Kostenloses eBook

LERNEN aframe

Free unaffiliated eBook created from
Stack Overflow contributors.

#aframe

Inhaltsverzeichnis

Über

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

Kapitel 1: Erste Schritte mit aframe

Bemerkungen

In diesem Abschnitt erhalten Sie einen Überblick über das, was ein Name ist und warum ein Entwickler es verwenden möchte.

Es sollte auch alle großen Themen innerhalb von aframe erwähnen und auf die verwandten Themen verweisen. Da die Dokumentation für aframe neu ist, müssen Sie möglicherweise erste Versionen dieser verwandten Themen erstellen.

Versionen

A-Frame 0.x

Ausführung	Veröffentlichungsdatum
0,6	2017-05-25
0,5	2017-02-10
0,4	2016-12-17
0,3	2016-08-18

Legacy-Versionen

Ausführung	Veröffentlichungsdatum
0,2	2016-03-26
0,1	2015-12-17

Examples

Fertig machen

A-Frame kann aus einer einfachen HTML-Datei entwickelt werden, ohne etwas installieren zu müssen! Eine gute Möglichkeit, A-Frame zu testen, um das Starter-Beispiel auf Glitch zu remixen, einem Online-Code-Editor, der sofort hostet und kostenlos bereitstellt. Oder erstellen Sie eine `.html` Datei und fügen Sie A-Frame in den `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>
```

Fügen Sie den JS Build hinzu

Um A-Frame in eine HTML-Datei aufzunehmen, löschen wir ein `script` Tag, das auf den CDN-Build verweist:

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

Installieren Sie ab npm

Wir können A-Frame auch über npm installieren:

```
$ npm install aframe
```

Dann können wir A-Frame in unsere Anwendung bündeln. Zum Beispiel mit Browserify oder Webpack:

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

Wenn Sie npm verwenden, können Sie die Befehlszeilenschnittstelle angle für A-Frame verwenden. angle kann eine Szenenvorlage mit einem einzigen Befehl initialisieren:

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

Eigenschaften

VR leicht gemacht

Fügen Sie einfach ein `script` Tag und `a-scene` . A-Frame kann mit 3D-Boilerplate, VR-Einstellungen und Standardsteuerungen arbeiten. Nichts zu installieren, keine Buildschritte.

Deklaratives HTML

HTML ist leicht zu lesen, zu verstehen und zu kopieren und einzufügen. A-Frame basiert auf HTML und ist für jeden zugänglich: Webentwickler, VR-Enthusiasten, Künstler, Designer, Pädagogen, Macher, Kinder.

Plattformübergreifende VR

Erstellen Sie VR-Anwendungen für Vive, Rift, Daydream, GearVR und Cardboard mit Unterstützung für alle entsprechenden Controller. Hast du kein Headset oder Controller? Kein Problem! A-Frame funktioniert weiterhin auf Standard-Desktops und Smartphones.

Entity-Component-Architektur

A-Frame ist ein leistungsstarkes Three.js-Framework, das eine deklarative, zusammensetzbare, wiederverwendbare Entity-Component-Struktur.js bereitstellt. HTML ist nur die Spitze des Eisbergs. Entwickler haben uneingeschränkten Zugriff auf JavaScript, DOM-APIs, three.js, WebVR und WebGL.

Performance

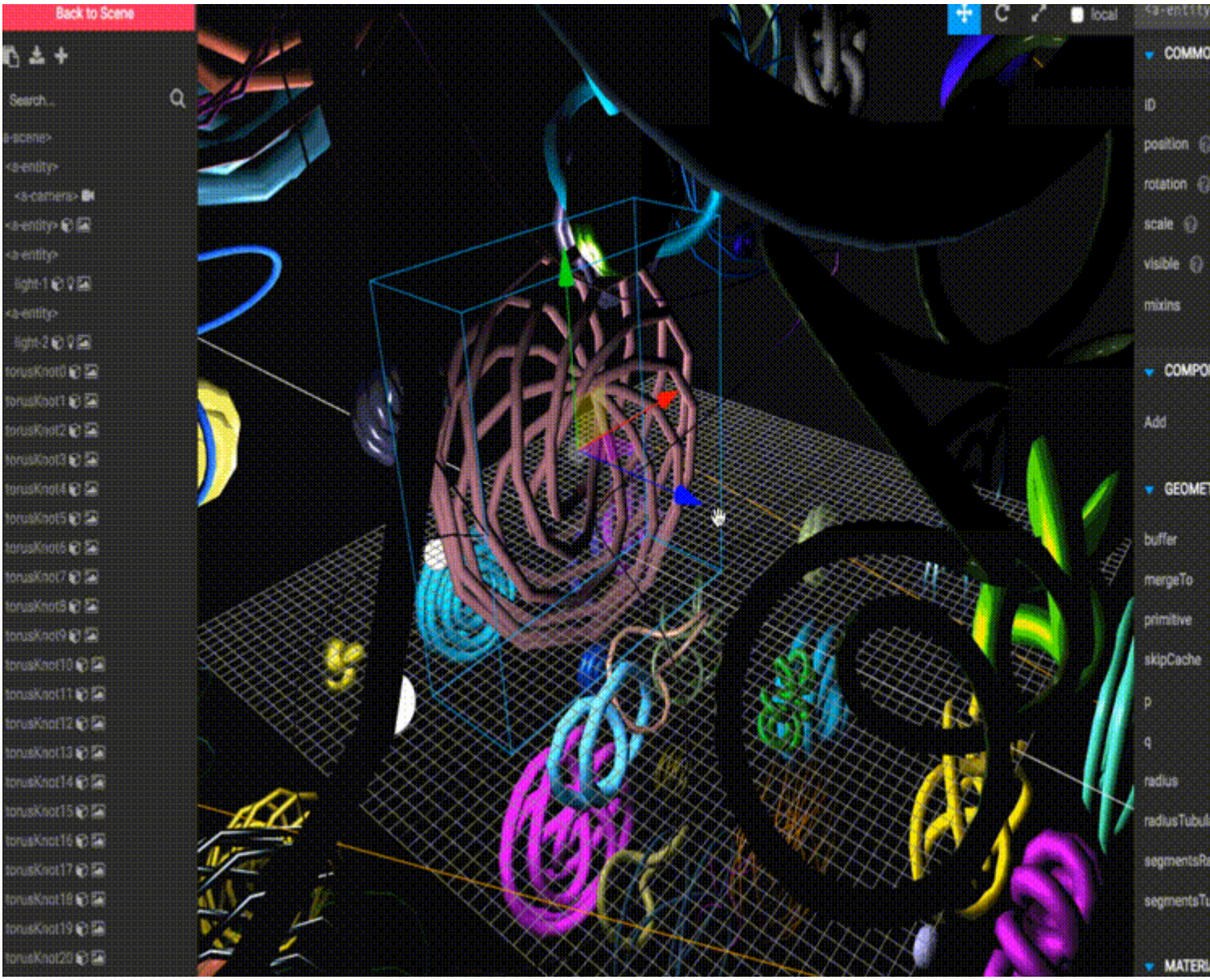
A-Frame ist von Grund auf für WebVR optimiert. Während A-Frame das DOM verwendet, berühren seine Elemente nicht die Browser-Layout-Engine. 3D-Objektaktualisierungen werden alle mit einem einzigen requestAnimationFrame-Aufruf mit wenig Aufwand im Speicher ausgeführt. Weitere Informationen finden Sie unter A-Painter, einem in A-Frame integrierten Tilt Brush-Klon, der wie native (90+ FPS) ausgeführt wird.

Werkzeug-Agnostiker

Da das Web auf der Idee des HTML basiert, ist A-Frame mit den meisten Bibliotheken, Frameworks und Tools kompatibel, darunter React, Preact, Vue.js, Angular, d3.js, Ember.js, jQuery.

Visueller Inspektor

A-Frame bietet einen praktischen integrierten visuellen 3D-Inspector. Öffne eine A-Frame-Szene, `drücke Strg` + `Alt` + `I` und fliege dahin, um hinter die Motorhaube zu `schauen` !



Registry

Nehmen Sie leistungsstarke Komponenten, die von Entwicklern veröffentlicht wurden, und fügen Sie sie direkt aus HTML hinzu. Ähnlich wie beim Unity Asset Store sammelt und kuratiert die A-Frame-Registry diese Komponenten für eine einfache Erkennung.

Komponenten

Mit den Kernkomponenten von A-Frame wie Geometrien, Materialien, Lichtern, Animationen, Modellen, Raycaster, Schatten, positioniertem Audio, Text und den Steuerelementen von Vive / Touch / Daydream / GearVR / Cardboard ist der Boden in Schwung. Machen Sie mit Community-Komponenten wie Partikelsystemen, Physik, Mehrbenutzer, Ozeane, Berge, Spracherkennung, Bewegungserfassung, Teleportation, Superhänden und Augmented Reality noch weiter.

Erste Schritte für AR

Um AR-Anwendungen im Web zu erstellen, müssen Sie eine neue Bibliothek mit dem Namen AR.js hinzufügen . Zuerst laden Sie A-Frame, gefolgt von AR.js.

Newt Sie müssen Ihre Szene mit dem A `a-scene` Tag A-Frames und dem `artoolkit` Artoolkit-`artoolkit` einrichten. Der `sourceType` muss Ihre Webcam sein. Damit wird auch die Font-Kamera Ihres Smartphones unterstützt.

Das `a-marker-camera` Tag markiert ein Bild innerhalb des aufgezeichneten Bildschirms, das ein Bild darstellt. In diesem Fall ist es `marker.png` . Wenn die Kamera diesen Marker erkennt, wird das Feld auf dem Marker angezeigt.

Nachfolgend finden Sie den Beispielcode:

```
<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>
```

Erste Schritte mit aframe online lesen: https://riptutorial.com/de/aframe/topic/10017/erste-schritte-mit-aframe

Kapitel 2: Animation

Einführung

Animationen und Übergänge in A-Frame werden mit dem Element `<a-animation>` als `<a-animation>` Element definiert. Das System basiert grob auf der Web-Animations-Spezifikation. A-Frame verwendet intern tween.js.

Bemerkungen

Attribute

Hier finden Sie eine Übersicht über Animationsattribute. Wir werden im Folgenden näher darauf eingehen.

Attribut	Beschreibung	Standardwert
Attribut	Attribut zu animieren. Um ein Komponentenattribut anzugeben, verwenden Sie die Syntax `componentName.property` (z. B. `light.intensity`).	Drehung
Start	Ereignisname, auf den gewartet werden soll, bevor die Animation beginnt.	"
verzögern	Verzögerung (in Millisekunden) oder Name des Ereignisses, auf den gewartet werden soll, bevor die Animation beginnt.	0
Richtung	Richtung der Animation (zwischen `from` und `to`). Eine von `alternate` , `alternateReverse` , `normal` , `reverse` .	normal
dur	Dauer in (Millisekunden) der Animation.	1000
Lockerung	Erleichterungsfunktion der Animation. Es gibt sehr viele zur Auswahl.	Leichtigkeit
Ende	Ereignisname, auf den gewartet werden soll, bevor die Animation angehalten wird.	"
füllen	Bestimmt den Effekt einer Animation, wenn sie nicht gerade aktiv ist. Eins von `backwards` , `both forwards` , `none` .	vorwärts
von	Startwert	Aktueller Wert.
wiederholen	Wiederholen Sie die Zählung oder `indefinite` .	0

Attribut	Beschreibung	Standardwert
zu	Endwert Muss angegeben werden	Keiner

Start

Das `begin` Attribut legt fest, wann die Animation abgespielt werden soll.

Dies kann entweder eine *Zahl sein* , die Millisekunden darstellt, die gewartet werden sollen, oder ein *Ereignisname,* auf den gewartet werden soll. Wir können beispielsweise eine Animation definieren, die 2 Sekunden wartet, bevor eine Entität skaliert wird.

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

Oder wir können eine Animation definieren, die darauf wartet, dass das übergeordnete Element ein Ereignis mit dem Namen `fade` auslöst, bevor eine Entität ausgeblendet wird.

