# Take Notes

## Introduction to A-Frame

A-Frame is a popular open-source web framework for building virtual reality experiences in the browser.

It is based on top of HTML, making it super easy to get started. But A-Frame doesn't just produce a static visual 3D scene. The language's core is a powerful entity-component framework that can be programmed extensively by three.js, a JavaScript library used to render 3D graphics.

It was originally developed by the Mozilla VR team in 2015 and it's currently maintained by developers from Supermedium.

In this lesson, we will give a preview of A-Frame and then dive into the basics — HTML & Primitives.

Let's get started!

**Note:** While you can get started with A-Frame without previous coding experience, we recommend taking the time to check out our Learn HTML and Learn JavaScript courses to understand more about what's going on under the hood.

## HTML File

A-Frame can be developed from a plain **.html** file. Because we can link directly to the library, there's no need for downloading any special software and you can get started right away.

So to use A-Frame and create a virtual environment in the browser, we just need to do two things:

- In the `<head>` element, add this line of code:

  ```
  <script src="https://aframe.io/releases/1.0.4/aframe.min.js"></script>
  ```

  The version of A-Frame that we are using in this course is 1.0.4.
  The `<script>` element here points to the external script file through the `src` source attribute.

- In the `<body>` element, add an `<a-scene>` element. The `<a-scene>` element contains every entity and sets up a 3D scene in the browser. This is where we will be writing our A-Frame code.

## Browser

Voilà! A few lines of code and we have ourselves a virtual environment in the browser.

Explore the environment by click-dragging the mouse and move around with `w`, `a`, `s`, `d` or arrow keys.

The A-Frame code is contained inside an `<a-scene>` element that establishes the scene in the browser. Here, we are using a few basic primitives:

- `<a-box>`
- `<a-sphere>`
- `<a-cylinder>`
- `<a-plane>`
- `<a-sky>`

A-Frame *primitives*, such as `<a-box>`, are a wrapper of larger pieces of code that we can easily reuse in our program.

They are great ways for beginners to create 3D objects, also known as *meshes*, right off the bat. They usually have a name that's easy to remember and come with a bunch of components (i.e. `position`, `color`, `width`).

The button on the bottom-right of the browser is for entering the VR mode with a headset or fullscreen mode on a desktop.

## <a-box>

Now that we've seen the power of A-Frame, we can begin to create our own universe. Let's start by learning how to create a primitive from scratch—the `<a-box>` primitive specifically.

We can use `<a-box>` just like a normal HTML element, defining the tag and using HTML attributes to customize it.

To place a box in our world, put an opening and closing `<a-box>` tag inside the `<a-scene>` tag:

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

And suppose we want to give it a color:

```
<a-scene>
  <a-box color="yellow"></a-box>
</a-scene>
```

We can give a `color` value using three different methods:

- Color names: One of the 140 color names.
- Hex codes: A six-digit code representing the amount of red, green, and blue for the color.
- RGB values: A value specified using the `rgb()` property.

See `<a-box>` documentation for more attributes.

## Position

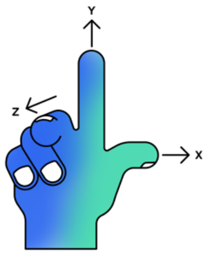
All entities, including the camera, have a `position` component. And because the default camera and the `<a-box>` are positioned at the default x, y, z position, `0` `0` `0`, we won't be able to see the box unless we move it.

We can do this by using the `position` component to transform the box in 3D space. For example:

```
<a-scene>
  <a-box position="2 4 -3"></a-box>
</a-scene>
```

- The x-axis value is `2`
- The y-axis value is `4`
- The z-axis value is `-3`

A-Frame uses a right-handed coordinate system:



With the default camera direction:

- Positive x-axis goes right.
- Positive y-axis goes up.
- Positive z-axis goes out of the screen towards us.

**Note:** A-Frame's distance unit is in meters (m).

## Rotation

Currently, the box looks like a 2D square because it's facing dead center towards the default camera. Let's change that.

