Теорія по A-Frame

A-Frame — це веб-фреймворк, який дозволяє створювати різні програми, ігри, сцени у віртуальній реальності (ВР). Все вищеописане буде доступне прямо із браузера вашого шолома ВР. Цей інструмент буде корисний як тим, хто хоче займатися розробкою ВР ігор у браузері, так і, наприклад, може стати в нагоді як платформа для створення веб ВР додатків, сайтів, посадкових сторінок. Сфери використання веб-ВР обмежені лише вашою уявою. Навскидку можу навести пару сфер діяльності людини, де ВР може бути корисним: освіта, медицина, спорт, продаж, відпочинок.

Що там усередині?

A-Frame не написано з 0 на чистому WebGL, в його основі лежить бібліотека *Three.js* . Тому рекомендую для початку розібратися з базовими концепціями Three.js перш ніж починати працювати з A-Frame, хоча це й не обов'язково, тому що A-Frame влаштований таким чином, щоб ви найменше думали про рендеринг геометрії і більше концентрувалися на логіці вашої програми. На те він і фреймворк.

Для цього A-Frame постулює три основні положення, про які ми поговоримо далі.

A-Frame працює з HTML

Багато базових елементів A-Frame, таких як scene, camera, box, sphere та ін, додаються на сцену через однойменні теги з префіксом **a-** . Кожен подібний елемент зареєстрований як власний. На сьогоднішній день A-Frame (0.8.0) використовує специфікацію v0.

```
<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-scene>
```

Даний невеликий фрагмент коду відобразить WebGL сцену, до якої будуть додані два об'єкти: куб і сфера із заданими параметрами. Крім згаданих вище двох елементів, існує ще ряд інших примітивів, які можуть бути додані на сцену таким же чином: *<a-circle>, <a-cone>, <a-dodecahedron>, <a-icosahedron>, <a-octahedron>, <a-plane>, <a-ring> torus>, <a-triangle>* . Також в A-Frame існує низка інших елементів, які виконують певні функції:

- **<a-camera>** — створює камеру. На даний момент підтримується лише перспективна камера (PerspectiveCamera)
- **<a-obj-model>, <a-collada-model>, <a-gltf-model>** — всі вони вантажать та відображають моделі відповідного формату.
- **<a-cursor>** — елемент, який дозволяє виконувати різні дії: клік, наведення та ін. Курсор прив'язаний до центру камери, таким чином він завжди буде по центру того, що бачить користувач.
- **<a-image>** — відображення вибраного зображення на площині (<a-plane>).
- **<a-link>** - те ж саме тег, тільки для 3D сцени.
- **<a-sky>** — величезний циліндр навколо сцени, що дозволяє відображати 360 фотографій. Або його просто можна залити якимось кольором.
- **<a-sound>** - створює джерело звуку в заданій позиції.
- **<a-text>** - малює плоский текст.
- **<a-video>** - програє відео на площині.

Також хотілося б відзначити, що ми працюємо з елементами DOM, а тому можемо використовувати стандартне DOM API, включаючи querySelector, getElementById, appendChild і т.д/

A-Frame використовує ECS

ECS (Entity Component System) - патерн проектування програм та ігор. Широку поширеність набув якраз у другому сегменті. Як видно з назви, три основні поняття патерну це Entity (Сутність), Component (Компонент), System (Система). У класичному вигляді вони взаємопов'язані один з одним так: ми маємо певний об'єкт-контейнер (Сутність), до якого можна додавати компоненти. Зазвичай компонент відповідає за окрему частину логіки. Наприклад, у нас є об'єкт Player (Гравець), у нього є компонент Health (Здоров'я). Цей компонент міститиме всю логіку, пов'язану з поповненням або втратою здоров'я гравця (об'єкта). А системи, у свою чергу, потрібні для того, щоб керувати набором сутностей, об'єднаних деякими компонентами. Зазвичай якийсь компонент може зареєструвати сутність усередині однойменної системи.