```
<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');
```

Richtung

Das `direction` definiert, auf welche Weise zwischen dem Startwert und dem Endwert animiert werden soll.

Wenn wir eine abwechselnde Richtung definieren, wechselt die Animation zwischen den Werten `from` und `to` zu einem Jo-Jo. Wechselnde Richtungen wirken sich nur aus, wenn wir die Animation wiederholen.

Wert	Beschreibung
wechseln	Animieren Sie in geradzahligen Zyklen von `from` bis `to` . Bei ungeradzahligen Zyklen Animation von `to` bis `from`
Alternate-Reverse	Animieren Sie in ungeradzahligen Zyklen von `from` bis `to` . Bei geradzahligen Zyklen Animation von `to` bis `from`
normal	Animieren von `from` bis `to` .
umkehren	Animieren von `to` bis `from`

Lockerung

Das `easing` definiert die Beschleunigungsfunktion der Animation, die standardmäßig `ease` . Es gibt zu viele Beschleunigungsfunktionen zum Auflisten, aber wir können sie implizit erklären.

Ein möglicher Wert ist `linear` . Und die grundlegenden Beschleunigungsfunktionen sind `ease` , `ease-in` , `ease-out` und `ease-in-out` .

Dann gibt es noch mehr Gruppen von Beschleunigungsfunktionen. Die obigen grundlegenden Beschleunigungsfunktionen stellen jeder Gruppe von Beschleunigungsfunktionen ein Präfix vor. Die Gruppen von Beschleunigungsfunktionen sind `cubic` , `quad` , `quart` , `quint` , `sine` , `expo` , `circ` , `elastic` , `back` und `bounce` .

Zum Beispiel würde die `cubic` Gruppe von Beschleunigungsfunktionen aus leicht `ease-cubic` , `ease-cubic` , `ease-in-cubic` `ease-out-cubic` `ease-in-out-cubic` .

Füllen

Das `fill` definiert die Wirkung von Animationen, wenn sie nicht gerade aktiv sind. Stellen Sie sich `fill` als die Werte vor, die die Animation *vor* und / oder *nach* jedem Animationszyklus für die Entität festlegt. Nachfolgend sind die möglichen Werte für die `fill` und ihre Auswirkungen aufgeführt.

Wert	Beschreibung
rückwärts	Bevor die Animation beginnt, stellen Sie den Startwert auf das `from` Wert.
beide	Kombinieren Sie die Effekte der Rückwärtsfüllung und der Rückwärtsfüllung.
vorwärts	Nachdem die Animation abgeschlossen ist, bleibt der endgültige Wert auf dem `to` Wert. Die Standardfüllung.
keiner	Stellen Sie vor Beginn der Animation den Startwert auf den Anfangswert ein. Setzen Sie den Wert nach Abschluss der Animation auf den Anfangswert zurück.

Wiederholen

Das `repeat` definiert, wie oft die Animation wiederholt wird. Wir nennen jede Wiederholung der Animation einen Zyklus. Wiederholen kann entweder eine Zahl sein, die bei jedem Animationszyklus heruntergezählt wird, bis `0` erreicht ist, an dem Punkt, an dem die Animation endet, oder wir können die `repeat` auf `indefinite` und die Animation wird kontinuierlich `repeat` , bis die Animation manuell entfernt oder gestoppt wird.

VERANSTALTUNGEN

Das Element `<a-animation>` gibt einige Ereignisse aus.

Veranstaltungsname	Beschreibung
Animationsende	Wird gesendet, wenn die Animation beendet ist. Bei Wiederholungen wird ausgegeben, wenn der Wiederholungszähler `0` erreicht. Nicht für unbestimmte Wiederholungen ausgegeben.
Animationsstart	Wird sofort gesendet, wenn die Animation abgespielt wird.

Examples

Beispielanimationen

Einführungsbeispiel: Um einen 5-Meter-Orbit auf einer Entität um die Y-Achse zu definieren, der 10 Sekunden dauert, können wir die Position verschieben und die Rotation animieren. Diese Animation beginnt mit der anfänglichen Drehung um die Y-Achse von 0 Grad und geht um 360 Grad herum. Es ist mit einer Dauer von 10000 Millisekunden definiert, behält den Endwert in jedem Zyklus der Animation bei und wiederholt Endlosschleifen.

```
<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>
```

Animieren verschiedener Arten von Eigenschaften

Das Animationssystem von A-Frame kann verschiedene Arten von Eigenschaften animieren.

vec3 Eigenschaften

A-Frame verfügt über standardmäßige `vec3` Komponenten (dh `position` , `rotation` und `scale`). Diese Komponenten bestehen aus drei Faktoren: X, Y und Z. Wir können drei durch Leerzeichen getrennte Zahlen an die Attribute `from` und übergeben `to` genau wie wir sie für eine Entität definieren würden. In diesem Fall geht das Animationssystem davon aus, dass wir einen `vec3` Wert animieren.

Wenn Sie beispielsweise eine Entität animieren möchten, die von einer Stelle zur anderen geht, können Sie die `position` animieren.

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

Boolesche Eigenschaften

A-Frame verfügt über Standardkomponenten, die einen einzigen booleschen Wert akzeptieren.

Boolesche Werte können auch "animiert" werden, indem der Boolesche Wert am Ende jedes Animationszyklus umgedreht wird.

Wir können beispielsweise eine Animation definieren, die die Sichtbarkeit einer Entität nach 5 Sekunden deaktiviert.

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

Numerische Eigenschaften

Wir können auch numerische Attribute animieren. Zum Beispiel können wir die Intensität des Lichtprimitivs animieren.

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

Farbeigenschaften

Wir können jede Komponenteneigenschaft mit einem Farbtyp animieren. Zum Beispiel können wir eine Box von weiß nach rot animieren.

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

Komponenteneigenschaften

Wir können eine bestimmte Eigenschaft einer Komponente mit mehreren Eigenschaften animieren. Dazu wählen wir die Komponenteneigenschaft mit der Punktsyntax aus: `componentName.propertyName` .

Um beispielsweise den oberen Radius eines Kegels zu animieren, können Sie den `radiusTop` Wert mit `geometry.radiusTop` auswählen.

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

Animation online lesen: https://riptutorial.com/de/aframe/topic/10071/animation--a-animation-

Kapitel 3: Asset Management System

Einführung

A-Frame verfügt über ein Asset-Management-System, mit dem wir unsere Assets an einem Ort platzieren und Assets für eine bessere Leistung vorladen und zwischenspeichern können.

Spiele und umfangreiche 3D-Erlebnisse laden ihre Assets (z. B. Modelle oder Texturen) traditionell vor dem Rendern ihrer Szenen auf. Dadurch wird sichergestellt, dass Assets nicht visuell fehlen. Dies ist für die Leistung von Vorteil, um sicherzustellen, dass Szenen beim Rendern nicht versuchen, Assets abzurufen.

Bemerkungen

Veranstaltungen

Da `<a-assets>` und `<a-asset-item>` *Knoten* in A-Frame sind, werden sie das `loaded` Ereignis ausgeben `loaded` wenn sie das Laden beendet haben.

Veranstaltungsname	Beschreibung
geladen	Alle Assets wurden geladen oder das Zeitlimit überschritten.
Auszeit	Zeitüberschreitung der Vermögenswerte

Fortschritt der einzelnen Assets laden

`<a-asset-item>`

`<a-asset-item>` ruft den FileLoader von three.js auf . Wir können `<a-asset-item>` für jeden Dateityp verwenden. Wenn Sie fertig sind, wird das `data` mit der Textantwort festgelegt.

Veranstaltungsname	Beschreibung
Error	Fehler beim Abrufen Ereignisdetails enthalten `xhr` mit der Instanz `XMLHttpRequest` .
Fortschritt	Über den Fortschritt informiert. Ereignisdetails enthalten `xhr` mit der Instanz `XMLHttpRequest` , `loadedBytes` und `totalBytes` .
geladen	Asset, auf das von `src` verwiesen wurde, wurde geladen.

`<img>`

Bilder sind ein Standard-DOM-Element, sodass wir uns die Standard-DOM-Ereignisse anhören können.

Veranstaltungsname	Beschreibung
Belastung	Bild wurde geladen.

`HTMLMediaElement`

Audio- und Video-Assets sind `HTMLMediaElement` . Der Browser löst bestimmte Ereignisse auf diesen Elementen aus. hier der Einfachheit halber notiert:

Veranstaltungsname	Beschreibung
Error	Beim Laden des Assets ist ein Fehler aufgetreten.
loadeddata	Fortschritt.
Fortschritt	Fortschritt.

A-Frame verwendet diese Fortschrittsereignisse und vergleicht, wie viel Zeit der Browser mit der Dauer des Assets puffert, um festzustellen, wann das Asset geladen wird.

Examples

Beispiel für die Verwendung von Assets

Wir platzieren Assets innerhalb von `<a-assets>` und wir legen `<a-assets>` innerhalb von `<a-scene>` . Assets beinhalten:

- `<a-asset-item>` - Verschiedene Assets wie 3D-Modelle und -Materialien
- `<audio>` - Audiodateien
- `<img>` - Bildtexturen
- `<video>` - `<video>`

Die Szene wird erst gerendert oder initialisiert, wenn der Browser alle Assets abruft (oder Fehler ausgibt) oder das Asset-System das Timeout erreicht.

Wir können unsere Vermögenswerte in `<a-assets>` und mithilfe von Selektoren auf diese Assets unserer Unternehmen verweisen:

```
<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>
```

Die Szene und ihre Entitäten warten auf jedes Asset (bis zum Timeout), bevor es initialisiert und gerendert wird.

Ressourcenübergreifende Ressourcenfreigabe (CORS)

Da A-Frame Assets mithilfe von XHRs abruft , muss der Browser für die Browsersicherheit Assets mit CORS-Headern bereitstellen , wenn sich das Asset in einer anderen Domäne befindet. Andernfalls müssten wir Assets mit demselben Ursprung wie die Szene hosten.

Für einige Optionen liefert GitHub Pages alles mit CORS-Headern. Wir empfehlen GitHub Pages als einfache Implementierungsplattform. Sie können Assets auch mit dem A-Frame + Uploadcare Uploader hochladen , einem Dienst, der Dateien mit eingestellten CORS-Headern bereitstellt .

Da CORS-Header festgelegt *sind* , werden `<a-assets>` `crossorigin` Attribute für `crossorigin` (z. B. `<audio>` , `<img>` , `<video>`) automatisch festgelegt, wenn festgestellt wird, dass sich die Ressource in einer anderen Domäne befindet.

Audio und Video vorladen

Audio- und Video-Assets blockieren die Szene nur, wenn wir `autoplay` oder `preload="auto"` einstellen:

```
<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>
```

Timeout einstellen

Wir können ein Timeout festlegen, nach dessen Erreichen die Szene mit dem Rendern beginnt und die Entitäten initialisiert werden, unabhängig davon, ob alle Assets geladen wurden. Das Standard-Timeout beträgt 3 Sekunden. Um ein anderes Timeout einzustellen, übergeben wir einfach die Anzahl der Millisekunden an das `timeout` Attribut:

Wenn das Laden einiger Assets sehr lange dauert, möchten wir möglicherweise ein geeignetes Timeout festlegen, sodass der Benutzer nicht den ganzen Tag wartet, falls sein Netzwerk langsam ist.