To rotate the box, we can change the `rotation` component:

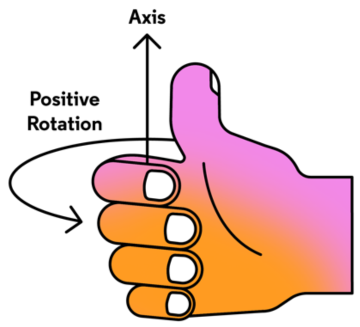
```
<a-scene>
  <a-box
    color="red"
    position="0 2 -5"
    rotation="33 45 0">
  </a-box>
</a-scene>
```

This will rotate our box at an angle:

- The x-axis rotation is 33°
- The y-axis rotation is 45°
- The z-axis rotation is 0°

A-Frame's rotational unit is in degrees, although it will get internally converted to radians when passing to **three.js**.

To determine the positive direction of rotation, use the right-hand rule:



Point our thumbs towards the direction of the positive x, y, or z-axis with your fingers curled. As a reminder:

- Positive x-axis goes right.
- Positive y-axis goes up.
- Positive z-axis goes out of the screen towards us.

The direction in which your fingers curl is the positive direction of rotation. The other way is the negative direction of rotation.

## Scale

The `scale` component defines a shrinking, stretching, or skewing transformation of an entity. It takes three scaling factors for the x, y, and z axes.

Here’s the `scale` component in action:

```
<a-scene>
  <a-box
    color="red"
    position="0 2 -5"
    rotation="0 45 45"
    scale="0.5 2 1">
  </a-box>
</a-scene>
```

- The x-axis value is shrunk by `0.5`
- The y-axis value is stretched by `2`

- The z-axis value is kept the same with `1`

All entities inherently have the `scale` component, so make sure to take advantage of it when designing 3D objects.

## Review

Woohoo! In this lesson, you learned how to create an A-Frame box primitive and change its attributes.

For review:

- A-Frame can be added by using:

```
<script
src="https://aframe.io/releases/1.0.4/aframe.min.js"></scri
pt>
```

- A-Frame code is contained inside an `<a-scene>` element that establishes the scene with the following primitives:
    - `<a-box>`
    - `<a-sphere>`
    - `<a-cylinder>`
    - `<a-plane>`
    - `<a-sky>`
- Each of the primitives can be transformed with the following components:
    - `color`
    - `position`
    - `rotation`
    - `scale`

With these tools, we now have the power to create the building blocks of our VR scene.

To learn more about primitives, visit the A-Frame documentation.

## BUILDING THE SCENE

A-Frame allows developers to create and share reusable components for others to easily use. The environment component procedurally generates a variety of entire environments for us with a single line of HTML. The environment component is a great and easy way to visually fill out our VR app, providing over a dozen environments with numerous parameters. If you're interested in taking a closer look at the component, take a look at the GitHub repo.

```
<html>
  <head>
    <script src="https://aframe.io/releases/1.0.4/aframe.min.js"></script>
    <script src="https://unpkg.com/aframe-environment-component/dist/aframe-
environment-component.min.js"></script>
</head>
  <body>
    <a-scene>
    <!-- Out of the box environment! -->
    <a-entity environment="preset: forest"></a-entity>
    </a-scene>
  </body>
</html>
```

Look at the many presets: forest, contact, Egypt, yavapai, arches, dream etc.

## Asset Management System

We need to build an environment piece by piece, using different images, still and moving, sounds, and 3D models. We call those *assets*.

A-frame has an asset management system that makes complex environments with multiple parts load faster. The system makes it easier for the browser to cache assets (e.g., images, videos, 3D models) and A-Frame will make sure all of the assets are fetched before rendering.

To utilize the asset management system, all we need is an `<a-assets>` element around every HTML element we want to include in the asset management system:

```
<a-scene>
  <a-assets>
    <!-- Asset management system -->
  </a-assets>
  <!-- Rest of <a-scene> -->
</a-scene>
```

Some examples of assets that we can store inside of `<a-assets>` include:

- `<img>`: image files
- `<audio>`: sound files
- `<video>`: video files
- `<a-asset-item>`: 3D model files

If we define an `<img>` in the asset management system, later **three.js** doesn't have to internally create an `<img>`. Creating the `<img>` ourselves also gives us more control and lets us reuse the texture across multiple entities.

To use the asset management system for an image texture:

```
<a-scene>
  <a-assets>
    <img
      id="stones"
      src="https://i.imgur.com/mYmmbrp.jpg">
  </a-assets>

  <a-box
    src="#stones"
    position="0 2 -5"
    rotation="0 45 45"></a-box>
</a-scene>
```

Inside `<a-assets>`:

- Define the texture as an `<img>`.
- Give the `<img>` an `id` (e.g., `id="stones"`).