В A-Frame цей патерн реалізований дуже просто та елегантно – за допомогою атрибутів. Як сутності використовуються будь-які елементи A-Frame - *<a-scene>, <a-box>, <a-sphere>,* та ін Але особняком звичайно ж стоїть елемент *<a-entity>* . Його ім'я говорить саме за себе. Всі інші елементи по суті є обгортками для компонентів і зроблені для зручності, тому що будь-який елемент можна створити і за допомогою *<a-entity>* . Наприклад *<a-box>* :

<a-entity geometry="primitive: box; width: 1; height: 1; depth: 1"></a-entity>

geometry — у разі є компонентом, який було додано до сутності *<a-entity>* . Сам по собі *<a-entity>* не має будь-якої логіки (у глобальному сенсі), а компонент *geometry* — по суті перетворює його на куб чи щось інше. Іншим не менш важливим ніж *geometry* компонентом є **матеріал** . Він додає до геометрії матеріалу. Матеріал відповідає за те, чи буде наш куб блищати як метал, чи матиме якісь текстури і т.д. У чистому *Three.js* нам довелося б створювати окремо геометрію, окремо матеріал, а потім це все потрібно було б скомбінувати в меші. Загалом такий підхід суттєво заощаджує час.

Будь-який компонент в A-Frame має бути зареєстрований глобально через спеціальну конструкцію:

```
AFRAME.registerComponent('hello-world', {
  init: function () {
    console.log('Hello, World!');
  }
});
```

Потім цей компонент можна буде додати на сцену або будь-який інший елемент.

<a-entity hello-world></a-entity>

Так як ми додали **init** коллбек для нашого компонента, як тільки елемент буде доданий в DOM, даний коллбек відпрацює і ми побачимо наше повідомлення в консолі. У компонентах A-Frame є інші коллбеки життєвого циклу. Давайте зупинимося на них докладніше:

- **update** — викликається як із ініціалізації як init, і при оновленні будь-якої властивості даного компонента.
- **remove** - викликається після видалення компонента або сутності, що його містить. Тобто, якщо ви видалите <a-entity> з DOM, всі його компоненти викликають remove коллбек.
- **tick** - викликається щоразу перед рендерингом сцени. Всередині цикл рендерингу використовує requestAnimaitonFrame
- **tock** - викликається щоразу після рендерингу сцени.
- **play** – викликається кожен при відновленні рендерингу сцени. Насправді після scene.play();
- **pause** - викликається щоразу при зупинці рендерингу сцени. Насправді після scene.pause();
- **updateSchema** - викликається щоразу після оновлення схеми.

Ще одним важливим концептом компонента A-Frame є схема. Схема визначає властивості компонента. Вона визначається так:

```
AFRAME.registerComponent('my-component', {
  schema: {
    arrayProperty: {type: 'array', default: []},
    integerProperty: {type: 'int', default: 5}
  }
}
```

В даному випадку наш компонент **my-component** міститиме дві властивості *arrayProperty* і *integerProperty* . Щоб передати їх у компонент, потрібно задати значення відповідного атрибута.

```
<a-entity my-component="arrayProperty: 1,2,3; integerProperty: 7"></a-entity>
```

Отримати ці властивості всередині компонента можна через властивість **data** .

```
AFRAME.registerComponent('my-component', {
  schema: {
    arrayProperty: {type: 'array', default: []},
    integerProperty: {type: 'int', default: 5}
  },
  init: function () {
    console.log(this.data);
  }
}
```

Щоб отримати властивості компонента з сутності до якої він доданий, можна скористатися трохи видозміненою функцією *getAttribute* . При зверненні до сутності A-Frame вона

поверне не просто рядкове значення атрибуту, а об'єкт *data* , згаданий вище.

```
 console.log(this.el.getAttribute('my-component'));
// {arrayProperty: [1,2,3], integerProperty: 7}
```

Приблизно так само можна змінити властивості компонента:

```
this.el.setAttribute('my-component',{arrayProperty: [], integerProperty: 5})
```

Тепер поговоримо про системи. Системи A-Frame реєструється схожим чином як і компоненти:

```
AFRAME.registerSystem('my-system', {
  schema: {},
  init: function () {
    console.log('Hello, System!');
  },
});
```

Як і компонент система має схему і коллбеки. Тільки у системи їх всього 5: **init, play, pause, tick, tock** . Систему не потрібно додавати як компонент сутності. Вона буде автоматично додана до сцени.

```
this.el.sceneEl.systems['my-system'];
```