```
<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>
```

Angabe des Antworttyps

Von `<a-asset-item>` abgerufene Inhalte werden als reiner Text zurückgegeben. Wenn Sie einen anderen Antworttyp verwenden `arraybuffer` , z. B. `arraybuffer` , verwenden Sie das `response-type` Attribut von `<a-asset-item>` :

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

Wie es intern funktioniert

Jedes Element in A-Frame erbt von `<a-node>` , dem Prototyp `AFRAME.ANode` . `ANode` steuert die Lade- und Initialisierungsreihenfolge. Damit ein Element initialisiert werden kann (entweder `<a-assets>` , `<a-asset-item>` , `<a-scene>` oder `<a-entity>`), müssen die untergeordneten Elemente bereits initialisiert sein. Knoten initialisieren von unten nach oben.

`<a-assets>` ist ein `ANode` und wartet, bis seine Kinder geladen werden, bevor es geladen wird. Und da `<a-assets>` ein Kind von `<a-scene>` , muss die Szene effektiv warten, bis alle Assets geladen sind. Wir haben auch eine zusätzliche Ladelogik zu `<a-entity>` hinzugefügt, sodass sie explizit auf das Laden von `<a-assets>` warten, wenn `<a-assets>` .

`<a-asset-item>` verwendet `THREE.FileLoader` , um Dateien abzurufen. three.js speichert die zurückgegebenen Daten in `THREE.Cache` . Jeder drei.js-Loader erbt von `THREE.FileLoader` , unabhängig `THREE.FileLoader` , ob es sich um `ColladaLoader` , `OBJLoader` , `ImageLoader` usw. handelt. Alle haben Zugriff und kennen den zentralen `THREE.Cache` . Wenn A-Frame bereits eine Datei abgerufen hat, versucht A-Frame nicht, sie erneut abzurufen.

Da wir also die Entitätsinitialisierung für Assets blockieren, sind alle Assets zum Zeitpunkt, zu dem Entities geladen werden, bereits abgerufen worden. Solange wir `<a-asset-item>` s definieren und die Entität Dateien mit dem Formular `THREE.FileLoader` , `THREE.FileLoader` Caching automatisch.

Zugriff auf `FileLoader` und Cache

Um auf den three.js FileLoader `FileLoader` wenn wir genauer `FileLoader` wollen:

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

So greifen Sie auf den Cache zu, in dem XHR-Antworten gespeichert sind:

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

Asset Management System online lesen: https://riptutorial.com/de/aframe/topic/10070/asset-management-system

Kapitel 4: Cursor

Einführung

Mit der Cursor-Komponente können wir durch Klicken und Betrachten mit Entitäten interagieren.

Syntax

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

Parameter

Eigentum	Beschreibung
Sicherung	Gibt an, ob der Cursor auf einer Sicherung basiert. Standardwert: " `false` auf dem Desktop, " `true` auf dem Handy
fuseTimeout	Wartezeit (in Millisekunden) bis zum Auslösen eines Sicherungs-Klickereignisses. Standardwert: 1500

Bemerkungen

Der Cursor ist eine spezifische Anwendung der Raycaster-Komponente

- Hört auf Mausklicks und blickbasierte Sicherungen
- Erfasst nur die erste durchschnittene Entität
- Gibt spezielle Maus- und Hover-Ereignisse aus (z. B. in Bezug auf Maus runter / runter / Enter / Verlassen)
- Hat mehr Zustände zum Schweben.

Wenn Sie mit der Maus klicken, wird die nächstliegende sichtbare Entität, die den Cursor überschneidet (sofern vorhanden), ein Klickereignis ausgeben. Beachten Sie, dass die Cursor-Komponente nur das Raycasting-Verhalten anwendet. Um dem Cursor eine Form oder ein Aussehen zu verleihen, können Sie die Geometrie- und Materialkomponenten anwenden.

Veranstaltungen

Veranstaltung	Beschreibung
klicken	Wird sowohl auf den Cursor als auch auf die überlappende Entität gesendet, wenn auf eine aktuell überlappende Entität geklickt wird

Veranstaltung	Beschreibung
	(entweder mit der Maus oder mit einer Sicherung).
Verschmelzen	Wird sowohl für den Cursor als auch für die geschnittene Entität ausgegeben, wenn der auf der Sicherung basierende Cursor mit dem Countdown beginnt.
Maus nach unten	Wird sowohl auf dem Cursor als auch auf der durchschnittenen Entität (falls vorhanden) bei Mausedown im Zeichenflächenelement gesendet.
Mouseenter	Wird sowohl auf den Cursor als auch auf die geschnittene Entität gesendet (sofern vorhanden), wenn der Cursor eine Entität kreuzt.
mouseleave	Wird sowohl auf dem Cursor als auch auf der geschnittenen Entität (falls vorhanden) ausgegeben, wenn der Cursor sich nicht mehr mit der zuvor verschobenen Entität schneidet.
Mouseup	Wird sowohl auf dem Cursor als auch auf dem überlappenden Element (falls vorhanden) beim Mouseup auf dem Zeichenelement ausgegeben.

Examples

Standardcursor

Zum Beispiel können wir einen ringförmigen Cursor erstellen, der in der Mitte des Bildschirms fixiert ist. Um den Cursor so auf dem Bildschirm zu fixieren, dass er immer sichtbar ist, egal wo wir hinschauen, platzieren wir ihn als Kind der aktiven Kameraeinheit. Wir ziehen es vor die Kamera, indem wir es auf der negativen Z-Achse platzieren. Wenn der Cursor auf die Box klickt, können wir das Klickereignis anhören.

```
<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);
    });
```

```
  }
});
```

Blickbasierte Interaktionen mit der Cursor-Komponente

Wir werden zuerst auf blickbasierte Interaktionen eingehen. Blickbasierte Interaktionen sind darauf angewiesen, dass wir den Kopf drehen und Objekte betrachten, um mit ihnen zu interagieren. Diese Art der Interaktion gilt für Headsets ohne Controller. Selbst bei einem Nur-Rotations-Controller (Daydream, GearVR) ist die Interaktion immer noch ähnlich. Da A-Frame standardmäßig Steuerelemente zum Ziehen per Mauszugriff bietet, kann die Blickrichtung auf dem Desktop verwendet werden, um eine Vorschau der Interaktion durch Ziehen der Kameradrehung anzuzeigen.

Um eine blickbasierte Interaktion hinzuzufügen, müssen Sie eine Komponente hinzufügen oder implementieren. A-Frame wird mit einer Cursor-Komponente geliefert, die eine blickbasierte Interaktion ermöglicht, wenn sie mit der Kamera verbunden ist:

1. Entität explizit definieren. Zuvor hatte A-Frame die Standardkamera bereitgestellt.
2. Fügen Sie unter dem Kameraobjekt `a-cursor` Element als untergeordnetes Element hinzu.
3. Konfigurieren Sie optional den vom Cursor verwendeten Raycaster.

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

ein Cursor-Primitiv

Das Cursor-Grundelement ist ein Fadenkreuz, das das Klicken und grundlegende Interaktivität mit einer Szene auf Geräten ohne Handcontroller ermöglicht. Das Standardaussehen ist eine Ringgeometrie. Der Cursor wird normalerweise als untergeordnetes Element der Kamera platziert.

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

Sicherungsbasierter Cursor

Wird auch als blickbasierter Cursor bezeichnet. Wenn wir den Cursor auf fuse-basiert setzen, löst der Cursor einen Klick aus, wenn der Benutzer eine bestimmte Zeit lang auf eine Entität blickt. Stellen Sie sich einen Laser vor, der am Kopf des Benutzers befestigt ist, und der Laser ragt in die Szene hinein. Wenn der Benutzer lange genug auf eine Entität starrt (z. B. fuseTimeout), wird der Cursor einen Klick auslösen.

Der Vorteil der auf Sicherungen basierenden Interaktionen für VR besteht darin, dass keine zusätzlichen Eingabegeräte außer dem Headset erforderlich sind. Der fuse-basierte Cursor ist hauptsächlich für Google Cardboard-Anwendungen gedacht. Der Nachteil von auf Sicherungen basierenden Interaktionen besteht darin, dass der Benutzer den Kopf viel drehen muss.

Konfigurieren des Cursors über die Raycaster-Komponente

Der Cursor baut auf der Raycaster-Komponente auf und ist davon abhängig. Wenn Sie die Raycasting-Teile des Cursors anpassen möchten, können Sie die Eigenschaften der Raycaster-Komponente ändern. Angenommen, wir möchten eine maximale Entfernung festlegen, weniger häufig nach Schnittpunkten suchen und festlegen, welche Objekte anklickbar sind:

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

Visuelles Feedback hinzufügen

Um dem Cursor visuelles Feedback hinzuzufügen, um anzuzeigen, wenn der Cursor klickt oder verschmilzt, können wir das Animationssystem verwenden. Wenn der Cursor die Entität schneidet, wird ein Ereignis ausgegeben und das Animationssystem nimmt das Ereignis mit dem begin-Attribut auf:

```
<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>
```

Mauszeiger

Hinweis: Für dieses Beispiel müssen Sie ein externes npm-Paket hinzufügen.

Wenn Sie einen Mauszeiger Ihres Computers verwenden möchten, müssen Sie eine `aframe-mouse-cursor-component` hinzufügen. Danach müssen Sie das Skript mit diesem Code einschließen:

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

// or this

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

Auf Ihrer Kamera müssen Sie die `mouse-cursor` Komponente hinzufügen.

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

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

Cursor online lesen: https://riptutorial.com/de/aframe/topic/10180/cursor

Kapitel 5: Entitäten

Einführung

A-Frame repräsentiert eine Entität über das Element `<a-entity>` . Wie im Entitäts-Komponentensystem-Muster definiert, sind Entitäten Platzhalterobjekte, in die Komponenten eingefügt werden, um ihnen Aussehen, Verhalten und Funktionalität zu bieten.

Syntax

- `<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:`

Parameter

Parameter	Einzelheiten
Komponenten	`<a-entity>.components` ist ein Objekt, das mit der Entität verbunden ist. Dies gibt uns Zugriff auf die Komponenten der Entität, einschließlich Daten, Status und Methoden jeder Komponente.
spielt	Ob die Entität aktiv ist und spielt. Wenn wir die Entität pausieren, wird isPlaying false.
object3D	`<a-entity>.object3D` ist eine Referenz auf die Object.D-Darstellung von drei.js der Entität. Insbesondere ist object3D ein `THREE.Group` Objekt, das verschiedene Typen von `THREE.Object3D` enthalten kann, z. B. Kameras, Netze, Lichter oder Sounds:
object3DMap	Die object3DMap eines Objekts ist ein Objekt, das Zugriff auf die verschiedenen Typen von THREE.Object3Ds (z. B. Kamera, Maschen, Lichter, Töne) bietet, die von Komponenten festgelegt wurden.
sceneEl	Eine Entität hat eine Referenz auf ihr Szenenelement.

Bemerkungen

Methoden

addState (stateName)

addState überträgt einen Zustand auf die Entität. Dadurch wird das **stateadded-** Ereignis **ausgegeben** , und wir können den Status mit **.is** auf Existenz überprüfen:

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

---

emittieren (Name, Detail, Blasen)

emit gibt ein benutzerdefiniertes DOM-Ereignis für die Entität aus. Zum Beispiel können wir ein Ereignis ausgeben, um eine Animation auszulösen:

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

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

Wir können auch Ereignisdetails oder Daten als zweites Argument übergeben:

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

Das Ereignis wird standardmäßig angezeigt. Wir können sagen, dass es nicht blubbern soll, indem Sie false für bubble übergeben:

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

---

flushToDOM (rekursiv)

flushToDOM serialisiert die Daten einer Entität manuell und aktualisiert das DOM.

---

getAttribute (Komponentenname)

getAttribute ruft **geparste** Komponentendaten (einschließlich Mixins und Standardwerte) ab.

```
// <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}
```

Wenn Komponentenname nicht der Name einer registrierten Komponente ist, verhält sich getAttribute wie gewöhnlich:

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

---

getDOMAttribute (Komponentenname)

getDOMAttribute ruft nur **geparste** Komponentendaten ab, die explizit im DOM oder über **setAttribute definiert sind** . Wenn **Komponentenname** der Name einer registrierten Komponente ist, gibt **getDOMAttribute** nur die im HTML definierten Komponentendaten als **geparstes** Objekt zurück. **getDOMAttribute** für Komponenten ist die Teilform von **getAttribute,** da die zurückgegebenen Komponentendaten keine angewendeten Mixins oder Standardwerte enthalten:

Vergleichen Sie die Ausgabe des obigen Beispiels für **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 (Typ)

getObject3D sucht ein **untergeordnetes THREE.Object3D, auf** das in **object3DMap** vom **Typ verwiesen wird** .

```
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 (Typ, Konstruktor)

Wenn für die Entität kein **THREE.Object3D** unter **type registriert** ist, registriert **getOrCreateObject3D** ein instantiiertes **THREE.Object3D** mit dem übergebenen **Konstruktor** . Wenn für die Entität ein **THREE.Object3D- Typ** unter **Typ** registriert ist, fungiert **getOrCreateObject3D** als **getObject3D** :

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

Pause ()

pause () stoppt dynamisches Verhalten, wie es von Animationen und Komponenten definiert wird. Wenn wir eine Entität unterbrechen, stoppt sie ihre Animationen und ruft **Component.pause ()** für jede ihrer Komponenten auf. Die Komponenten entscheiden sich für die Implementierung des Pausenverhaltens, wodurch häufig Ereignis-Listener entfernt werden. Eine Entität ruft pause () für ihre untergeordneten Entitäten auf, wenn eine Entität angehalten wird.

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

Beispielsweise entfernt die Look-Control-Komponente bei Pause Ereignis-Handler, die auf Eingaben warten.

---

abspielen ()

play () startet jedes dynamische Verhalten, das durch Animationen und Komponenten definiert wird. Dies wird automatisch aufgerufen, wenn das DOM eine Entität anfügt. Wenn eine Entität **play ()** startet, ruft die Entity **play ()** für ihre untergeordneten Entitäten auf.

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

Zum Beispiel beginnt die Soundkomponente beim Abspielen des Sounds.

---

setAttribute (Komponentenname, Wert, [Eigenschaftswert | Clobber])

Wenn **Komponentenname** nicht der Name einer registrierten Komponente ist oder die Komponente eine Komponente mit einer einzigen Eigenschaft ist, verhält sich **setAttribute** wie **folgt** :

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

Wenn **Komponentenname** der Name einer registrierten Komponente ist, kann sie die spezielle Analyse für den Wert verarbeiten. Die **Positionskomponente** ist beispielsweise eine **Komponente** mit einer einzigen Eigenschaft, ihr Parser für Eigenschaftstyp ermöglicht jedoch die Übernahme eines Objekts:

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

---

setObject3D (Typ, Objekt)

setObject3D registriert das übergebene **Objekt** , ein **THREE.Object3D** , als **type** unter der **object3DMap** der Entität. A-Frame fügt **obj** als **untergeordnetes Objekt** des Entity- **Objekts object3D hinzu** .