Outside `<a-assets>`:

- Reference the asset using the `id` in selector format (`src="#stones"`).

Notice how there is an extra hashtag `#` when referencing the `id`.

## Sky and 360° Image

Okay, now we learned how to use the asset management system, let's create the environment little by little.

We can add a background with the <a-sky> primitive that surrounds the scene.

The sky primitive adds a background color or 360° image to a scene. A sky is a large sphere with a color or texture mapped to the inside.

For example, to add a color khaki background:

```
<a-sky color="khaki"></a-sky>
```

To add a 360° background with an image texture, we use `src` instead of `color`:

```
<a-scene>
  <!-- Asset management system -->
  <a-assets>
    <img id="city" src="neotokyo.jpg">
  </a-assets>

  <!-- a-sky -->
  <a-sky src="#city"></a-sky>
</a-scene>
```

In this code above, we are adding the image neotokyo.jpg into the asset management system and then using it in the <a-sky> primitive.

## Ground

To add a ground, we can use the `<a-plane>` primitive. By default, planes are oriented parallel to the x-y axis. To make it parallel to the ground, we need to orient it along the x-z axis.

We can do so by rotating the plane negative 90° on the x-axis:

```
<a-plane
  rotation="-90 0 0"></a-plane>
```

We'll want the ground to be large, so we can increase the width and height. Let's make it 20 meters in each direction:

```
<a-plane
  rotation="-90 0 0"
  width="20"
  height="20"></a-plane>
```

Then we can apply an image texture to our ground:

```
<a-assets>
  <img
    id="groundTexture"
    src="https://cdn.aframe.io/a-painter/images/floor.jpg">
</a-assets>

<a-plane
  src="#groundTexture"
  rotation="-90 0 0"
  width="20"
  height="20"></a-plane>
```

To tile our texture, we can use the `repeat` attribute. `repeat` takes two numbers:

- How many times to repeat in the x-direction
- How many times to repeat in the y-direction

These numbers are commonly referred to as U and V for textures.

```
<a-plane
  src="#groundTexture"
  rotation="-90 0 0"
  width="20"
  height="20"
  repeat="10 10"></a-plane>
```

## Text

To communicate something to the viewer inside the environment, we can use the <a-text> primitive. Texts can be game-like instruction, tell a story, or label objects in the world.

In 2D web development, displaying some text is rather simple because the browser's renderer and layout engine handle everything. In a 3D context, we don't have those luxuries.

There are several ways to render text, each with its own advantages and disadvantages. A-Frame comes with a signed distance field (SDF) text implementation that is relatively sharp, loads faster, and easy to use.

Here's an example using <a-text>:

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

Note that there are two new attributes that we haven't seen:

value="Hello A-Frame!"

align="left"

This will make the text Hello A-Frame! appear in the color gray and align it to the left.

Check out the <a-text> documentation for more info.

## 3D Models

Sometimes we will have 3D assets that we want to add to our scene. For example, a 3D model of a television set, a frog, or a rainbow.

A-Frame provides ways for loading glTF and OBJ files. We recommend using glTF 2.0 if possible, as it is becoming the standard for displaying 3D models over the web since its release in 2015. It uses the JSON standard.

There is no HTML tag for a glTF file, so to import a **.gltf** file into the scene, we need to add the file to the asset management system using an `<a-asset-item>` element.

And then we can point to it using the `<a-gltf-model>` primitive:

```
<a-assets>
  <a-asset-item
    id="tree"
    src="redwood.gltf"></a-asset-item>
</a-assets>

<a-gltf-model src="#tree"></a-gltf-model>
```

Here, the `<a-asset-item>` element is used to load a **redwood.gltf** file and the `<a-gltf-model>` has a `src` attribute that points to the `<a-asset-item>`.

After it's in the scene, we can transform it using `position`, `rotation`, `scale`, etc.

Places to download 3D models as glTF files include Google Poly, Sketchfab, and Clara.io. Programs to create models include Blender, Autodesk Maya, and Supercraft.

**Note:** Importing external 3D models is notoriously difficult. You may need to play around with `position`, `rotation`, and `scale` to get them to look the way you want (or even see it in the first place). Also, sometimes models just don't work! Don't get discouraged and should find other ones.

