se registra de manera similar a un componente A-Frame.

Si el nombre del sistema coincide con el nombre de un componente, entonces el componente tendrá una referencia al sistema como este.sistema:

```
AFRAME.registerSystem('my-component', {
  schema: {},  // System schema. Parses into `this.data`.
  init: function () {
    // Called on scene initialization.
  },
  // Other handlers and methods.
});
AFRAME.registerComponent('my-component', {
  init: function () {
    console.log(this.system);
  }
});
```

Accediendo a un sistema

Se puede acceder a un sistema instanciado a través de la escena:

```
document.querySelector('a-scene').systems[systemName];
```

Se puede acceder a los prototipos registrados del sistema a través de `AFRAME.systems` .

Separación de lógica y datos

Los sistemas pueden ayudar a separar la lógica y el comportamiento de los datos si se desea. Dejamos que los sistemas se encarguen del trabajo pesado, y los componentes solo se preocupan por la administración de sus datos a través de sus métodos de ciclo de vida:

```
AFRAME.registerSystem('my-component', {
  createComplexObject: function (data) {
    // Do calculations and stuff with data.
    return new ComplexObject(data);
  }
});

AFRAME.registerComponent('my-component', {
  init: function () {
    this.myObject = null;
  },

  update: function () {
    // Do stuff with `this.data`.
    this.myObject = this.system.createComplexObject(this.data);
  }
```

```
});
```

Recopilación de todos los componentes de un sistema

No hay una API estricta para definir cómo los sistemas administran los componentes. Un patrón común es que los componentes se suscriban al sistema. El sistema entonces tiene referencias a todos sus componentes:

```
AFRAME.registerSystem('my-component', {
  init: function () {
    this.entities = [];
  },

  registerMe: function (el) {
    this.entities.push(el);
  },

  unregisterMe: function (el) {
    var index = this.entities.indexOf(el);
    this.entities.splice(index, 1);
  }
});

AFRAME.registerComponent('my-component', {
  init: function () {
    this.system.registerMe(this.el);
  },

  remove: function () {
    this.system.unregisterMe(this.el);
  }
});
```

Lea Sistema en línea: https://riptutorial.com/es/aframe/topic/10067/sistema

Capítulo 16: Sistema de gestión de activos

Introducción

A-Frame tiene un sistema de administración de activos que nos permite colocar nuestros activos en un solo lugar y precargar y almacenar en caché los activos para un mejor desempeño.

Los juegos y las ricas experiencias en 3D tradicionalmente cargan sus recursos, como modelos o texturas, antes de representar sus escenas. Esto asegura que los activos no falten visualmente, y esto es beneficioso para el rendimiento para garantizar que las escenas no intenten recuperar los activos mientras se procesan.

Observaciones

Eventos

Como `<a-assets>` y `<a-asset-item>` son *nodos* en A-Frame, emitirán el evento `loaded` cuando digan que han terminado de cargarse.

Nombre del evento	Descripción
cargado	Todos los activos se cargaron o se agotaron los activos.
se acabó el tiempo	Los activos expiraron.

Cargar el progreso en activos individuales

`<a-asset-item>`

`<a-asset-item>` invoca a three.js FileLoader . Podemos usar `<a-asset-item>` para cualquier tipo de archivo. Cuando termine, establecerá su miembro de `data` con la respuesta de texto.

Nombre del evento	Descripción
error	Fetch error. El detalle del evento contiene `xhr` con la instancia `XMLHttpRequest` .
Progreso	Emitido en progreso. El detalle del evento contiene `xhr` con la instancia `XMLHttpRequest` , `loadedBytes` y `totalBytes` .
cargado	El activo apuntado por `src` fue cargado.

`<img>`

Las imágenes son un elemento DOM estándar, por lo que podemos escuchar los eventos DOM estándar.

Nombre del evento	Descripción
carga	La imagen fue cargada.

`HTMLMediaElement`

Los activos de audio y video son `HTMLMediaElement` s. El navegador activa eventos particulares en estos elementos; señalado aquí por conveniencia:

Nombre del evento	Descripción
error	Se ha producido un error al cargar el activo.
datos cargados	Progreso.
Progreso	Progreso.

A-Frame utiliza estos eventos de progreso, comparando la cantidad de tiempo que el navegador almacenó en búfer con la duración del activo, para detectar cuándo se carga el activo.

Examples

Ejemplo de uso de activos

Colocamos activos dentro de `<a-assets>` , y colocamos `<a-assets>` dentro de `<a-scene>` . Los activos incluyen:

- `<a-asset-item>` - Activos diversos, como modelos y materiales 3D
- `<audio>` - Archivos de sonido
- `<img>` - Texturas de imagen
- `<video>` - Texturas de video

La escena no se renderizará o inicializará hasta que el navegador recupere (o produzca errores) todos los activos o el sistema de activos llegue al tiempo de espera.

Podemos definir nuestros activos en `<a-assets>` y señalar aquellos activos de nuestras entidades utilizando selectores:

```
<a-scene>
  <!-- Asset management system. -->
  <a-assets>
    <a-asset-item id="horse-obj" src="horse.obj"></a-asset-item>
    <a-asset-item id="horse-mtl" src="horse.mtl"></a-asset-item>
    <a-mixin id="giant" scale="5 5 5"></a-mixin>
    <audio id="neigh" src="neigh.mp3"></audio>
    <img id="advertisement" src="ad.png">
```

```
    <video id="kentucky-derby" src="derby.mp4"></video>
  </a-assets>

  <!-- Scene. -->
  <a-plane src="advertisement"></a-plane>
  <a-sound src="#neigh"></a-sound>
  <a-entity geometry="primitive: plane" material="src: #kentucky-derby"></a-entity>
  <a-entity mixin="giant" obj-model="obj: #horse-obj; mtl: #horse-mtl"></a-entity>
</a-scene>
```

La escena y sus entidades esperarán a cada activo (hasta el tiempo de espera) antes de inicializar y renderizar.