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

---

removeAttribute (Komponentenname, Eigenschaftsname)

Wenn **Komponentenname** der Name einer registrierten Komponente ist und das Attribut aus dem DOM entfernt wird, **trennt removeAttribute** auch die Komponente von der Entität und ruft die **Entfernungszyklusmethode** der Komponente auf.

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

Wenn **propertyName** angegeben ist, setzt **removeAttribute** den Eigenschaftswert der von propertyName angegebenen **Eigenschaft** auf den Standardwert der Eigenschaft zurück:

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

---

removeObject3D (Typ)

removeObject3D entfernt das durch **type** angegebene Objekt aus der **THREE.Group** der Entität und somit aus der Szene. Dadurch wird **object3DMap** der Entität **aktualisiert** und der Wert des **Typenschlüssels** auf **null gesetzt** . Dies wird im Allgemeinen von einer Komponente aus aufgerufen, häufig im Remove-Handler:

```
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 zeigt einen Zustand aus der Entität an. Dadurch wird das **stateremoved -Ereignis**

ausgegeben , und wir können den Status des Entfernens mit **.is** überprüfen:

```
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
```

VERANSTALTUNGEN

Veranstaltungsname	Beschreibung
mit Kindern verbunden	Eine untergeordnete Entität wurde an die Entität angehängt.
Kinderhaus	Eine untergeordnete Entität wurde von der Entität getrennt.
Komponente geändert	Eine der Komponenten der Entität wurde geändert.
Komponenteinit	Eine Komponente des Unternehmens wurde initialisiert.
Komponententfernt	Eine Komponente der Entität wurde entfernt.
geladen	Das Unternehmen hat seine Komponenten angehängt und initialisiert.
object3dset	THREE.Object3D wurde mit setObject3D (Name) für die Entität festgelegt. Ereignisdetails enthalten den Namen, der zur Einstellung in object3DMap verwendet wird.
Pause	Die Entität ist jetzt inaktiv und pausiert in Bezug auf dynamisches Verhalten.
abspielen	Die Entität ist jetzt aktiv und spielt in Bezug auf dynamisches Verhalten.
Stateadded	Die Entität erhielt einen neuen Staat.
stateremoved	Die Entität hat keinen bestimmten Zustand mehr.
Schema geändert	Das Schema einer Komponente wurde geändert.

VERANSTALTUNGSDETAILS

Nachfolgend finden Sie die Details zu den einzelnen Ereignissen:

Veranstaltungsname	Eigentum	Beschreibung
mit Kindern verbunden	el	Verweis auf das angefügte untergeordnete Element.
Komponente geändert	Name	Name der Komponente, deren Daten geändert wurden.
	Ich würde	ID der Komponente, deren Daten geändert wurden.
	neue Daten	Neue Daten der Komponente, nachdem sie geändert wurden.
	alteDaten	Vorherige Daten der Komponente, bevor sie geändert wurden.
Komponenteinitialisiert	Name	Name der Komponente, die initialisiert wurde.
	Ich würde	ID der Komponente, deren Daten geändert wurden.
	Daten	Komponentendaten.
Komponententfernt	Name	Name der Komponente, die entfernt wurde.
	Ich würde	ID der Komponente, die entfernt wurde.
Stateadded	Zustand	Der angehängte Zustand (String).
stateremoved	Zustand	Der Zustand, der getrennt wurde (Zeichenfolge).
Schema geändert	Komponente	Name der Komponente, deren Schema geändert wurde.

Examples

Auf Komponentenänderungen warten

Wir können das Ereignis " **componentchanged" verwenden** , um Änderungen an der Entität zu überwachen:

```
entity.addEventListener('componentchanged', function (evt) {
```

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

Auf untergeordnete Elemente warten, die angefügt und getrennt werden

Wir können die **Child-Attached -** Ereignisse und **Child-Detached-** Ereignisse verwenden, um zu überwachen, wann die Szene eine Entität anfügt oder entfernt:

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

Daten der Entität mit mehreren Eigenschaften (setAttribute)

Aktualisieren von Komponentendaten für mehrere Eigenschaften

Um Komponentendaten für eine Komponente mit mehreren Eigenschaften zu aktualisieren, können wir den Namen einer registrierten Komponente als **Komponentenname** und ein Objekt mit Eigenschaften als **Wert übergeben** . Eine Zeichenfolge ist ebenfalls zulässig (z. B. **type: spot; distance: 30**), aber Objekte sparen A-Frame etwas Arbeit beim Parsen:

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

Um einzelne Eigenschaften für eine Komponente mit mehreren Eigenschaften zu aktualisieren, können Sie den Namen der registrierten Komponente als **Komponentenname** , einen Eigenschaftennamen als zweites Argument und den Eigenschaftswert als drittes Argument übergeben:

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

Beachten Sie, dass sich Array-Eigenschaftstypen eindeutig verhalten:

- Arrays sind veränderbar. Sie werden per Referenz zugewiesen, sodass Änderungen an Arrays für die Komponente sichtbar sind.
- Aktualisierungen der Eigenschaften des Array-Typs lösen weder die Aktualisierungsmethode der Komponente aus, noch werden Ereignisse ausgelöst.

Aktualisieren von Komponentendaten für mehrere Eigenschaften

Wenn **true** als drittes Argument an **.setAttribute übergeben wird** , werden nicht angegebene Eigenschaften zurückgesetzt und gesäubert:

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

Entität abrufen

Wir können eine Entität einfach mithilfe von DOM-APIs abrufen.

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

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

Entitätskomponenten abrufen

Wenn wir beispielsweise das Three.js-Kameraobjekt oder das Materialobjekt einer Entität packen wollten, könnten wir in ihre Komponenten greifen

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

Wenn eine Komponente eine API verfügbar macht, können wir ihre Methoden aufrufen:

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

Entitäten online lesen: https://riptutorial.com/de/aframe/topic/10066/entitaten--a-entity-

Kapitel 6: gltf-modell (komponente)

Einführung

Die gltf-Modellkomponente ermöglicht die Verwendung von 3D-Modellen im glTF-Format mit A-Frame. glTF ist ein Khronos-Standard für effiziente 3D-Modelle mit voller Szene und ist für die Verwendung im Internet optimiert.

Syntax

- `<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>`

Parameter

Parameter	Einzelheiten
`url(...)`	lädt das glTF-Modell von der in `url()` URL.
`#example`	lädt das `<a-asset-item>` mit dem ID- `example`

Examples

Laden eines glTF-Modells über 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>
```

Laden eines gltf-Modells über das Asset-System

```
<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>
```

gltf-modell (komponente) online lesen: https://riptutorial.com/de/aframe/topic/10758/gltf-modell--komponente-

Kapitel 7: Kamera

Einführung

Die Kamerakomponente definiert, aus welcher Perspektive der Benutzer die Szene betrachtet. Die Kamera ist in der Regel mit Steuerungskomponenten gekoppelt, mit denen Eingabegeräte die Kamera bewegen und drehen können.

Syntax

- <A-Entity-Kamera> </ A-Entity>
- <a-camera> </ a-camera>

Parameter

Eigentum	Beschreibung
aktiv	Ob die Kamera die aktive Kamera in einer Szene mit mehr als einer Kamera ist.
weit	Kamera frustum weit Ausschnittsfläche.
fov	Sichtfeld (in Grad).
nahe	Kamera-Frustum nahe Ausschnittsfläche.
userHeight	Wie viel Höhe muss der Kamera hinzugefügt werden, wenn sie sich nicht im VR-Modus befindet. Die Standardkamera hat diese Einstellung auf 1,6 (Meter, um die durchschnittliche Augenhöhe darzustellen).
Zoomen	Zoomfaktor der Kamera.

Bemerkungen

Wenn Sie sich nicht im VR-Modus befinden, übersetzt `userHeight` die Kamera bis zur ungefähren Höhe des menschlichen Auges. Die eingespritzte Kamera hat diese Einstellung auf 1,6 (Meter). Bei der Eingabe von VR wird dieser Höhenversatz *entfernt* , sodass die vom VR-Headset zurückgegebene absolute Position verwendet wurde. Der Versatz ist praktisch für Erlebnisse, die sowohl innerhalb als auch außerhalb der VR funktionieren, und das Erleben von Erlebnissen von einem Desktop-Bildschirm aus betrachtet, anstatt den Boden abzuschneiden, wenn das Headset auf dem Boden lag.

Beim Verlassen der VR-Kamera setzt die Kamera ihre Rotation wieder ein, bevor sie in die VR-Kamera eintritt. Wenn Sie VR beenden, ist die Drehung der Kamera für einen Desktop-Bildschirm wieder normal.

Examples

Standardkamera

Eine Kamera, die sich auf der durchschnittlichen Höhe des menschlichen Auges befindet (1,6 Meter oder 1,75 Yard).

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

Aktive Kamera wechseln

Wenn die aktive Eigenschaft umgeschaltet wird, benachrichtigt die Komponente das Kamerasystem, die aktuell vom Renderer verwendete Kamera zu ändern:

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

Befestigen von Objekten an der Kamera

Um Objekte auf der Kamera so zu fixieren, dass sie unabhängig vom Standort des Benutzers sichtbar bleiben, können Sie diese Objekte als untergeordnetes Element der Kamera anhängen. Anwendungsfälle können ein Head-Up-Display (HUD) sein.

```
<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>
```

Beachten Sie, dass Sie HUDs sparsam verwenden sollten, da sie im VR zu Reizungen und Augenbelastungen führen. Erwägen Sie, Menüs in das Gewebe der Welt selbst zu integrieren. Wenn Sie ein HUD erstellen, stellen Sie sicher, dass sich das HUD mehr in der Mitte des Sichtfelds befindet, sodass der Benutzer seine Augen nicht dazu zwingen muss, es zu lesen.

ein Kamera-Primitiv

Das Kamera-Grundelement bestimmt, was der Benutzer sieht. Wir können das Ansichtsfenster ändern, indem Sie die Position und Drehung der Kameraeinheit ändern.

Beachten Sie, dass der Ursprung der Kamera standardmäßig im Desktop-Modus 0 1.6 0 und im VR-Modus 0 0 0 beträgt. Informieren Sie sich über die Eigenschaft camera.userHeight.

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

Kamera manuell positionieren

Um die Kamera zu positionieren, legen Sie die Position auf einem Wrapper fest. Stellen Sie die Position nicht direkt auf dem Kamera-Grundelement ein, da die gesetzte Position durch Steuerelemente schnell überschrieben wird:

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

Kamera online lesen: https://riptutorial.com/de/aframe/topic/10181/kamera

Kapitel 8: Komponenten

Einführung

Im Entity-Component-System-Muster ist eine Komponente ein wiederverwendbarer und modularer Datenblock, den wir in eine Entity einfügen, um Aussehen, Verhalten und / oder Funktionalität hinzuzufügen.

In A-Frame ändern Komponenten Objekte, die 3D-Objekte in der Szene sind. Wir mischen und komponieren Komponenten, um komplexe Objekte zu erstellen. Damit lassen wir three.js und JavaScript-Code in Module einkapseln, die wir deklarativ aus HTML verwenden können. Komponenten sind in etwa analog zu CSS.

Bemerkungen

Methoden zum Definieren von Lebenszyklus-Handlern

Da das Schema die Anatomie ist, sind die Lebenszyklusmethoden die Physiologie. Das Schema definiert die Form der Daten, *verwenden* die Lifecycle - Handler Methoden , um die Daten , die die Einheit zu modifizieren. Die Handler interagieren normalerweise mit der **Entity-API** .

Methodenübersicht

Methode	Beschreibung
drin	Wird einmal aufgerufen, wenn die Komponente initialisiert wird. Dient zum Einrichten des Anfangsstatus und zum Instanziieren von Variablen.
aktualisieren	Wird sowohl bei der Initialisierung der Komponente als auch beim Aktualisieren einer der Eigenschaften der Komponente aufgerufen (z. B. über *setAttribute*). Wird verwendet, um die Entität zu ändern.
Löschen	*Wird* aufgerufen, wenn die Komponente aus der Entität entfernt wird (z. B. über *removeAttribute*) oder wenn die Entität von der Szene getrennt wird. Wird verwendet, um alle vorherigen Änderungen an der Entität rückgängig zu machen.
Tick	Wird für jede Render-Schleife oder jeden Tick der Szene aufgerufen. Wird für ständige Änderungen oder Prüfungen verwendet.
abspielen	Wird aufgerufen, wenn die Szene oder das Objekt abgespielt wird, um Hintergrund oder dynamisches Verhalten hinzuzufügen. Wird auch einmal aufgerufen, wenn die Komponente initialisiert wird. Wird verwendet, um das Verhalten zu starten oder wieder aufzunehmen.
Pause	Wird aufgerufen, wenn die Szene oder das Objekt pausiert, um Hintergrund

Methode	Beschreibung
	oder dynamisches Verhalten zu entfernen. Wird auch aufgerufen, wenn die Komponente aus der Entität entfernt wird oder wenn die Entität von der Szene getrennt wird. Wird verwendet, um das Verhalten anzuhalten.
updateSchema	Wird aufgerufen, wenn eine der Eigenschaften der Komponente aktualisiert wird. Kann verwendet werden, um das Schema dynamisch zu ändern.