## Animation

We can add animations to our entities using A-Frame's built-in `animation` component.

Animations are essentially programmed changes in an entity's values. The changes can be based on time intervals or on interactions.

Suppose we have a box in our scene and we want it to hover up and down to add some motion. Check out the `animation` component:

```
<a-box
  color="red"
  position="2 2 -5"
  rotation="0 45 45"
  animation="property: object3D.position.x;
    to: 3;
    dir: alternate;
    dur: 2000;
    loop: true">
</a-box>
```

Note that there are five `animation` properties separated by semicolons. Here, we are telling the animation component to:

- Animate the object `position`'s x-axis.
- Animate from the original x-position to `3`, which is 1 meter to the right.
- Set the `dir` (direction) to `alternate` so it goes back and forth.
- Set the `dur` (duration) to `2000` millisecond on each cycle.
- Set the `loop` to `true` to repeat the animation forever.

Check out the `animation` documentation for more info.

## Lighting

We can change how the scene is lit by using the `<a-light>` primitive. By default, if we don't specify any lights, A-Frame adds a light source. If A-Frame didn't add lights for us, the default scene would be black.

Once we add lights of our own, however, the default lighting setup is removed and replaced with our setup.

`ambient` lights are applied to all entities in the scene. Think of it as the color of the sunlight in your scene (it doesn't have to be yellow!). Below, we're adding an ambient light that has a slight-blue-green-hue:

```
<a-scene>
  <a-light
    type="ambient"
    color="#445451"></a-light>
</a-scene>
```

The other type of light in a scene is the `"point"` light.

Point lights are like light bulbs; we can position them around the scene, and the effect of the point light on an entity depends on its distance to the entity:

```
<a-scene>
  <a-light
    type="point"
    intensity="2"
    position="2 4 4"></a-light>
</a-scene>
```

- The `intensity` of the light. The default is `1`.
- The `position` component is the location of the point light.

Check out the `<a-light>` documentation for more info.

## Sound and Positional Audio

Audio is important for providing immersion and presence in VR. Even adding simple white noise in the background goes a long way. We recommend having some audio for every scene.

One way would be to add an `<audio>` element to our asset management system and then an `<a-sound>` primitive to play the audio file:

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

  <a-sound
    src="#background"
    autoplay="true"></a-sound>
</a-scene>
```

Notice that the `autoplay="true"` component value makes the sound clip play automatically when the user enters the scene.

In the real physical world, sound emits from a source.

So in A-frame, we can add a `position` component to `<a-sound>`. This makes the sound get louder as we approach it and get softer as we distance from it. This is known as *positional audio*.

We could give the sound in our scene a `position` component:

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

  <a-sound
    src="#background"
    autoplay="true"
    position="-3 1 -4"></a-sound>
</a-scene>
```

So when the sound is now coming from the position `"-3 1 -4"`.

