First create ray caster

To shoot a ray we need to tell the origin and direction of ray

The direction has to be normalized for a raycaster to work.

To cast we can call intersect object or objects

const raycaster = new THREE.Raycaster();

const rayOrigin = new THREE.Vector3(-3, 0, 0);

const rayDirection = new THREE.Vector3(10, 0, 0);

rayDirection.normalize();

raycaster.set(rayOrigin, rayDirection);

const intersect = raycaster.intersectObject(object2);

const intersects = raycaster.intersectObjects([object1, object2, object3]);

Here is how to implement into tick with moving balls and change their color

  object1.position.y = Math.sin(elapsedTime \* 0.3) \* 1.5;

  object2.position.y = Math.sin(elapsedTime \* 0.8) \* 1.5;

  object3.position.y = Math.sin(elapsedTime \* 1.4) \* 1.5;

  // cast a ray

  const rayOrigin = new THREE.Vector3(-3, 0, 0);

  const rayDirection = new THREE.Vector3(1, 0, 0);

  rayDirection.normalize();

  raycaster.set(rayOrigin, rayDirection);

  const objectsToTest = [object1, object2, object3];

  const intersects = raycaster.intersectObjects(objectsToTest);

  for (const object of objectsToTest) {

    object.material.color.set("#ff0000");

  }

  for (const intersect of intersects) {

    intersect.object.material.color.set("#0000ff");

  }

Now we work on hover – Bruno likes normalizing values

const mouse = new THREE.Vector2();

window.addEventListener("mousemove", (event) => {

  mouse.x = (event.clientX / sizes.width) \* 2 - 1;

  mouse.y = -(event.clientY / sizes.height) \* 2 + 1;

  console.log(mouse.x, mouse.y);

});

This tracks the mouse and makes top right + 1 + 1 for normalized value and bottom left -1 -1 for x and y for example. Also, a Vector 2 just creates {x:0, y:0} so don’t over think what it is.

Then we add this stuff back to the code:

  raycaster.setFromCamera(mouse, camera);

  const objectsToTest = [object1, object2, object3];

  const intersects = raycaster.intersectObjects(objectsToTest);

  for (const object of objectsToTest) {

    object.material.color.set("#ff0000");

  }

  for (const intersect of intersects) {

    intersect.object.material.color.set("#0000ff");

  }

Now on hover the spheres change color

Now lets work on the mouse enter and mouse leave events

We create a witness variable.

This is how to setup a bacis mouse enter and leave event trigger

  if (intersects.length) {

    if (currentIntersect === null) {

      console.log("mouse enter");

    }

    currentIntersect = intersects[0];

  } else {

    if (currentIntersect) {

      console.log("mouse leave");

    }

    currentIntersect = null;

  }

You also need to define currentIntersect before the tick function as null.

What about mouse click event?

window.addEventListener("click", () => {

  if (currentIntersect) {

    console.log("click on a sphere");

  }

});

This basically is like a shooting game now only triggered if hovered over a sphere and clicked

window.addEventListener("click", () => {

  if (currentIntersect) {

    if (currentIntersect.object === object1) {

      console.log("click on object 1");

    } else if (currentIntersect.object === object2) {

      console.log("click on object 2");

    } else if (currentIntersect.object === object3) {

      console.log("click on object 3");

    }

  }

});

This code specifies which sphere was clicked