Якщо компонент матиме таке саме ім'я як і система, то система буде доступна за посиланням **this.system** .

```
AFRAME.registerSystem('enemy', {
  schema: {},
  init: function () {},
});

AFRAME.registerComponent('enemy', {
  schema: {},
  init: function () {
    const enemySystem = this.system;
  },
});
```

Зазвичай системи потрібні для того, щоб збирати та керувати сутностями з відповідними компонентами. Тут, за ідеєю, має бути логіка, яка відноситься до колекції сутностей. Наприклад, управління появою ворогів у грі.

Комунікація в A-Frame відбувається за допомогою подій браузера

Навіщо винаходити колесо, якщо на даний момент існує чудова вбудована реалізація видавець-передплатник для елементів DOM. Події DOM дозволяють слухати як події браузера, такі як натискання на клавішу клавіатури, клік мишкою та інші, так і події користувача. A-Frame пропонує нам зручний і простий спосіб комунікації між сутностями, компонентами та системами, через події користувача. Для цього кожен елемент A-Frame пропатчений функцією **emit** , вона приймає три параметри: *перший* - назва події, *другий* - дані, яку потрібно передати, *третій* - чи повинна подія спливати.

```
this.el.emit('start-game', {level: 1}, false);
```

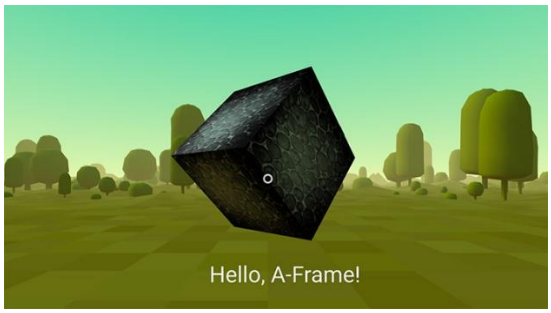
Підписатися на цю подію можна звичним усім нам способом, використовуючи **addEventListener:**

```
const someEntity = document.getElementById(‘someEntity’);
someEntity.addEventListener(‘start-game’, () => {...});
```

Спливання подій (bubbling) тут є дуже важливим моментом, адже іноді дві сутності, які повинні комунікувати з один одним, знаходяться на одному рівні, в такому випадку можна додати слухач події на елемент сцени. Він, як ви могли бачити раніше, доступний всередині кожного елемента через посилання **sceneEl** .

```
this.el.sceneEl.addEventListener(‘start-game’, () => {...});
```

Створюємо базову сцену в A-Frame



Від автора перекладу: у попередній статті ми обговорили базові концепції A-Frame. Щоб продовжити цикл, я хотів створити урок, який ілюстрував би основні можливості A-Frame, але зрозумів, що краще зробити переклад статті з офіційного сайту, яка, на мою думку написана чудово, і повторювати подібну статтю просто немає жодного сенсу.

Давайте зробимо базову сцену в A-Frame, щоб зрозуміти, як фреймворк працює. Нам знадобиться початкове розуміння HTML. Під час цього уроку ми вивчимо:

- як додавати 3D об'єкти за допомогою примітивів;
- як трансформувати об'єкти у 3-х мірному просторі за допомогою, переміщень, поворотів та масштабування;
- як додати оточення;
- як додати текстури;
- як додати базову інтерактивність за допомогою подій та анімації;
- Як додати текст

HTML:

```
<html>
 <head>
  <script src="https://aframe.io/releases/0.8.0/aframe.min.js"></script>
 </head>
 <body>
  <a-scene>
  </a-scene>
 </body>
</html>
```

Ми підключаємо A-Frame через тег <script>, доданий у розділ <head>. Код бібліотеки знаходиться на CDN. Підключення A-Frame *має бути додане до* **<a-scene>** так як A-Frame реєструє користувацькі HTML елементи, які повинні бути визначені до того, як <a-scene> буде додано до DOM, інакше сцена залишиться порожньою.