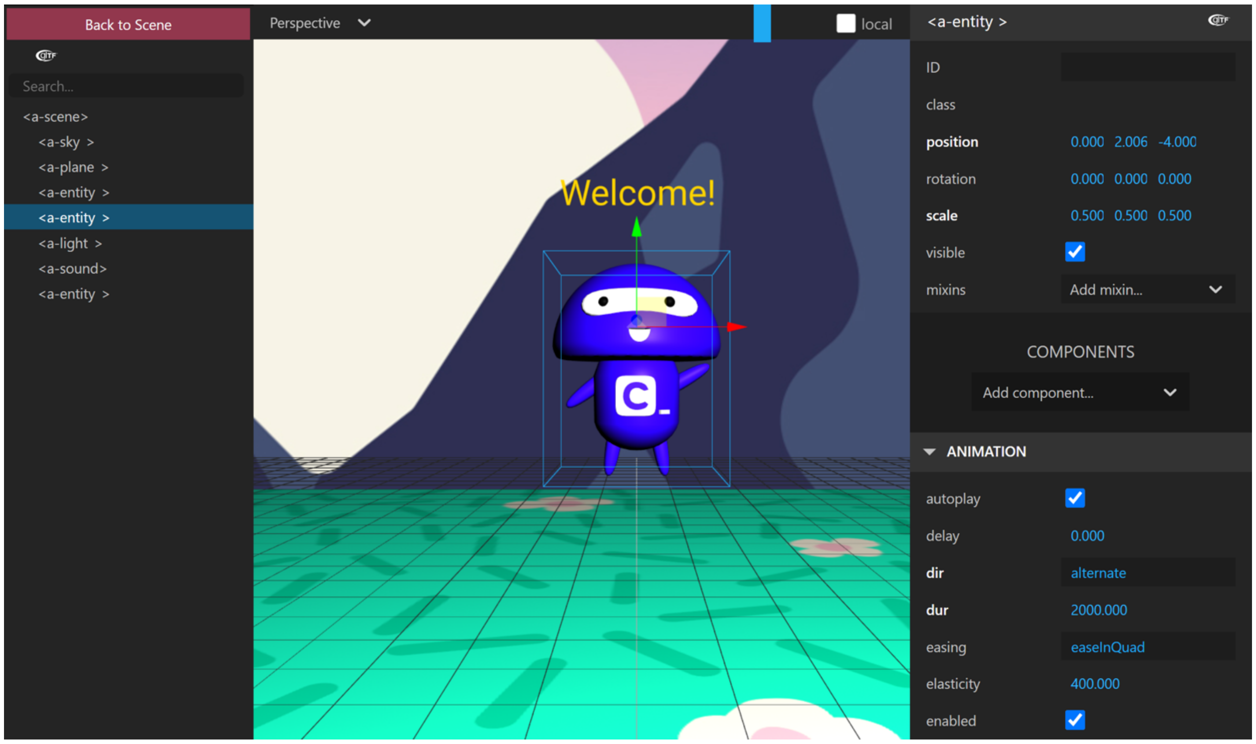
Check out the `<a-sound>` documentation for more info.

## Inspector

Want to edit your scene without guessing desired values and reloading your code each time? The **inspector** helps us do just that.

Similar to the regular web inspector, we can open the A-Frame inspector by clicking inside the browser and holding down a keyboard shortcut:

- Mac: control + option + i
- Windows: ctrl + alt + i



| | |
|---|---|
| ID | |
| class | |
| position | 0.000 2.006 -4.000 |
| rotation | 0.000 0.000 0.000 |
| scale | 0.500 0.500 0.500 |
| visible | ✓ |
| mixins | Add mixin... |

COMPONENTS

Add component...

ANIMATION

| | |
|---|---|
| autoplay | ✓ |
| delay | 0.000 |
| dir | alternate |
| dur | 2000.000 |
| easing | easeInQuad |
| elasticity | 400.000 |
| enabled | ✓ |

The left-hand panel lists all of the entities in the scene. The right-hand panel displays information about a selected entity's properties. Click on one of the entities to see its `position`, `rotation`, `scale`, etc. This panel will also show us if a material or texture is attached to an entity.

Once the inspector is open, we have a couple of options for transforming your entities:

### *In the Inspector Panel*

On the right-hand side of the inspector panel, we can click on a property value and type in a new value, which will immediately transform your entity in the scene. If you're not sure which value you'd like to use, or you just want to play around with positioning, you can use your mouse wheel to scroll through values.

### *In the Scene*

The inspector also allows you to navigate through the scene and manually transform objects. To re-position an entity, click to select it and then use the handlebars to move it around:

- The red handle changes the x position.
- The green handle changes the y position.
- The blue handle changes the z position.

## Instructions

Open up the inspector using the hotkeys. And take a moment to look through our list of entities.

Try doing the following:

- Move the text a little bigger and higher.
- Move the character so that it's closer to the default camera.
- Rotate the scene so that you're looking up at the character through the ground.

Just remember — if you make any changes in the inspector, they won't carry over unless you copy the new values over into your code!

## Review

You made it to the end of the lesson! High five. 🙌

Here is a review of the lesson:

- The asset management system is where we can store textures, audio, and 3D model files inside `<a-assets>`.
- `<a-sky>` for adding the sky or 360° image.
- `<a-plane>` for adding the ground.
- `<a-light>` for adding the lighting.
- `<a-sound>` for adding the sound.
- `<a-text>` for adding text.
- `<a-gltf-model>` for adding 3D models.
- `animation` component for adding animation.

And an in-browser tool called the inspector that move around objects. To open the inspector:

- control + option + I (Mac)
- ctrl + alt + I (Windows)