3D Animations within your website

STEPS:

File prep:

- create your index.html, style.css & app.js.
    - link css & script into your html
    - Go to threejs.org and download the main zip.
    - When downloaded, head to following locations and copy those files into your project folder
        - build/three.min.js
        - examples/js/loaders/GLTFLoader.js

Three.mini.js & GLTFLoader will allow us to load up the 3Dmodels with the GLTF extension.

HTML/Canvas Prep:

- Add div with .scene class in body.
    - This is where we load up our 3D animation
- Link Three.min.js & gltf.js to your index.html as well.
- Go to style.css and add following properties:
    - Body {
        - Font-family (in case for later)
        - Overflow: hidden (if model would be too big)
    - .scene, canvas {
        - Position: absolute;
        - Width: 100%;
        - Height: 100%;
        - (Three.js will automatically add a canvas to scene later on)

3D Model prep:

Get a GLTF 3D model from a legitimate site (sketchfab, etc…). For now use a free model and download with GLTF extension (creating an account may be necessary)

When the file is downloaded we will need next files:

- Scene.bin
- Scene.glth
- (sometimes textures)

First JS steps:

Build your variables

- Const camera, container, scene, renderer, object/robot/nameofyourmodel

We're gonna make a function called init, wherein we will define all are needed variables with the threeJS properties and so on, which we will call later on to execute when the page is loaded;

Function init(){

};

First we will define our container, which in this case is .scene ->
: container = document.querySelector('.scene');

Now to create a scene and define our settings for our model:

- Create scene:
  - Scene = new THREE.Scene();
- Setup camera variables:
  - Const fov = 35;
    - Field of view, the angle in which we see thinks (us humans can see I think around 120° us, a smaller field of view is usually better when the screen is further away, feel free to experiment what works best for you)
  - Const aspect = container.clientWidth / container.clientHeight
    - Think aspect ratio like for monitors, so we don't for example get a very wide and fat, but smushed down model. This wil make the scene scale nicely in regards to the size of your client/browser
  - Const near = 0.1;
  - Const far = 10000;
    - Clipping near & far makes it so that when an object is further than your near or far values, your model wil clip out/away. The same as when an image will clip out (like overflow hidden) when it gets out of its parents boundaries.
    - In this case we don't really care if it does as we will only load 1 model, so we can put it at very near and very far so it will never clip out.
- Actual Camera setup:
  - Camera = new THREE.PerspectiveCamera(fov, aspect, near, far);
    - These parameters are the variables we just defined, so we can give them to the threejs.property.
  - Camera.position.set(0, -1, 1500);
    - The actual position of the camera, these are X, Y & Z values, think a 3D chart. Something to play around with until you get it right how you want. We will or can adjust this later on.

Rendering setup and loading our model for the first time:

- Now we will setup what space or canvas we actually want to render.
  - Renderer = new THREE.WebGLRenderer ({antialias:true});
    - We can add an object to our argments and 'turn on' the antialias function of the renderer. If you don't know antialias, it's a technique to smooth down curved or jagged edges. As you know screens and images are built out of pixels, 3D models out of polygons, they all have sharp and hard edges, this technique smooths it out so that you don't have weird lines.
- Now we will define the size of our rendered canvas:
  - Renderer.setSize(container.clientWidth, container.clientHeight);
    - Here we tell the renderer to always use/render the clients full size, whatever size it is on whatever device.
- Now we set the pixelratio
  - Renderer.setPixelRatio(window.devicePixelRatio);
    - This will make sure the correct amount of pixeldensity/ratio is used to display this model on every monitor, an older monitor will not have the same pixelDensity as brand new monitor.
- Now that we know what canvas we want to render: we want to actually put it inside our HTML, making it visible to work our 3D model on.
  - Container.appendChild(renderer.domElement);

- Loading up our model for the first time:
    - We're gonna create a loader variable and a function to load the actual model. So:
        - Let loader = new THREE.GLTFLoader();
            - Loader.load('./3D/scene.gltf', function (gltf) {
            - We 'pull the 3Dmodel into our website) and define the following function when we do so.
    - Now we add the model itself to our scene:
        - Scene.add(gltf.scene);
    - Now we render our actual scene and camera with their previously set values;
        - Renderer.render(scene, camera);

Call our function for the first time:

Under the whole function init, we now call our function , so just type: init();

Now we won't see anything, why is that? We actually have a standard background we cannot change in 2D; So we will add an alpha channel to our background, so we can make it whatever we want.

- Add alpha:true, as parameter to renderer object (with antialias);
- This will make it transparent so we can see any background we might add later in css.

Now we should be able to actually see our model. However it is completely black. You can however already play with position in the camera.position.set value to your liking.

Light setup

But we will need a lightsource, as in reallife to actually see our model in 3D.

We'll add those now. There are different kind of lighting properties we can assign to the scene, but usually we'll be using these 2: (between camera & renderer)

- Ambient lighting
    - Which is a global illumination, it will not take shading or placement in consideration and will illuminate your model every exactly the same. This will allow us to actually see our model.
- Directional lighting.
    - The directional lighting is our actual lightsource (pointlight also exists but is a little bit trickier to get right, so we're using the easy way now)
- Setup of the light:
    - Const ambient = new THREE.AmbientLight(0x404040, 1);
        - 0x is so that our IDE won't read our colorcode as an argument;
        - The 1 is the 'amount of light' we want. So we can go very bright or very dark.
        - (Ambient light might not work with every model, it is a simple light based function dependant on material settings made by the modeler if the light is reflected or not)
    - Scene.add(ambient);
        - Now we add our ambient light to our scene.
        - Lets load our model once again and look at the result.

Try out animation

Now we can actually see our model, we'll add a quick animation so that it's easier for us to see if the directionalLight looks right on our model or not, can be hard on an unmoving object.

Inside load 3Dmodel (under scene.add(gltf.scene); function:

- Robot = gltf.scene.children[0];
    - We take our actual object, so in our case the robot, and define it as the actual model (the children array should be our data containing our model);

- Now we create a function (outside of function init, above init(); function call) to animate our actual model:
    - Function animate(){
        - requestAnimationFrame(animate);
            - (this makes us run this function over & over & over again)
        - House.rotation.z += 0.005;
            - This wil rotate our model per 0.005 frame, so it will look very smooth.
    - Now instead of rendering our image once, as we have done before, we will cut the following:
        - Renderer.render(scene, camera) to our animate() function;
    - Now we replace the cut line with animate();

Add directional lightsource

now lets do the actual lightsource: lets go back to our light setup:

-
    - const light = new THREE.DirectionalLight(0xffffff, 5)
        - (5 will be to light, replace with 1 or 2 later on)
    - Here we use the same as with our ambient light, our color and our light strength.
- Now we will have to add the position of our lightsource, so:
    - Light.position.set(10,10,2) (again an X Y Z index)
- And add the light to our scene
    - Scene.add(light);

We are done! Lets take a look at our model now.
maybe add a quick background

If there is a minute more time, I will add some quick code to fix an issue I have had before with resizing your window regarding the model: it will scale very very weirdly. So lets create a function to get rid of it.

Window resizing

Add function:

- Function windowResize (){
    - Camera.aspect = container.cliendtWidth / container.clientHeight;
    - Camera.updateProjectionMatrix();
    - Renderer.setSize(container.clientWidth, container.clientHeight);
- }

Now to call our function

- Window.addEventListener('resize', windowResize);