Далі ми додаємо тег <a-scene> до тіла документа <body>. <a-scene> це елемент, за допомогою якого відбувається керування сценою. Додаючи елементи до <a-scene> ми додаємо їх на фактичну 3D сцену. <a-scene> відповідає за все, що необхідно: встановлює WebGL, <canvas>, камеру, світло, renderer (відмальовувач), а так само і все, що пов'язано з підтримкою WebVR на таких платформах як HTC Vive, Oculus Rift, Samsung GearVR, Oculus Go та Google Cardboards.

Додаємо об'єкти

Використовуючи <a-scene>, ми додаємо 3D об'єкти за допомогою стандартних примітивів A-Frame, наприклад <a-box>. Ми можемо використовувати <a-box> просто як

звичайний HTML елемент, додаючи до нього атрибути кастомізації. Ось ще кілька стандартних примітивів доступних в A-Frame: <a-cylinder>, <a-plane>, <a-sphere>.

У наступному прикладі коду ми додамо кольоровий <a-box>. Більше атрибутів для <a-box> можна знайти тут .

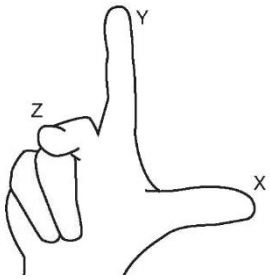
Необхідно зауважити, що примітиви в A-Frame є лише обгортками для деяких компонентів. Це зручно, але потрібно розуміти, що під <a-box> криється <a-entity> з geometry та material компонентами:

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

У будь-якому випадку, оскільки камера і куб, додані на сцену, знаходяться в одній точці *0,0,0* , ми не зможемо побачити куб до тих пір, поки не змінимо його позицію. Зробити це досить просто: потрібно лише поміняти атрибут position, щоб перемістити наш куб у просторі.

Трансформація об'єкта в 3D

Давайте спочатку поговоримо про 3D простір. A-Frame використовує праву (right-handed) систему координат. Напрямок камери за замовчуванням: позитивна X-вісь йде вправо, позитивна Y-вісь йде вгору і позитивна Z-вісь йде назад від екрана в напрямку нас:



A-Frame вимірює дистанцію в метрах (а не в пікселях як в React360) так як WebVR API повертає дані позиції та пози в . Коли ви проектуєте ВР сцену, дуже важливо розглядати саме реальні розміри об'єктів. Куб з висотою 10 метрів може виглядати нормально на екранах наших комп'ютерів, але він буде занадто масивним при зануренні у віртуальну реальність.

Одиниці обертання в A-Frame - це градуси (не радіани як Three.js), тому вони автоматично будуть перераховані в радіани при попаданні в Three.js. Для визначення позитивного напрямку обертання використовується правило правої руки. Направте палець правої руки вздовж позитивного напрямку будь-якої осі, тоді напрямок у якому наші пальці зігнуті та будуть позитивним напрямком обертання.

Щоб перемістити, обертати чи масштабувати наш куб, ми можемо змінити відповідно position , rotation і scale компоненти (представлені атрибутами тега <a-box>). Давайте спробуємо для початку застосувати обертання та масштабування:

```
<a-scene>
 <a-box color="red" rotation="0 45 45" scale="2 2 2"></a-box>
</a-scene>
```

Це має повернути наш куб на задані кути та розтягнути його по всіх осях у два рази.

Трансформації предків та нащадків

Граф сцени в A-Frame реалізований за допомогою HTML. Кожен елемент такої структури може мати кілька нащадків і лише предка. Будь-який нащадок у такій структурі успадковує властивості трансформації (position, rotation, scale) свого предка.

Наприклад, ми можемо додати сферу як нащадка куба:

```
<a-scene>
 <a-box position="0 2 0" rotation="0 45 45" scale="2 4 2">
  <a-sphere position="1 0 3"></a-sphere>
 </a-box>
</a-scene>
```

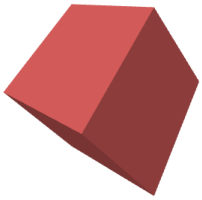
Позиція сфери на сцені буде *123* , а не *103* , оскільки за умовчанням сфера знаходиться в координатах свого предка, тобто куба - *020* . Ця точка у разі буде точкою відліку для сфери, що має координати *1 0 3* . Те саме стосується обертання і масштабу. Якщо будь-який із атрибутів куба буде змінено, ця зміна автоматично торкнеться всіх нащадків (у нашому випадку сферу).