Eigenschaften des Komponentenprototyps

Innerhalb der Methoden haben wir Zugriff auf die Komponente Prototyp über `this` :

Eigentum	Beschreibung
diese Daten	Analysierte Komponenteneigenschaften, die aus den Schema-Standardwerten, Mixins und Attributen der Entität berechnet wurden.
this.el	Referenz auf die Entität [Entität] als HTML-Element.
this.el.sceneEl	Referenz auf die [Szene] [Szene] als HTML-Element.
this.id	Wenn die Komponente [mehrere Instanzen] [mehrere] haben kann, die ID der einzelnen Instanz der Komponente (z. B. `foo` from `sound__foo`).

Methoden

.drin ()

`.init ()` wird einmal zu Beginn des Lebenszyklus der Komponente aufgerufen. Eine Entität kann den `init` Handler der Komponente aufrufen:

- Wenn die Komponente für die Entität in der HTML-Datei statisch festgelegt ist und die Seite geladen wird.
- Wenn die Komponente über `setAttribute` auf eine verbundene Entität `setAttribute` .
- Wenn die Komponente auf ein nicht verbundenes `appendChild` ist und das `appendChild` dann über `appendChild` an die Szene `appendChild` .

Der `init` Handler wird häufig verwendet, um:

- Richten Sie den Anfangsstatus und die Variablen ein
- Bindungsmethoden
- Event-Listener anhängen

Beispielsweise würde das `init` einer Cursor-Komponente Zustandsvariablen setzen, Methoden binden und Ereignis-Listener hinzufügen:

```
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 (alteDaten)

`.update (oldData)` wird aufgerufen, wenn sich die Eigenschaften der Komponente ändern, auch zu Beginn des Lebenszyklus der Komponente. Eine Entität kann den `update` Handler einer Komponente aufrufen:

- Nach dem Aufruf von `init ()` zu Beginn des Komponentenlebenszyklus.
- Wenn die Eigenschaften der Komponente mit `.setAttribute` aktualisiert `.setAttribute` .

Der `update` Handler wird häufig verwendet, um:

- `this.data` Sie die meiste Arbeit aus, indem `this.data` mit `this.data` Änderungen an der Entität `this.data` .
- Ändern Sie die Entität, wenn sich eine oder mehrere Komponenteneigenschaften ändern.

Granulare Änderungen an der Entität können durch [diffing] [diff] des aktuellen Datensatzes (`this.data`) mit dem vorherigen Datensatz vor dem Update (`oldData`) durchgeführt werden.

A-Frame ruft `.update()` sowohl zu Beginn des Lebenszyklus einer Komponente als auch bei jeder Änderung der Daten einer Komponente auf (z. B. infolge von `setAttribute`). Der Update-Handler verwendet häufig diese `this.data` , um die Entität zu ändern. Der Update-Handler hat über sein erstes Argument Zugriff auf den vorherigen Status der Komponentendaten. Anhand der vorherigen Daten einer Komponente können wir genau feststellen, welche Eigenschaften geändert wurden, um granulare Aktualisierungen durchzuführen.

Die `update` der **sichtbaren** Komponente legt beispielsweise die Sichtbarkeit der Entität fest.

```
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;
  }
  // ...
});
```

.Löschen ()

`.remove ()` wird aufgerufen, wenn die Komponente von der Entität getrennt wird. Eine Entität kann den `remove` Handler einer Komponente aufrufen:

- Wenn die Komponente über `removeAttribute` aus der Entität `removeAttribute` .
- Wenn das `removeChild` von der Szene getrennt wird (z. B. `removeChild`).

Der `remove` Handler wird häufig verwendet, um:

- Entfernen, rückgängig machen oder bereinigen Sie alle Änderungen der Komponente an der Entität.
- Event-Hörer trennen.

Wenn beispielsweise [Lichtkomponente] [Licht] entfernt wird, entfernt die Lichtkomponente das Lichtobjekt, das zuvor für das Objekt festgelegt wurde, und entfernt es somit aus der Szene.

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

.tick (Zeit, ZeitDelta)

`.tick ()` wird bei jedem Tick oder Frame der Render-Schleife der Szene aufgerufen. Die Szene ruft den `tick` Handler einer Komponente auf:

- In jedem Frame der Render-Schleife.
- In der Größenordnung von 60 bis 120 Mal pro Sekunde.
- Wenn das Objekt oder die Szene nicht angehalten ist (z. B. ist der Inspector geöffnet).
- Wenn das Objekt noch an die Szene angehängt ist.

Der `tick` Handler wird häufig verwendet, um:

- Ändern Sie die Entität kontinuierlich in jedem Frame oder in einem Intervall.
- Umfrage nach Bedingungen.

Dem `tick` Handler werden die globale Betriebszeit der Szene in Millisekunden (`time`) und der Zeitunterschied in Millisekunden seit dem letzten Frame (`timeDelta`) `timeDelta` . Diese können zur Interpolation oder zum Ausführen von Teilen des `tick` Handlers in einem festgelegten Intervall verwendet werden.

Die **nachverfolgte Steuerungskomponente führt** beispielsweise die Animationen des Controllers fort, aktualisiert die Position und Drehung des Controllers und prüft, ob Tasten gedrückt werden.

```
AFRAME.registerComponent('tracked-controls', {
```

```
  // ...
  tick: function (time, timeDelta) {
    this.updateMeshAnimation();
    this.updatePose();
    this.updateButtons();
  }
  // ...
});
```

.Pause ()

`.pause ()` wird aufgerufen, wenn das `.pause ()` oder die Szene pausiert. Die Entität kann den `pause` einer Komponente aufrufen:

- Bevor die Komponente entfernt wird, bevor der `remove` Handler aufgerufen wird.
- Wenn die Entität mit `Entity.pause ()` angehalten wird.
- Wenn die Szene mit `Scene.pause ()` angehalten wird (z. B. der Inspector wird geöffnet).

Der `pause` Handler wird häufig verwendet, um:

- Event-Listener entfernen
- Entfernen Sie alle Chancen für dynamisches Verhalten.

Zum Beispiel wird die **Klangkomponente** den Ton anhalten und einen Ereignis - Listener entfernen , die einen Ton auf einem Event gespielt hätten:

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

.abspielen ()

`.play ()` wird aufgerufen, wenn das `.play ()` oder die Szene `.play ()` wird. Die Entität kann den `play` Handler einer Komponente aufrufen:

- Wenn die Komponente zum ersten Mal angeschlossen wird, nachdem der `update` Handler aufgerufen wurde.
- Wenn die Entität angehalten wurde, dann aber mit `Entity.play ()` .
- Wenn die Szene angehalten wurde, dann aber mit `Scene.play ()` .

Der `play` Handler wird häufig verwendet, um:

- Fügen Sie Ereignis-Listener hinzu.

Zum Beispiel gibt die **Soundkomponente** den Sound wieder und aktualisiert den Event-Listener, der bei einem Event einen Sound abspielen würde:

```
AFRAME.registerComponent('sound', {
  // ...
  play: function () {
    if (this.data.autoplay) { this.playSound(); }
    this.updateEventListener();
  }
  // ...
});
```

.updateSchema (Daten)

`.updateSchema ()` , falls definiert, bei jeder Aktualisierung aufgerufen, um zu prüfen, ob das Schema dynamisch geändert werden muss.

Der `updateSchema` Handler wird häufig verwendet, um:

- Das Schema dynamisch aktualisieren oder erweitern, normalerweise abhängig vom Wert einer Eigenschaft.

Die **Geometriekomponente** prüft beispielsweise, ob sich die `primitive` geändert hat, um festzustellen, ob das Schema für einen anderen Geometrietyp aktualisiert werden soll:

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

KOMPONENTEN-PROTOTYP-VERFAHREN

.flushToDOM ()

Um CPU-Zeit bei der Stringifizierung zu sparen, aktualisiert A-Frame nur im Debug-Modus die serialisierte Darstellung der Komponente im tatsächlichen DOM. Durch Aufrufen von flushToDOM () werden die Daten der Komponente manuell serialisiert und das DOM aktualisiert:

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

Examples

Registrieren Sie eine benutzerdefinierte A-Frame-Komponente

AFRAME.registerComponent (Name, Definition)

Registrieren Sie eine A-Frame-Komponente. Wir müssen Komponenten registrieren, bevor wir sie irgendwo in verwenden . Das heißt, aus einer HTML-Datei sollten Komponenten vorher in Ordnung kommen .

- **{string} name** - Komponentenname. Die öffentliche API der Komponente, dargestellt durch einen HTML-Attributnamen.
- **{Objekt} Definition** - Komponentendefinition. Enthält Schema- und Lebenszyklus-Handler-Methoden.

Komponente in foo in Ihrer js-Datei registrieren, z. B. foo-component.js

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

Verwendung der FOO- Komponente in Ihrer Szene

```
<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>
```

Komponenten-HTML-Formular

Eine Komponente enthält einen Datenbereich in Form einer oder mehrerer Komponenteneigenschaften. Komponenten verwenden diese Daten, um Entitäten zu ändern. Betrachten Sie eine Motorkomponente, können wir Eigenschaften wie Leistung oder Zylinder definieren.

HTML-Attribute stehen für Komponentennamen und der Wert dieser Attribute für

Komponentendaten.

Einzelneigenschaftskomponente

Wenn eine Komponente eine Einzeleigenschaftskomponente ist, dh ihre Daten aus einem einzelnen Wert bestehen, sieht der Komponentenwert in HTML wie ein normales HTML-Attribut aus:

```
<!-- `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>
```

Multi-Property-Komponente

Wenn eine Komponente eine Komponente mit mehreren Eigenschaften ist, dh die Daten bestehen aus mehreren Eigenschaften und Werten, dann ähnelt der Komponentenwert in HTML den Inline-CSS-Stilen:

```
<!-- `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>
```

Definieren eines zugehörigen Schemaobjekts

Das Schema ist ein Objekt, das die Eigenschaft oder die Eigenschaften der Komponente definiert und beschreibt. Die Schlüssel des Schemas sind die Namen der Eigenschaft, und die Werte des Schemas definieren die Typen und Werte der Eigenschaft (im Fall einer Komponente mit mehreren Eigenschaften):

Definieren des Schemas in Ihrer Komponente

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

Überschreiben definierter Schema-Standardwerte

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

Single-Property-Schema

Eine Komponente kann entweder eine Komponente mit nur einer Eigenschaft (bestehend aus

einem anonymen Wert) oder eine Komponente mit mehreren Eigenschaften (bestehend aus mehreren benannten Werten) sein. A-Frame ermittelt anhand der Struktur des Schemas, ob es sich bei einer Komponente um eine Einzeleigenschaft gegenüber einer Mehrfacheigenschaft handelt.

Das Schema einer Komponente mit einer einzigen Eigenschaft enthält `type` und / oder `default` Die Werte des Schemas sind Werte und keine Objekte:

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

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

A-Frame-Eigenschaftstypen für das Komponentenschema

Eigenschaftstypen definieren hauptsächlich, wie das Schema eingehende Daten aus dem DOM für jede Eigenschaft analysiert. Die geparsten Daten stehen dann über die `data` Eigenschaft des Prototyps der Komponente zur Verfügung. Nachfolgend finden Sie die integrierten Eigenschaftstypen von A-Frame:

Art der Immobilie	Beschreibung	Standardwert
Array	Analysiert durch Kommas getrennte Werte in ein Array (dh `"1, 2, 3" to ['1', '2', '3']`) .	[]
Vermögenswert	Für URLs, die auf allgemeine Assets verweisen. Kann URL aus einer Zeichenfolge in Form von `url(<url>)` analysieren. Wenn der Wert ein Element-ID-Selektor ist (z. B. `#texture`), `#texture` dieser Eigenschaftstyp `getElementById` und `getAttribute('src')` auf, um eine URL zurückzugeben. Der `asset` Eigenschaftstyp kann möglicherweise nicht geändert werden, um XHRs zu verarbeiten oder MediaElements direkt zurückzugeben (z. B. `<img>` -Elemente).	"
Audio-	Gleiche Analyse wie der `asset` Property-Typ. Wird möglicherweise vom A-Frame Inspector zum Präsentieren von Audio-Assets verwendet.	"
boolean	Parses string zu Boolean (dh `"false"` zu false, alles andere wahr).	falsch
Farbe	Derzeit wird kein Parsing durchgeführt. Wird hauptsächlich vom A-Frame Inspector verwendet, um	#F F F

Art der Immobilie	Beschreibung	Standardwert
	einen Farbwähler darzustellen. Außerdem ist es erforderlich, einen Farbtyp zu verwenden, damit Farbanimationen funktionieren.	
int	`parseInt` (z. B. `"124.5"` bis `124`).	0
Karte	Gleiche Analyse wie der `asset` Property-Typ. Wird möglicherweise im A-Frame Inspector zum Präsentieren von Textur-Assets verwendet.	"
Modell-	Gleiche Analyse wie der `asset` Property-Typ. Wird möglicherweise vom A-Frame Inspector zur Präsentation von Modell-Assets verwendet.	"
Nummer	`parseFloat` (z. B. `'124.5'` bis `'124.5'`).	0
Wähler	`querySelector` (z. B. `"#box"` an `<a-entity id="box">`).	Null
SelectorAll	`querySelectorAll` und konvertiert `NodeList` in `Array` (z. B. `".boxes"` in [<a-entity class = "Boxen", ...]).	Null
Schnur	Parsing nicht	"
vec2	Parst zwei Zahlen in ein `{x, y}` -Objekt (z. B. `1 -2` bis `{x: 1, y: -2}` .	{x: 0, y: 0}
vec3	Analysiert drei Zahlen in ein `{x, y, z}` -Objekt (z. B. `1 -2 3` bis `{x: 1, y: -2, z: 3}` .	{x: 0, y: 0, z: 0}
vec4	Parst vier Zahlen in ein Objekt `{x, y, z, w}` (z. B. `1 -2 3 -4.5` bis `{x: 1, y: -2, z: 3, w: -4.5}` .	{x: 0, y: 0, z: 0, w: 0}

Eigenschaftstyp Inferenz

Das Schema versucht, auf einen Eigenschaftstyp nur bei einem Standardwert zu schließen:

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

Das Schema legt einen Standardwert fest, wenn es nicht angegeben wird, wenn der Eigenschaftstyp angegeben ist

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

Benutzerdefinierter Eigenschaftstyp

Wir können auch unseren eigenen Eigenschaftstyp oder Parser definieren, indem Sie anstelle eines `type` eine `parse` bereitstellen:

```
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('/');
    }
  }
}
```

Zugriff auf Mitglieder und Methoden einer Komponente

Auf die Member und Methoden einer Komponente kann über das Entity-Objekt über die Entität **zugegriffen** werden. Suchen Sie die Komponente über die Entitätskarte der Komponenten und wir haben Zugriff auf die Komponenten der Komponente. Betrachten Sie diese Beispielkomponente:

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

Lassen Sie uns auf die **Bar- Member-** und **Qux-** Methode **zugreifen** :

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

Komponenten online lesen: https://riptutorial.com/de/aframe/topic/10068/komponenten

Kapitel 9: Licht (Komponente)

Einführung

Die Lichtkomponente definiert die Entität als Lichtquelle. Licht wirkt sich auf alle Materialien aus, für die kein Flat-Shading-Modell mit Shader angegeben wurde: Flat. Beachten Sie, dass Lichter rechnerisch teuer sind. Wir sollten die Anzahl der Lichter in einer Szene begrenzen.

Syntax

- <a-entity light = "color: #AFA; Intensität: 1,5" position = "- 1 1 0"> </ a-entity>
- <a-light type = "Punkt" color = "blau" position = "0 5 0"> </ a-light>

Parameter

Parameter	Einzelheiten
Art	Eine von Umgebung, Richtung, Halbkugel, Punkt, Punkt.
Farbe	Helle Farbe.
Intensität	Lichtstärke

Examples

Umgebungs

Umgebungslichter wirken sich global auf alle Objekte in der Szene aus. Die Farb- und Intensitätseigenschaften definieren Umgebungslicht. Darüber hinaus haben Position, Drehung und Skalierung keinen Einfluss auf Umgebungslichter.

Es wird empfohlen, Umgebungslicht so zu verwenden, dass Schattenbereiche nicht vollständig schwarz sind und indirekte Beleuchtung imitiert wird.

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

Directional

Richtungslichter sind wie eine Lichtquelle, die unendlich weit entfernt ist, aber aus einer bestimmten Richtung wie die Sonne scheint. Die absolute Position hat also keinen Einfluss auf die Intensität des Lichts einer Entität. Wir können die Richtung über die Positionskomponente angeben.

Das folgende Beispiel erstellt eine Lichtquelle, die von links oben in einem Winkel von 45 Grad scheint. Beachten Sie, dass Position = "- 100 100 0" und Position = "- 1 1 0" gleich sind, da nur der Vektor von Bedeutung ist.

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

Wir können die Richtung des gerichteten Lichts mit seiner Ausrichtung angeben, indem Sie eine untergeordnete Entität erstellen, auf die es gerichtet ist. Zum Beispiel, wenn Sie auf die -Z-Achse zeigen:

```
<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>
```

Hemisphäre

Hemisphere-Lichter sind wie ein Umgebungslicht, haben aber zwei verschiedene Farben, eine von oben (Farbe) und eine von unten (Grundfarbe). Dies kann für Szenen mit zwei unterschiedlichen Lichtfarben nützlich sein (z. B. ein Grasfeld unter einem grauen Himmel).

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

Eigentum	Beschreibung	Standardwert
Grundfarbe	Lichtfarbe von unten.	#F f f

Punkt

Punktlichter sind im Gegensatz zu direktionalen Lichtern omnidirektional und wirken sich abhängig von ihrer Position und Entfernung auf Materialien aus. Point Likes sind wie eine Glühbirne. Je näher die Glühlampe einem Objekt kommt, desto stärker leuchtet das Objekt.

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

Eigentum	Beschreibung	Standardwert
zerfallen	Menge, die das Licht entlang der Entfernung des Lichts schwimmt.	1,0
Entfernung	Abstand, bei dem Intensität zu 0 wird. Wenn Abstand 0 ist, fällt das Punktlicht nicht mit der Entfernung ab.	0,0

Stelle

Punktlichter sind wie Punktlichter in dem Sinne, dass sie Materialien abhängig von ihrer Position

und Entfernung beeinflussen, aber Punktlichter sind nicht omnidirektional. Sie werfen hauptsächlich Licht in eine Richtung, wie das Bat-Signal.

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

Eigentum	Beschreibung	Standardwert
Winkel	Maximale Ausdehnung des Punktlichts aus seiner Richtung (in Grad).	60
zerfallen	Menge, die das Licht entlang der Entfernung des Lichts schwimmt.	1,0
Entfernung	Abstand, bei dem Intensität zu 0 wird. Wenn Abstand 0 ist, fällt das Punktlicht nicht mit der Entfernung ab.	0,0
Halbschatten	Prozentsatz des Scheinwerferkegels, der durch Halbschatten gedämpft wird.	0,0
Ziel	Element, auf das der Punkt zeigen soll. auf null setzen, um Spotlight nach Orientierung zu transformieren und auf seine -Z-Achse zu zeigen.	Null

Standardbeleuchtung

A-Frame-Szenen erzeugen standardmäßig eine Standardbeleuchtung, ein Umgebungslicht und ein gerichtetes Licht. Diese Standardleuchten sind im DOM mit dem Attribut data-aframe-default-light sichtbar. Wann immer wir Lichter hinzufügen, werden die Standardlichter von A-Frame aus der Szene entfernt.

```
<!-- 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>
```

Licht (Komponente) online lesen: https://riptutorial.com/de/aframe/topic/10078/licht--komponente-

Kapitel 10: Mischmodell (Komponente)

Einführung

blend-model-Komponente Lädt ein JSON-Modell im Three.js-Format, das die Skelettanimation mit **THREE.BlendCharacter enthält** . Dies wird hauptsächlich zur Darstellung der Hand- und Vive-Controller verwendet.

Syntax

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

Bemerkungen

WERTE

Art	Beschreibung
Wähler	Selektor für eine `<a-asset-item>`
Schnur	`url()` geschlossener Pfad zu einer JSON-Datei

VERANSTALTUNGEN

Veranstaltungsname	Beschreibung
Modell geladen	Das JSON-Modell wurde in die Szene geladen.

Examples

Beispiel für die Verwendung von 'blend-model'

Wir können das Modell laden, indem wir mit der ID auf eine src auf eine Datei zeigen:

```
<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>
```

Mischmodell (Komponente) online lesen:
https://riptutorial.com/de/aframe/topic/10073/mischmodell--komponente-

Kapitel 11: Mixins

Einführung

Mixins bieten eine Möglichkeit, häufig verwendete Sätze von Komponenteneigenschaften zusammenzustellen und wiederzuverwenden. Sie werden mit dem `<a-mixin>` -Element definiert und in `<a-assets>` . Mixins sollten mit einer ID festgelegt werden. Wenn eine Entität diese ID als Mixin-Attribut festlegt, übernimmt die Entität alle Attribute des Mixins.

Examples

Beispiel für die Verwendung von 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>
```

Die Entität mit dem roten Würfel nimmt die Eigenschaften aus der roten Mischung und der Würfelmischung in dieser Reihenfolge auf. Ebenso mit dem blauen Würfel. Konzeptionell erweitern sich die obigen Elemente um:

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

Komponenteneigenschaften zusammenführen

Eigenschaften einer Komponente mit mehreren Eigenschaften werden zusammengeführt, wenn sie von mehreren Mixins und / oder der Entität definiert werden. Zum Beispiel:

```
<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>
```

Alle Eigenschaften der Geometriekomponenten werden zusammengeführt, da sie als Mixins enthalten und in der Entität definiert sind. Die Entität wäre dann äquivalent zu:

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

Ordnung und Vorrang

Wenn eine Entität mehrere Mixins enthält, die dieselben Komponenteneigenschaften definieren, hat das Mixin ganz rechts Vorrang. Im folgenden Beispiel enthält die Entität sowohl `red` als auch `blue` Mixins. Da das `blue` Mixin zuletzt enthalten ist, ist die endgültige Farbe des Würfels blau.

```
<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>
```

Wenn eine Entität selbst eine Eigenschaft definiert, die bereits durch ein Mixin definiert ist, hat die Definition der Entität Vorrang. Im folgenden Beispiel umfasst die Entität sowohl `red` als auch `blue` Mixins und definiert auch eine grüne Farbe. Da das Objekt direkt seine eigene Farbe definiert, ist die endgültige Farbe des Würfels grün.

```
<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>
```

Mixins online lesen: https://riptutorial.com/de/aframe/topic/10072/mixins--a-mixin-

Kapitel 12: Primitive

Einführung

Primitive sind nur `<a-entity>` unter der Haube. Dies bedeutet, dass Grundelemente dieselbe API wie Elemente zum Positionieren, Drehen, Skalieren und Anfügen von Komponenten haben. A-Frame bietet eine Handvoll Elemente wie `<a-box>` oder `<a-sky>` als Grundelemente bezeichnet werden, die das Entitätskomponentenmuster umschließen, um es für Anfänger attraktiv zu machen. . Entwickler können auch ihre eigenen Grundelemente erstellen.

Bemerkungen

Unter der Haube

Primitive fungieren als Convenience-Schicht (dh syntaktischer Zucker) in erster Linie für Neuankömmlinge. Denken Sie im Moment daran, dass Primitive `<a-entity>` s unter der Haube sind:

- Einen semantischen Namen haben (zB `<a-box>`)
- Sie haben ein voreingestelltes Bündel von Komponenten mit Standardwerten
- Zuordnung oder Proxy-HTML-Attribute zu [Komponente] [Komponente] -Daten

Primitive ähneln Prefabs in Unity . Einige Literaturangaben zum Entity-Component-System-Muster beziehen sich auf Assemblagen . Sie abstrahieren die Kern-Entitätskomponenten-API wie folgt:

- Stellen Sie nützliche Komponenten zusammen mit den vorgeschriebenen Standardwerten vor
- Als Abkürzung für komplexe, aber häufig vorkommende Entitäten dienen (z. B. `<a-sky>`)
- Bieten Sie eine vertraute Benutzeroberfläche für Anfänger, da A-Frame HTML in eine neue Richtung lenkt

Unter der Haube dieses `<a-box>` -Primitiv:

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

stellt dieses Entitätskomponentenformular dar:

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

`<a-box>` die Eigenschaft `geometry.primitive` standardmäßig auf `box` . Und die primitive bildet das HTML - `width` - Attribut auf die zugrunde liegende `geometry.width` Eigenschaft sowie das HTML - `color` auf die zugrunde liegende `material.color` Eigenschaft.

Examples

Registrieren eines Primitivs

Wir können unsere eigenen Grundelemente (dh ein Element registrieren) mit `AFRAME.registerPrimitive(name, definition)` registrieren. `definition` ist ein JavaScript-Objekt, das diese Eigenschaften definiert:

Eigentum	Beschreibung
defaultComponents	Objekt, das Standardkomponenten des Grundelements angibt. Die Schlüssel sind die Namen der Komponenten und die Werte sind die Standarddaten der Komponenten.
Zuordnungen	Objekt, das die Zuordnung zwischen HTML-Attributnamen und Komponenteneigenschaftsnamen angibt. Immer wenn der HTML-Attributname aktualisiert wird, aktualisiert das Grundelement die entsprechende Komponenteneigenschaft. Die Komponenteneigenschaft wird mit der Punktsyntax `${componentName}.${propertyName}` .