Intercambio de recursos de origen cruzado (CORS)

Dado que A-Frame recupera los activos que utilizan XHR , la seguridad del navegador requiere que el navegador sirva activos con encabezados de intercambio de recursos de origen cruzado (CORS) si el activo está en un dominio diferente. De lo contrario, tendríamos que alojar activos en el mismo origen que la escena.

Para algunas opciones, GitHub Pages sirve todo con encabezados CORS. Recomendamos GitHub Pages como una plataforma de implementación simple. O también puede cargar activos utilizando A-Frame + Uploadcare Uploader , un servicio que sirve archivos con los encabezados CORS establecidos.

Dado que los encabezados CORS *están* establecidos, `<a-assets>` establecerá automáticamente los atributos de origen `crossorigin` en los elementos de medios (por ejemplo, `<audio>` , `<img>` , `<video>`) si detecta que el recurso está en un dominio diferente.

Precarga de Audio y Video

Los recursos de audio y video solo bloquearán la escena si configuramos la `autoplay` o si configuramos `preload="auto"` :

```
<a-scene>
  <a-assets>
    <!-- These will not block. -->
    <audio src="blockus.mp3"></audio>
    <video src="loadofblocks.mp4"></video>

    <!-- These will block. -->
    <audio src="blocky.mp3" autoplay></audio>
    <video src="blockiscooking.mp4" preload="auto"></video>
  </a-assets>
</a-scene>
```

Establecer un tiempo de espera

Podemos establecer un tiempo de espera que cuando se alcance, la escena comenzará a renderizarse y las entidades comenzarán a inicializarse independientemente de si se han cargado todos los activos. El tiempo de espera predeterminado es de 3 segundos. Para establecer un

tiempo de espera diferente, solo pasamos el número de milisegundos al atributo de `timeout` :

Si algunos activos tardan mucho tiempo en cargarse, es posible que deseemos establecer un tiempo de espera adecuado para que el usuario no esté esperando todo el día en caso de que su red sea lenta.

```
<a-scene>
  <a-assets timeout="10000">
    <!-- You got until the count of 10 to load else the show will go on without you. -->
    <img src="bigimage.png">
  </a-asset>
</a-scene>
```

Especificando el tipo de respuesta

El contenido obtenido por `<a-asset-item>` se devolverá como texto sin formato. Si queremos usar un tipo de respuesta diferente, como `arraybuffer` , use el atributo de `response-type` `<a-asset-item>` :

```
<a-asset-item response-type="arraybuffer" src="model.gltf"></a-asset-item>
```

Cómo funciona internamente

Cada elemento en A-Frame hereda de `<a-node>` , el prototipo `AFRAME.ANode` . Control de `ANode` carga y orden de inicialización. Para que un elemento se inicialice (ya sea `<a-assets>` , `<a-asset-item>` , `<a-scene>` o `<a-entity>`), sus elementos secundarios ya deben haberse inicializado. Los nodos se inicializan de abajo hacia arriba.

`<a-assets>` es un `ANode` , y espera a que sus hijos se carguen antes de cargarse. Y dado que `<a-assets>` es un hijo de `<a-scene>` , la escena debe esperar a que se carguen todos los activos. También agregamos lógica de carga adicional a `<a-entity>` modo que esperen explícitamente a que se carguen `<a-assets>` si hemos definido `<a-assets>` .

`<a-asset-item>` utiliza `THREE.FileLoader` para recuperar archivos. three.js almacena los datos devueltos en `THREE.Cache` . Cada cargador de three.js hereda de `THREE.FileLoader` , ya sea un `ColladaLoader` , `OBJLoader` , `ImageLoader` , etc. Y todos tienen acceso y conocen el `THREE.Cache` central. Si A-Frame ya recuperó un archivo, A-Frame no intentará recuperarlo nuevamente.

Por lo tanto, dado que bloqueamos la inicialización de entidades en los activos, para cuando las entidades se carguen, todos los activos ya se habrán recuperado. Siempre y cuando definamos `<a-asset-item>` s, y la entidad esté recuperando archivos con algún tipo de `THREE.FileLoader` , el almacenamiento en caché funcionará automáticamente.

Accediendo al `FileLoader` y al Cache

Para acceder a three.js `FileLoader` si queremos escuchar más de cerca:

```
console.log(document.querySelector('a-assets').fileLoader);
```

Para acceder al caché que almacena respuestas XHR:

```
console.log(THREE.Cache);
```

Lea Sistema de gestión de activos en línea: https://riptutorial.com/es/aframe/topic/10070/sistema-de-gestion-de-activos

Creditos

S. No	Capítulos	Contributors
1	Empezando con aframe	Community, H. Pauwelyn, M.Kungla
2	Animación	M.Kungla
3	Cámara	H. Pauwelyn
4	Componentes	M.Kungla
5	Controles (componente)	H. Pauwelyn
6	cursores	H. Pauwelyn
7	Entidades	M.Kungla
8	Escena	H. Pauwelyn, M.Kungla
9	luz (componente)	H. Pauwelyn
10	mezcla-modelo (componente)	M.Kungla
11	Mixins	M.Kungla
12	modelo gltf (componente)	geekonaut
13	Primitivas	M.Kungla
14	Raycasters (componente)	H. Pauwelyn, M.Kungla
15	Sistema	M.Kungla
16	Sistema de gestión de activos	M.Kungla