Якби ми додали циліндр як нащадка сфери, трансформації циліндра було б пораховано виходячи з трансформацій його предків: куба і сфери.

Розміщуємо наш куб попереду від камери

Тепер зробимо наш куб видимим для нашої камери. Ми можемо пересунути наш куб на 5 метрів по осі Z у негативному напрямку (тобто від нас). Давайте також піднімемо його на два метри вгору по осі Y. Все це можна зробити, змінивши атрибут position:

```
<a-scene>
<a-box color="red" position="0 2 -5" rotation="0 45 45" scale="2 2 2"></a-box>
</a-scene>
```



Тепер ми його бачимо!

Управління

Для плоских дисплеїв (ноутбуки, комп'ютери) керування камерою за замовчуванням здійснюється через перетягування мишкою та WASD або стрілки на клавіатурі. У телефонах відповідає акселерометр. Незважаючи на те, що A-Frame є фреймворком для WebVR, він підтримує дані схеми управління для того, щоб сцену можна було переглянути і без шолома віртуальної реальності.

Якщо ви використовуєте шолом віртуальної реальності, управління буде здійснюватися за допомогою контролерів і шолома. Щоб потрапити у віртуальну реальність із браузера вашого шолома, вам потрібно натиснути на іконку із зображенням шолома віртуальної реальності, яка розташована у правому нижньому кутку. Якщо ви використовуєте шолом з 6-ма ступенями свободи, і у вас є реальний простір, ви можете фізично походити за створеною вами сценою.

Додаємо оточення

A-Frame дозволяє розробникам створювати та ділитися компонентами, які потім легко використовувати у своїх проектах. Компонент оточення створений Дієго Гоберна генерує різні оточення і все це досягається лише одним рядком коду! Компонент оточення

це чудовий та простий спосіб створити візуальну платформу для наших BP додатків. Він дозволяє створювати десятки оточень, що настроюються численною кількістю параметрів. Для початку потрібно підключити скрипт до розділу після A-Frame:

```
<head>
 <script src="https://aframe.io/releases/0.8.0/aframe.min.js"></script>
 <script src="https://unpkg.com/aframe-environment-component/dist/aframe-
 environment-component.min.js"></script>
 </head>
```

Після цього, вже в сцені, потрібно додати тег <a-entity> з атрибутом environment та заданими налаштуваннями, нехай це буде forest (ліс) з 200 деревами:

```
<a-scene>
 <a-box color="red" position="0 2 -5" rotation="0 45 45" scale="2 2 2"></a-box>
 <!-- Окружение, прямо из коробки! -->
 <a-entity environment="preset: forest; dressingAmount: 200"></a-entity>
 </a-scene>
```

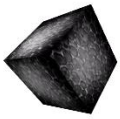
Позначка : майте на увазі, що при використанні слабкого "заліза" така сцена може і пригальмовувати. Наприклад, це може статися на мобільних пристроях Oculus Go. Так що якщо ви підтримуєте і мобільні пристрої, стежте за кількістю об'єктів, що генеруються.

Застосовуємо текстури

Ми можемо застосувати текстуру до нашого куба, використовуючи зображення, відео або canvas за допомогою атрибута src, також як ми робимо це зі звичайним тегом. Також нам необхідно видалити color=«red» оскільки колір все одно не відображатиметься під час використання текстури.

```
<a-scene>
<a-box src="https://i.imgur.com/mYmmbrp.jpg" position="0 2 -5" rotation="0 45 45"
  scale="2 2 2"></a-box>
<a-sky color="#222"></a-sky>
</a-scene>
```

Тепер ми маємо побачити наш куб уже з текстурою, підвантаженою он-лайн.



Використовуємо систему завантаження файлів

Щоб покращити продуктивність вашої програми на A-Frame, ми рекомендуємо вам використовувати систему керування завантаженням файлів . Ця система дозволяє браузеру кешувати різні файли (зображення, відео, 3d моделі). A-Frame відповідає за те, щоб усі ці файли були завантажені до рендерингу.