Im Folgenden finden Sie die Registrierung von A-Frame für das `<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'
  }
}));
```

Welche wir dann benutzen können

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

stellt dieses Entitätskomponentenformular dar:

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

Primitive online lesen: https://riptutorial.com/de/aframe/topic/10074/primitive

Kapitel 13: Raycaster (Komponente)

Einführung

Die Raycaster-Komponente führt allgemeine Kreuzungsprüfungen mit einem Raycaster durch. Raycasting ist die Methode, eine Linie von einem Ursprung in eine Richtung auszudehnen und zu prüfen, ob sich diese Linie mit anderen Elementen schneidet. Die Raycaster-Komponente ist ein Wrapper auf dem Raycaster von three.js. Er prüft in einem bestimmten Intervall nach Schnittpunkten anhand einer Liste von Objekten und gibt Ereignisse in der Entität aus, wenn Schnittpunkte erkannt oder Schnittpunkte gelöscht werden (dh wenn der Raycaster nicht mehr vorhanden ist

Parameter

Parameter	Einzelheiten
weit	Maximale Entfernung, unter der die resultierenden Entitäten zurückgegeben werden. Kann nicht niedriger als in der Nähe sein.
Intervall	Anzahl der Millisekunden, die zwischen jedem Schnittpunkttest gewartet werden muss. Eine niedrigere Anzahl ist besser für schnellere Updates. Eine höhere Anzahl ist besser für die Leistung.
nahe	Mindestentfernung, über die erneute Entitäten zurückgegeben werden. Kann nicht niedriger als 0 sein.
Objekte	Abfrageselektor zum Auswählen der Objekte, die auf Schnittpunkt getestet werden sollen. Wenn nicht angegeben, werden alle Entitäten getestet.
rekursiv	Überprüft alle untergeordneten Objekte, wenn gesetzt. Sonst werden nur Schnittpunkte mit Stammobjekten geprüft.

Bemerkungen

Veranstaltungen

Name	Einzelheiten
`raycaster-intersected`	Wird an der durchschnittenen Entität gesendet. Entität kreuzt sich mit einem Raycaster. Ereignisdetails enthalten el, die Raycasting-Entität und Kreuzung, ein Objekt, das detaillierte Daten zur Kreuzung enthält.
`raycaster-intersected-`	Wird an der durchschnittenen Entität gesendet. Eine Entität schneidet

Name	Einzelheiten
`cleared`	nicht mehr mit einem Raycaster. Das Ereignisdetail enthält das Raycasting-Element el.
`raycaster-intersection`	Wird auf der Raycasting-Einheit gesendet. Raycaster kreuzt sich mit einer oder mehreren Entitäten. Ereignisdetails enthalten ein Array, ein Array mit den überkreuzten Entitäten und Schnittpunkte, ein Array von Objekten, die detaillierte Daten zu den Schnittpunkten enthalten.
`raycaster-intersection-cleared`	Wird auf der Raycasting-Einheit gesendet. Raycaster schneidet nicht mehr mit einer Entität. Das Ereignisdetail enthält el, die zuvor überschnittene Entität.

Mitglied

Mitglied	Beschreibung
`intersectedEls`	Einheiten, die den Raycaster aktuell schneiden.
`objects`	Three.js-Objekte, die auf Schnittpunkte getestet werden sollen. Wird scene.children, wenn nicht object property angegeben ist.
`raycaster`	drei .js Raycaster-Objekt.

Methode

Methode	Beschreibung
`refreshObjects`	Aktualisiert die Liste der Objekte, die auf der Objekteigenschaft basieren, um die Kreuzung zu testen.

Examples

Ursprung und Richtung des Raycasters einstellen

Der Raycaster hat einen Ursprung, von dem der Strahl ausgeht, und eine Richtung, in die der Strahl geht.

Der Ursprung des Raycaster liegt an der Position der Raycaster-Entität. Wir können den Ursprung des Raycaster ändern, indem Sie die Positionskomponente der Raycaster-Entität (oder der übergeordneten Entitäten der Raycaster-Entität) festlegen.

Die Richtung des Raycaster befindet sich in "vorderer" Richtung der Raycaster-Entität (dh 0 0 -1

auf der negativen Z-Achse). Wir können die Richtung des Raycaster ändern, indem Sie die Rotationskomponente der Raycaster-Entität (oder der übergeordneten Entitäten der Raycaster-Entität) festlegen.

Zum Beispiel wird hier ein Raycaster entlang der Länge eines gedrehten Geschosses angewendet:

```
<!-- 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>
```

Whitelisting Entities zum Testen auf Schnittmenge

Wir möchten normalerweise nicht alles in der Szene auf Schnittpunkte testen (z. B. für Kollisionen oder für Klicks). Selektive Kreuzungen sind gut für die Leistung, um die Anzahl der zu prüfenden Entitäten zu begrenzen, da der Schnittpunkttest eine Operation ist, die mehr als 60 Mal pro Sekunde ausgeführt wird.

Um die Objekte auszuwählen oder auszuwählen, die auf Schnittpunkte getestet werden sollen, können wir die Objekteigenschaft verwenden. Wenn diese Eigenschaft nicht definiert ist, testet der Raycaster jedes Objekt in der Szene auf Schnittmenge. Objekte nimmt einen Abfrageauswählwert an:

```
<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>
```

Raycaster (Komponente) online lesen: https://riptutorial.com/de/aframe/topic/10036/raycaster--komponente-

Kapitel 14: Steuerelemente (Komponente)

Einführung

Controller sind für das Eintauchen von Personen in eine VR-Anwendung unerlässlich. Das Potenzial von VR wird nicht ohne sie erfüllt, nämlich Controller mit sechs Freiheitsgraden (6DoF). Mit den Controllern können Personen die Szene erreichen und mit den Händen interagieren.

A-Frame bietet Komponenten für Controller im gesamten Spektrum, die von den jeweiligen WebVR-Browsern über die Gamepad-Web-API unterstützt werden. Es gibt Komponenten für Vive-, Oculus Touch-, Daydream- und GearVR-Controller.

Bemerkungen

Es ist möglich, dass Sie Gamepadextentions aktivieren müssen. Sie können dies mit diesen Schritten tun:

- **In Chrome:** Navigieren Sie zu `chrome://flags`
- **Unter Firefox:** Navigieren Sie zu " `about:config`
- **Unter IE:** Wechseln Sie auf Ihrem Desktop zum Gruppenrichtlinien-Editor
- **Auf Opera:** navigieren Sie zu `opera:config`
- **Am Rand:** navigieren Sie zu `about:flags`

Examples

Wasd steuert

Die WASD-Steuerelemente steuern eine Entität mit den Tasten `W` , `A` , `S` und `D` oder den Pfeiltasten. Die Komponente `wasd-controls` mit der Kamerakomponente an eine Entität angehängt.

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

Für Azerty-Tastaturen können Sie die Tasten `Z` , `Q` , `S` und `D verwenden`

Siehe Steuerelemente

Die Look-Control-Komponente:

- Dreht das Objekt, wenn wir ein VR-Display (HMD) drehen.
- Dreht das Objekt, wenn Sie die Maus mit gedrückter Maustaste ziehen.
- Dreht die Entität, wenn Sie den Touchscreen per Tastendruck ziehen.

Die Look-Control-Komponente wird normalerweise zusammen mit der Kamerakomponente verwendet.

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

Vorsichtsmaßnahmen

Wenn Sie eine eigene Komponente für Look-Steuerelemente erstellen möchten, müssen Sie die HMD-Tracking-Bits kopieren und in Ihre Komponente einfügen. In der Zukunft haben wir möglicherweise ein System, mit dem die Benutzer ihre Steuerelemente leichter erstellen können.

Blick zum Cursor hinzufügen

Dazu müssen Sie Ihrer Kamera eine Cursor-Komponente hinzufügen

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

Weitere Informationen finden Sie unter Cursor (Komponente) .

Handsteuerungen

A-Frame 0.x 0.3

A-Frame bietet eine Implementierung für die Unterstützung mehrerer Arten von 6DoF-Controllern (Vive, Oculus Touch) über die Handsteuerungskomponente. Die Handsteuerungskomponente ist in erster Linie für 6DoF-Steuerungen gedacht, da sie auf Interaktionen im Raummaßstab wie z. B. das Ergreifen von Objekten ausgerichtet ist. Die Handsteuerungskomponente funktioniert auf den Controllern von Vive und Oculus Touch durch:

- Einstellen der Vive-Controls und der Oculus-Touch-Control-Komponente
- Überschreiben der Controller-Modelle mit einem einfachen Handmodell
- Zuordnen von vive-spezifischen und Oculus Touch-spezifischen Ereignissen zu Handereignissen und Gesten (z. B. Griffabzug und Triggerdown nach Daumen)

So fügen Sie die Handsteuerungskomponente hinzu:

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

Leider gibt es noch keine 3DoF-Controller-Komponente, die alle Arten von 3DoF-Controllern (z. B. Daydream, GearVR) gut abstrahiert. Wir könnten einen benutzerdefinierten Controller erstellen, der mit beiden Controllern funktioniert. Es wäre ziemlich einfach abzudecken, da 3DoF-Controller kein großes Interaktionspotenzial bieten (dh nur Rotationsverfolgung mit einem Touchpad).

Die Handsteuerung gibt nachverfolgte Hände (mit einem vorgeschriebenen Modell) mit animierten Gesten. Handsteuerungen umschließen die Komponenten für die vive-Steuerelemente und die Oculus-Touch-Steuerelemente, die wiederum die Komponente für die nachverfolgte Steuerelemente umschließen. Die Komponente bietet zusätzliche Ereignisse und behandelt Handanimationen und Posen.

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

Verfolgte Steuerelemente

A-Frame 0.x 0.3

Die Tracked-Control-Komponente ist die Basis-Controller-Komponente von A-Frame, die die Grundlage für alle Controller-Komponenten von A-Frame bildet. Die Tracked-Control-Komponente:

- Ruft ein Gamepad-Objekt aus der Gamepad-API mit einer ID oder einem Präfix ab.
- Wendet die Position (Position und Ausrichtung) der Gamepad-API an, um die Controller-Bewegung zu lesen.
- Sucht nach Änderungen in den Schaltflächenwerten des Gamepad-Objekts, um Ereignisse anzuzeigen, wenn Schaltflächen gedrückt oder berührt werden und Achsen und Touchpads geändert werden (`axischanged` . `buttonchanged` `buttondown` , `buttonup` , `touchstart` , `touchend` , `buttonup` , `touchstart` , `touchend`).

Alle Controller-Komponenten von A-Frame bauen auf der `tracked-controls` Komponente auf:

- Festlegen der Tracked-Control-Komponente für die Entität mit der entsprechenden Gamepad-ID (z. B. Oculus Touch (Right)). Zum Beispiel macht die vive-controls-Komponente dies

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

  `tracked-controls` dann eine Verbindung zum entsprechenden Gamepad-Objekt her, um Pose und Ereignisse für die Entität bereitzustellen.
- Zusammenfassung der Ereignisse, die von nachverfolgten Steuerelementen bereitgestellt werden. Tracked-Control-Ereignisse sind auf niedrigem Niveau. Es wäre schwierig für uns zu erkennen, welche Schaltflächen aufgrund dieser Ereignisse alleine gedrückt wurden, da wir die Tastenzuordnungen vorher kennen mussten. Controller-Komponenten können die Zuordnungen für ihre jeweiligen Controller im Voraus kennen und mehr semantische Ereignisse wie `triggerdown` oder `xbuttonup` .
- Modell zur Verfügung stellen. `tracked-controls` allein bieten kein Erscheinungsbild. Controller-Komponenten können ein Modell bereitstellen, das visuelles Feedback, Gesten und Animationen anzeigt, wenn Tasten gedrückt oder berührt werden. Die folgenden

Controller-Komponenten werden nur aktiviert, wenn sie feststellen, dass der Controller in der Gamepad-API gefunden wurde und als verbunden erscheint.

Die `tracked-controls` Komponentenschnittstellen mit nachgeführt Controllern. Tracked-Controls verwenden die Gamepad-API zur Verarbeitung von Tracked-Controllern und werden von der Handsteuerungskomponente sowie den Komponenten Vive-Controls und Oculus-Touch-Controls abstrahiert. Diese Komponente wählt den geeigneten Controller aus, wendet die Position an die Entität an, überwacht den Status der Schaltflächen und gibt entsprechende Ereignisse aus.

Beachten Sie, dass aufgrund aktueller browserspezifischer Änderungen Vive-Controller möglicherweise von der Gamepad-API mit den ID-Werten *"OpenVR Gamepad"* oder *"OpenVR Controller" zurückgegeben* werden. `idPrefix` Verwendung von `idPrefix` für Vive / OpenVR-Controller empfohlen.

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

3Dof- und 6Dof-Controller

Hinzufügen von 3DoF-Controllern

Controller mit 3 Freiheitsgraden (3DoF) sind auf die Rotationsverfolgung beschränkt. 3DoF-Controller haben keine Positionsverfolgung, was bedeutet, dass wir die Hand weder hin- und herbewegen noch auf und ab bewegen können. Einen Controller mit nur 3DoF zu haben, ist wie eine Hand und ein Handgelenk ohne Arm. Lesen Sie mehr über die Freiheitsgrade für VR.

Die 3DoF-Controllerkomponenten bieten eine Rotationsverfolgung, ein Standardmodell, das der realen Hardware entspricht, und Ereignisse, um die Tastenzuordnungen zu abstrahieren. Die Controller für Google Daydream und Samsung GearVR verfügen über 3DoF und beide unterstützen nur einen Controller für eine Hand.

A-Frame 0.x 0.6

Tagtraum-Controller

Die Komponente daydream-Steuerelemente ist mit den Google Daydream-Controllern verbunden. Die Komponente mit den verfolgten Steuerelementen wird eingebettet, während Tastenzuordnungen, Ereignisse und ein Daydream-Controllermodell hinzugefügt werden, das die berührten und / oder gedrückten Tasten (Trackpad) hervorhebt.

Passen Sie den Daydream-Controller an, falls vorhanden, unabhängig von der Hand.

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

Passen Sie den Daydream-Controller an, falls vorhanden und für die angegebene Hand.

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

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

GearVR-Controller

Die Komponente gearvr-controls ist mit den VR-Controllern von Samsung / Oculus Gear verbunden. Es umschließt die Tracked-Control-Komponente und fügt Tastenzuordnungen, Ereignisse und ein Gear VR-Controller-Modell hinzu, das die berührten und / oder gedrückten Tasten (Trackpad, Trigger) hervorhebt.

