Animation is like doing stop motion.

Most screens run at 60 FPS (frame per second), but not always. Your animation must look the same regardless of the framerate.

We need to update objects and do a render on each frame. We are going to do that in a function and call this function with window.requestAnimationFrame(…)

Request animation frame:

The purpose is to call the function provided on the next frame.

We are going to call the same function on each new frame.

We can use new Data() to track the time between ticks and manage the update function to keep things all uniform. This is because without a time variable the framerate of the computer dictates how fast the animation is happening.

const tick = () => {

  //time

  const currentTime = Date.now();

  const deltaTime = currentTime - time;

  time = currentTime;

  console.log(deltaTime);

  //Update Object

  mesh.rotation.y += 0.001 \* deltaTime;

  //   mesh.rotation.x -= 0.01;

  // Render

  renderer.render(scene, camera);

  window.requestAnimationFrame(tick);

};

tick();

here is a quick example of using the time function of new Date() and finding the difference between the previous ticks to change your animation. This works to keep the site or animation relative to a computer’s framerate and not have things go too fast or too slow.

Three.js has another built in solution called clock that we can also use.

Instantiate a Clock and use its getElapsedTime() function method

const clock = new THREE.Clock();

// Animations

const tick = () => {

  //time first solution

  //   const currentTime = Date.now();

  //   const deltaTime = currentTime - time;

  //   time = currentTime;

  //clock

  const elapsedTime = clock.getElapsedTime();

  console.log(elapsedTime);

  //   console.log(deltaTime);

  //Update Object

  mesh.rotation.y = elapsedTime;

  //   mesh.rotation.x -= 0.01;

  // Render

  renderer.render(scene, camera);

  window.requestAnimationFrame(tick);

};

This is how to use the clock function to get elapsed time.

Want to rotate the cube one full rotation per second? Here is what that would look like.

 mesh.rotation.y = elapsedTime \* Math.PI \* 2;

This accomplishes that from when the clock starts, the cube will rotate a full rotation every second from the start.

Make a cube go away in the z value?

  mesh.position.z = -elapsedTime;

make a cube go up and down?

  mesh.position.y = Math.sin(elapsedTime);

add that to the z value change and you have a cube going up and down while going away just like a game!

How about making something go in a circle?

  mesh.position.y = Math.sin(elapsedTime);

  mesh.position.x = Math.cos(elapsedTime);

using a sin on the y and cos on the x makes an object go in a circle.

But we can also animate the camera and use getDelta from clock.

DO NOT USE GETDELTA – messes with interval of clock.

  camera.position.y = Math.sin(elapsedTime);

  camera.position.x = Math.cos(elapsedTime);

this will look like the same result but it will actually be the camera moving around the cube instead of the cube moving

now make the camera look at the cube.

  camera.position.y = Math.sin(elapsedTime);

  camera.position.x = Math.cos(elapsedTime);

  camera.lookAt(mesh.position);

now the cube is still but the camera looks like its going around the cube looking at it from different angles.

Now lets use a library to get more control.

If you want more control, create timelines, create tweens, etc you can use GSAP

Add the green sock library by:

npm install --save [gsap@3.5.1](mailto:gsap@3.5.1)

then

import gsap from "gsap";

now we can move things on a specified timeline:

gsap.to(mesh.position, {

  duration: 1,

  delay: 1,

  x: 2,

});

This makes the cube wait a second, and then take a second to move to the x-2 position. To move the cube back after this add another similar script:

gsap.to(mesh.position, {

  duration: 1,

  delay: 2,

  x: 0,

});

Important to remember that the green sock library has its own tick, so you do not need to tell greensock to update itself.

But you still need to run the render results by yourself.