Якщо ми напишемо тег всередині <a-assets>, A-Frame не відображатиме його як у звичному HTML, він обробить джерело і відправить його до Three.js. Важливо відзначити, що завантаживши зображення один раз ми можемо використовувати його будь-де, наприклад як текстуру. Також A-Frame подбає про крос-доменності та інші можливі проблеми, пов'язані з передачею файлів по мережі.

Щоб використовувати систему керування завантаженням файлів для додавання текстури потрібно:

- Додати тег **<a-assets>** всередину тега <a-scene> (на перший рівень)

• Додати тег **<img>** , атрибут src повинен містити посилання на файл зображення (так само як і в звичному нам HTML)

• Потрібно встановити атрибут **id** для тега img. В ідеалі він повинен описувати саму текстуру (наприклад, id=”boxTexture”)

• Додати атрибут **src** для об'єкта, який повинен її містити. Наприклад, для <a-box src="#boxTexture" >

```
<a-scene>
 <a-assets>
   <img id="boxTexture" src="https://i.imgur.com/mYmmbrp.jpg">
 </a-assets>
 <a-box src="#boxTexture" position="0 2 -5" rotation="0 45 45" scale="2 2 2"></a-box>
 <a-sky color="#222"></a-sky>
</a-scene>
```

Створюємо довільне оточення

Ми вже говорили про компонент оточення вище. Воно генерується автоматично. А якщо нам потрібно зробити своє власне оточення: додати хмари, землю, інші об'єкти? Додаємо фонове зображення для сцени

Для додавання фонового зображення для сцени в A-Frame є спеціальний елемент <a-sky> . <a-sky> дозволяє додавати як чистий колір, так і 360 зображень або відео. Наприклад, щоб додати темно-сірий фон потрібно написати наступний код:

```
<a-scene>
 <a-box color="red" position="0 2 -5" rotation="0 45 45" scale="2 2 2"></a-box>
 <a-sky color="#222"></a-sky>
</a-scene>
```

Або ми можемо використовувати 360 фотографій:

```
<a-scene>
 <a-assets>
   <img id="boxTexture" src="https://i.imgur.com/mYmmbrp.jpg">
   <img id="skyTexture"
    src="https://cdn.aframe.io/360-image-gallery-boilerplate/img/sechelt.jpg">
 </a-assets>
 <a-box src="#boxTexture" position="0 2 -5" rotation="0 45 45" scale="2 2 2"></a-box>
 <a-sky src="#skyTexture"></a-sky>
</a-scene>
```

Додаємо землю

Щоб додати землю, ми будемо використовувати елемент <a-plane>. За замовчуванням площини A-Frame орієнтовані паралельно осі XY. Щоб зробити площину паралельної землі, нам потрібно повернути її так, щоб вона була паралельна осі XZ. Це можна зробити, повернувши площину на -90° по осі X.

```
<a-plane src="#groundTexture" rotation="-90 0 0" width="30" height="30"
 repeat="10 10"></a-plane>
```

Працюємо над світлом

Щоб змінити освітлення нашої сцени, ми можемо додати або переналаштувати елемент <a-light> . За замовчуванням, у нас не було жодних джерел світла, але A-Frame сам додає розсіяне світло (ambient light) і направлене світло (directional light). Якби A-Frame не додав ці джерела світла, сцена була б повністю чорною. Якщо ми додамо власні джерела світла, то джерела, які додає A-Frame за замовчуванням, будуть видалені.

Ми додамо одне джерело розсіяного світла, яке матиме синювато-зелений відтінок (щоб гармоніювати з небом). Розсіяне світло має потрапляти на всі елементи нашої сцени (звичайно якщо вони матимуть матеріали, але за умовчанням це так).