```
<!-- 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>
```

Hinzufügen von 6DoF-Controllern

Controller mit 6 Freiheitsgraden (6DoF) verfügen sowohl über Rotations- als auch Positionsverfolgung. Im Gegensatz zu Controllern mit 3DoF, die auf die Orientierung beschränkt sind, können Controller mit 6DoF sich frei im 3D-Raum bewegen. 6DoF ermöglicht es uns, nach vorne, hinter unseren Rücken zu greifen, unsere Hände über unseren Körper oder nahe an unser Gesicht zu bewegen. 6DoF zu haben ist wie die Realität, wo wir sowohl Hände als auch Arme haben. 6DoF gilt auch für das Headset und zusätzliche Tracker (z. B. Füße, Requisiten). 6DoF ist ein Minimum für ein wirklich intensives VR-Erlebnis.

Die 6DoF-Controller-Komponenten bieten eine vollständige Nachverfolgung, ein Standardmodell, das der realen Hardware entspricht, und Ereignisse, um die Tastenzuordnungen zu abstrahieren. HTC Vive und Oculus Rift with Touch bieten 6DoF und Controller für beide Hände. HTC Vive bietet auch Tracker für die Verfolgung zusätzlicher Objekte in der realen Welt in der VR.

A-Frame 0.x 0.3

Vive Controller

Die Vive-Control-Komponente ist mit den HTC Vive-Controllern / -Leitern verbunden. Es umschließt die Tracked-Control-Komponente und fügt Tastenzuordnungen, Ereignisse und ein Vive-Controller-Modell hinzu, das die gedrückten Tasten (Trigger, Griff, Menü, System) und Trackpad hervorhebt.

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

A-Frame 0.x 0.5

Oculus Touch-Controller

Die Oculus-Touch-Control-Komponente ist mit den Oculus Touch-Controllern verbunden. Die Komponente mit den verfolgten Steuerelementen wird umrahmt, während Tastenzuordnungen, Ereignisse und ein Touch-Controller-Modell hinzugefügt werden.

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

Maussteuerung

Maussteuerungen werden nur außerhalb des VR-Modus unterstützt und können für Spiele ohne HMD verwendet werden. Weitere Informationen zu Maussteuerelementen finden Sie im Beispiel des Mauszeigers .

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

Steuerelemente (Komponente) online lesen:
https://riptutorial.com/de/aframe/topic/10112/steuerelemente--komponente-

Kapitel 15: System

Einführung

Ein System des Entity-Component-System-Patterns bietet globalen Klassen, Services und Management für Klassen von Komponenten. Es bietet öffentliche APIs (Methoden und Eigenschaften) für Komponentenklassen. Auf ein System kann über das Szenenelement zugegriffen werden und Komponenten können mit der globalen Szene verbunden werden.

Das Kamerasystem verwaltet beispielsweise alle Objekte mit der Kamerakomponente und steuert, welche Kamera die aktive Kamera ist.

Parameter

Parameter	Einzelheiten
Daten	Durch das Schema bereitgestellte Daten für Handler und Methoden
el	Verweis auf `<a-scene>`
Schema	Verhält sich wie Komponentenschemas. Parses zu Daten.

Bemerkungen

Methoden

Ein System definiert wie eine Komponente Lifecycle-Handler. Es kann auch Methoden definieren, die als öffentliche API gedacht sind.

Methode	Beschreibung
drin	Wird einmal aufgerufen, wenn das System initialisiert wird. Wird zum Initialisieren verwendet.
Pause	Wird aufgerufen, wenn die Szene angehalten wird. Wird verwendet, um dynamisches Verhalten zu stoppen.
abspielen	Wird aufgerufen, wenn die Szene startet oder fortgesetzt wird. Wird verwendet, um dynamisches Verhalten zu starten.
Tick	Wenn definiert, wird bei jedem Tick der Render-Schleife der Szene aufgerufen.

Examples

Ein System registrieren

Ein System wird ähnlich wie eine A-Frame-Komponente registriert.

Wenn der Systemname mit einem Komponentennamen übereinstimmt, hat die Komponente als this.system eine Referenz auf das System

```
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);
  }
});
```

Zugriff auf ein System

Ein instanziiertes System kann über die Szene aufgerufen werden:

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

Auf registrierte Systemprototypen kann über `AFRAME.systems` zugegriffen werden.

Trennung von Logik und Daten

Systeme können helfen, Logik und Verhalten von Daten zu trennen, wenn dies gewünscht wird. Wir lassen Systeme das schwere Heben übernehmen, und die Komponenten kümmern sich nur um die Verwaltung ihrer Daten über ihre Lebenszyklusmethoden:

```
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);
  }
```

```
});
```

Alle Komponenten eines Systems zusammenstellen

Es gibt keine strikte API, um zu definieren, wie Systeme Komponenten verwalten. Ein übliches Muster ist, dass Komponenten sich selbst beim System anmelden. Das System hat dann Verweise auf alle seine Komponenten:

```
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);
  }
});
```

System online lesen: https://riptutorial.com/de/aframe/topic/10067/system

Kapitel 16: Szene

Einführung

Eine Szene wird durch das Element `<a-scene>` . Die Szene ist das globale Stammobjekt, und alle Entitäten befinden sich in der Szene.

Die Szene erbt von der Entity-Klasse, sodass sie alle ihre Eigenschaften, ihre Methoden, die Möglichkeit zum Anfügen von Komponenten und das Verhalten zum Warten auf alle untergeordneten Knoten (z. B. `<a-assets>` und `<a-entity>`) `<a-entity>` vor dem Start der Render-Schleife laden.

Parameter

Parameter	Einzelheiten
Verhalten	Array von Komponenten mit Tick-Methoden, die auf jedem Frame ausgeführt werden.
Kamera	Aktive Drei-Kamera.
Segeltuch	Verweis auf das Canvas-Element.
isMobile	Ob die Umgebung als mobil erkannt wird oder nicht.
object3D	THREE.Scene-Objekt.
Renderer	Active THREE.WebGLRenderer.
RenderStarted	Ob Szene gerendert wird.
bewirken	Renderer für VR, erstellt durch Übergeben des aktiven Renderers an THREE.VREffect.
Systeme	Instantiierte Systeme.
Zeit	Globale Betriebszeit der Szene in Sekunden.

Bemerkungen

Methoden

Name	Beschreibung
enterVR	Wechseln Sie zu Stereo-Rendering und verschieben Sie den Inhalt auf das Headset. Muss in einem vom Benutzer generierten Event-Handler wie `click` aufgerufen werden. Beim ersten Aufrufen einer Seite in VR.
exitVR	Wechseln Sie zum Mono-Renderer, und der Inhalt wird nicht mehr auf dem Headset angezeigt.
neu laden	Bringen Sie die Szene in ihren ursprünglichen Zustand zurück.

VERANSTALTUNGEN

Name	Beschreibung
enter-vr	Der Benutzer hat VR eingegeben und das Headset beginnt mit der Präsentation von Inhalten.
exit-vr	Der Benutzer hat VR verlassen und das Headset wurde nicht mehr angezeigt.
geladen	Alle Knoten wurden geladen.
Renderstart	Die Render-Schleife hat begonnen.

Examples

Szenenkomponenten anbringen

Komponenten können sowohl an die Szene als auch an Objekte angehängt werden. A-Frame wird mit wenigen Komponenten zur Konfiguration der Szene geliefert:

Komponente	Einzelheiten
eingebettet	Entfernen Sie Vollbildstile von der Leinwand.
Nebel	Fügen Sie Nebel hinzu.
Tastatürkürzel	Tastenkombinationen umschalten
Inspektor	Injizieren Sie den A-Frame Inspector.
Statistiken	Leistungsstatistiken umschalten
vr-mode-ui	Umschalten der Benutzeroberfläche zum Ein- und Ausschalten von VR.
debuggen	Aktiviert die Serialisierung von Komponenten zu DOM.

Eingebettete Szenen verwenden

Wenn Sie WebVR-Inhalte, die mit HTML-Inhalt gemischt sind, verwenden möchten, beispielsweise wenn Sie einen erweiterten Inhalt eines Showcase-Schlüssels erstellen, können Sie das `embedded` Tag verwenden. Wenn Sie dies verwenden, können Sie sich mit dem Gyroskop Ihres Smartphones in 360 ° -Inhalten umsehen oder auf den Computer klicken und ziehen.

```
<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>
```

Debuggen

Die Debug-Komponente ermöglicht die Serialisierung von Komponente zu DOM.

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

Komponenten-zu-DOM-Serialisierung

Aus Performancegründen aktualisiert A-Frame das DOM standardmäßig nicht mit Komponentendaten. Wenn wir den DOM-Inspector des Browsers öffnen, werden nur die Komponentennamen (und nicht die Werte) angezeigt.

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

A-Frame speichert die Komponentendaten im Speicher. Das Aktualisieren des DOM benötigt CPU-Zeit für die Konvertierung interner Komponentendaten in Strings. Wenn wir das DOM-Update für Debugging-Zwecke anzeigen möchten, können wir die Debug-Komponente an die Szene anhängen. Komponenten prüfen, ob eine aktivierte Debug-Komponente vorhanden ist, bevor sie versuchen, die DOM zu serialisieren. Dann können wir Komponentendaten im DOM anzeigen:

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

Stellen Sie sicher, dass diese Komponente in der Produktion nicht aktiv ist.

Manuelles Serialisieren in DOM

Um die DOM manuell zu serialisieren, verwenden Sie Entity.flushToDOM oder 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.
```

Ausführen von Content-Skripts in der Szene

Die empfohlene Methode ist, eine Komponente zu schreiben und sie an das Szenenelement anzuhängen.
Die Szene und ihre Kinder werden vor dieser Komponente initialisiert.

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

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

Wenn Sie aus bestimmten Gründen keine dedizierte Komponente schreiben möchten, müssen Sie warten, bis die Szene das Initialisieren und Anfügen abgeschlossen hat:

```
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');
}
```

Szene online lesen: https://riptutorial.com/de/aframe/topic/10069/szene--a-scene-

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit aframe	Community, H. Pauwelyn, M.Kungla
2	Animation	M.Kungla
3	Asset Management System	M.Kungla
4	Cursor	H. Pauwelyn
5	Entitäten	M.Kungla
6	gltf-modell (komponente)	geekonaut
7	Kamera	H. Pauwelyn
8	Komponenten	M.Kungla
9	Licht (Komponente)	H. Pauwelyn
10	Mischmodell (Komponente)	M.Kungla
11	Mixins	M.Kungla
12	Primitive	M.Kungla
13	Raycaster (Komponente)	H. Pauwelyn, M.Kungla
14	Steuerelemente (Komponente)	H. Pauwelyn
15	System	M.Kungla
16	Szene	H. Pauwelyn, M.Kungla