4 properties to change an object

Scale

Position

Rotation

Quaternion

All objects that inherit from the Object3D posses those properties like PerspectiveCamera or Mesh. You can find more about inheritance in the Three.js docs

These properties will be compiled into matrices

Position has three properties: x, y and z

X is right and left and y is up and down and Z is to and from you, BECAUSE we are natively on an overhead view for three.js

Also, 1 or units can be anything. It is up to you.

mesh.position.x = 0.7;

mesh.position.y = -0.6;

mesh.position.z = 1;

playing with the mesh properties makes the cube appear different places within the scene or canvas?

Basically the .7 on x goes right, the -.06 y makes it go downward, and the 1 makes the cube come slightly toward us.

This code does need to go before the render of the scene or the properties will not be updated.

Renderer.render actually “take the picture”. So if you ask someone to move after you take a picture, you will not see it.

ALSO, can not do these changes before creating the variable, so it is best to do right after the variable is created.

Position is an inherited property from Vector3 which has many useful methods. You can get length of a vector.

Position is actually apart of Vector3 which is a class that you can use to position things in space. In three.js there is vector 2, 3 and 4.

Vector3 is more than just position, it has a lot of methods available.

console.log(mesh.position.length());

can use properties to find the distance from the center of the scene and the objects position.

What about distance form the camera and the object?

console.log(mesh.position.distanceTo(camera.position))

want to normalize objects?

mesh.position.normalize();

this will take the vector length and change it so that it is one.

It will reduce the vector until the length is one.

What if you want to change all 3 properties of an object at once?

Example: mesh.position.set(x,y,z)

mesh.position.set(0.7, -0.6, 1);

positioning an object in space is hard.

Use the axes helper which is a class in Threejs

const axesHelper = new THREE.AxesHelper(2);

scene.add(axesHelper);

adds an axes helper to the screen. The “2” is the length of the axes you will see.

Now lets scale an object. Is also Vector3.

mesh.scale.x = 2;

mesh.scale.y = 0.5;

mesh.scale.z = 0.5;

examples of scaling a mesh

mesh.scale.set(2, 0.5, 0.5);

can also do it with set.

Now lets talk about rotation. Can use rotation property or use quaternion. Updating one will update the other.

Rotation has properties of x, y and z but it’s a Euler.

mesh.rotation.y = 3.14159;

to do half a rotation on the r axes. But you can also use Math.PI.

can do a quarter rotation by using:

mesh.rotation.y = Math.PI \* 0.5;

or

mesh.rotation.y = Math.PI / 2;

be careful when you rotate things on an axis. You might also rotate other axis. The rotation goes by default which is x, y then z. in order and you can get strange results like an axis not working anymore.

This is called gimbal lock.

You can change the order or fix this by using:

reorder() method => object.rotation.reorder(‘yxz’)

Do this before changing the rotation.

mesh.rotation.reorder("YXZ");

mesh.rotation.x = Math.PI \* 0.25;

mesh.rotation.y = Math.PI \* 0.25;

doing the reorder after will not fix the order.

Euler is easy to understand but this axis order can be problematic. This is why most engines and 3D software use Quaternion.

It is like a more mathematical type of rotation. It is hard to imagine compared to Euler.

You can also tell any object3D to look at something. With lookAt(…) which will rotate the object so that its -z faces the target you provide. The target must be a Vector3

camera.lookAt(new THREE.Vector3(0, -1, 0));

this will make our camera look at something.

camera.lookAt(mesh.position);

or make the camera look at our box by referencing its position!

Scene graph.

Sometimes we created complex objects. Might decide things are too small or not at the right place.

If you create a group, and put everything in the group, it will all change together.

Put things in groups to use scale and rotation and quaternion all together. This is how you move and animation all together!

Do this with the Group class

const group = new THREE.Group();

scene.add(group);

now lets create three cubes in the group and rotate them all.

const group = new THREE.Group();

scene.add(group);

const cube1 = new THREE.Mesh(

  new THREE.BoxGeometry(1, 1, 1),

  new THREE.MeshBasicMaterial({ color: 0xff0000 })

);

group.add(cube1);

here is the first cube created with a geometry and mesh, and then added to the group to appear on the screen.

After adding similar cubes, we can manipulate the group like:

const group = new THREE.Group();

group.position.y = 1;

group.scale.y = 2;

scene.add(group);

or rotate the entire group:

group.rotation.y = Math.PI / 5;