Ми додамо також джерело точкового світла (point light). Джерело точкового світла схоже на лампочку. Ефект освітлення такого джерела повністю залежатиме від дистанції до об'єкта.

```
    <a-scene>
 <a-assets>
   <img id="boxTexture" src="https://i.imgur.com/mYmmbrp.jpg">
   <img id="skyTexture"
     src="https://cdn.aframe.io/360-image-gallery-boilerplate/img/sechelt.jpg">
   <img id="groundTexture" src="https://cdn.aframe.io/a-painter/images/floor.jpg">
 </a-assets>
 <a-box src="#boxTexture" position="0 2 -5" rotation="0 45 45" scale="2 2 2"></a-box>
 <a-sky src="#skyTexture"></a-sky>
 <a-light type="ambient" color="#445451"></a-light>
 <a-light type="point" intensity="2" position="2 4 4"></a-light>
</a-scene>
```

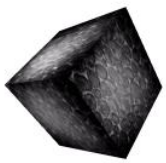
Додаємо анімацію

У нас є можливість додати анімацію, використовуючи вбудовану систему анімації A-Frame. Анімація змінює деяке значення (властивість компонента) з часом. Нам потрібно лише додати елемент <a-animation> як нащадок деякого об'єкта, наприклад <a-box>. Спробуємо анімувати куб таким чином, щоб він рухався вгору-вниз.

```
<a-scene>
 <a-assets>
   <img id="boxTexture" src="https://i.imgur.com/mYmmbrp.jpg">
 </a-assets>
 <a-box src="#boxTexture" position="0 2 -5" rotation="0 45 45" scale="2 2 2">
   <a-animation attribute="position" to="0 2.2 -5" direction="alternate" dur="2000"
     repeat="indefinite"></a-animation>
 </a-box>
</a-scene>
```

Ми говоримо <a-animation>:

- Що потрібно анімувати атрибут position.
- Анімувати до *0 2.2 -5* що на 20 сантиметрів вище за першочергову позицію.
- Змінювати напрямок анімації на кожному циклі.
- Цикл анімації займатиме 2 секунди (2000 мілісекунд).
- Повторювати анімацію нескінченно.



Додаткові деталі

<a-animation> використовує цикл рендерингу A-Frame, тому кожна зміна властивостей об'єкта відбувається один раз за кадр. Якщо вам потрібно більше контролю і ви хочете змінювати значення вручну, ви можете написати компонент A-Frame з колбеком tick та бібліотекою такою як Tween.js (яка, до речі, доступна за посиланням AFRAME.TWEEN). Для кращої продуктивності, покадрові операції повинні бути виконані на рівні A-Frame, не потрібно створити власну функцію *requestAnimationFrame* коли A-Frame **вже має її** .

Додаємо інтерактивність

Давайте додамо можливість взаємодіяти з нашим кубом: коли ми дивимося на нього, ми збільшимо його розміри, а при натисканні він прокрутиться навколо своєї осі.

Припускаючи, що більшість розробників не мають BP шоломів з контролерами, ми сфокусуємося на використанні базового компонента для введення на комп'ютерах і мобільних пристроях — курсорі . Курсор надає можливість "клікнути" на об'єкт через наведення для мобільних пристроїв та натисканням мишки для комп'ютерів. Але треба розуміти, що курсор це лише один із способів взаємодії, все буде трохи інакше якщо у нас будуть справжні BP контролери.

Щоб прив'язати курсор до камери, нам потрібно додати його як нащадок до елемента камери (<a-camera>).

Оскільки ми ще не визначили камеру, A-Frame зробив це автоматично. але оскільки нам потрібно додати курсор до камери, ми визначимо <a-camera> вручну і додамо туди <a-cursor>:

```
    <a-scene>
 <a-assets>
   <img id="boxTexture" src="https://i.imgur.com/mYmmbrp.jpg">
 </a-assets>
 <a-box src="#boxTexture" position="0 2 -5" rotation="0 45 45" scale="2 2 2">
   <a-animation attribute="position" to="0 2.2 -5" direction="alternate" dur="2000"
     repeat="indefinite"></a-animation>
 </a-box>
 <a-camera>
   <a-cursor></a-cursor>
 </a-camera>
</a-scene>
```

Якщо ми подивимося документацію курсора , ми бачимо, що він обробляє події наведення такі як mouseenter, mouseleave а ще й подія click.

Компонент слухача подій

Один із способів обробити події курсору це додати слухач подій через JavaScript просто як для звичайного DOM елемента. Якщо ви не впевнені у своїх знаннях JavaScript, ви можете пропустити цей розділ до наступного.

У *JavaScript* ми отримуємо елемент через querySelector, а використовуючи *addEventListener* , і setAttribute щоб збільшити розмір куба при наведенні курсору. Примітка: A-Frame змінює setAttribute так, щоб він міг працювати відразу з кількома компонентами. Ми можемо додати {x, y, z} як другий аргумент.

```
    <script>
 var boxEl = document.querySelector('a-box');
 boxEl.addEventListener('mouseenter', function () {
   boxEl.setAttribute('scale', {x: 2, y: 2, z: 2});
```

```
});
</script>
```

Але більш швидким способом інкапсулюватиме логіку всередині компонента A-Frame. Цей метод не вимагає очікування завантаження сцени, нам не потрібно використовувати селектори, тому що компонент дає нам контекст:

```
<script>
AFRAME.registerComponent('scale-on-mouseenter', {
  schema: {
    to: {default: '2.5 2.5 2.5'}
  },

  init: function () {
    var data = this.data;
    this.el.addEventListener('mouseenter', function () {
      this.setAttribute('scale', data.to);
    });
  }
});
</script>
```

Ми можемо додати цей компонент через HTML атрибут:

```
<script>
AFRAME.registerComponent('scale-on-mouseenter', {
  // ...
});
</script>

<a-scene>
 <!-- ... -->
 <a-box src="#boxTexture" position="0 2 -5" rotation="0 45 45" scale="2 2 2"
       scale-on-mouseenter="to: 2.2 2.2 2.2">
   <a-animation attribute="position" to="0 2.2 -5" direction="alternate" dur="2000"
     repeat="indefinite"></a-animation>
 </a-box>
 <!-- ... -->
</a-scene>
```

Анімація, подія

<a-animation> має можливість починати та закінчувати свою анімацію, коли об'єкт продукує подію. Це можна зробити через атрибут begin вказавши ім'я події.

Ми можемо додати дві анімації для компонента курсор — *mouseenter* і *onemouseleave* події змінити розміри нашого куба, і один для поворотів куба навколо осі Y за подією click:

```
<a-box color="#FFF" width="4" height="10" depth="2"
      position="-10 2 -5" rotation="0 0 45" scale="2 0.5 3"
      src="#texture">
 <a-animation attribute="position" to="0 2.2 -5" direction="alternate" dur="2000"
   repeat="indefinite"></a-animation>
 <!-- These animations will start when the box is looked at. -->
 <a-animation attribute="scale" begin="mouseenter" dur="300" to="2.3 2.3 2.3"></a-
```

```
animation>
 <a-animation attribute="scale" begin="mouseleave" dur="300" to="2 2 2"></a-animation>
 <a-animation attribute="rotation" begin="click" dur="2000" to="360 405 45"></a-
animation>
</a-box>
```

Додаємо аудіо

Аудіо дуже важливо для повного занурення у віртуальну реальність. Навіть додавання просто білого шуму на задній план може тривати час. Ми рекомендуємо використовувати звук для кожної сцени. Один із способів додавання звуку: додати <audio> елемент у <a-assets> та атрибут autoplay:

```
    <a-scene>
 <a-assets>
   <audio src="https://cdn.aframe.io/basic-guide/audio/backgroundnoise.wav" autoplay
     preload></audio>
 </a-assets>
<!-- ... -->
</a-scene>
```

Або ми можемо додати просторове аудіо за допомогою елемента <a-sound>. Цей компонент робить звук гучнішим коли ми підходимо ближче до джерела. Ми можемо позиціонувати <a-sound> за допомогою атрибута position:

```
<a-scene>
 <!-- ... -->
 <a-sound src="https://cdn.aframe.io/basic-guide/audio/backgroundnoise.wav" autoplay="true"
   position="-3 1 -4"></a-sound>
 <!-- ... -->
</a-scene>
```

Додаємо текст

A-Frame має текстовий компонент. Існує кілька способів його відобразити і кожен спосіб має свої переваги та недоліки. A-Frame має SDF імплементацію через three-bmfont-text, яка є досить продуктивною.

Для цього уроку пропонуємо використовувати найпростішу форму тексту, <a-text>:

```
    <a-text value="Hello, A-Frame!" color="#BBB" position="-0.9 0.2 -3" scale="1.5 1.5
1.5"></a-text>
```