Subdivision gives you more triangles to have more details on a plane.

We can add triangles to a mesh by doing

const geometry = new THREE.BoxGeometry(1, 1, 1, 2, 2, 2);

the 2’s represent the amount of additional triangles. So now every side has 8 triangles instead of 2

Now we are going to add a wire frame to view this.

const material = new THREE.MeshBasicMaterial({

  color: 0xff0000,

  wireframe: true,

});

Most computers will have a thickness of 1 for the individual wires.

We are going to create a bunch of triangles with a buffer geometery. But we need to understand how to store the data first.

WE are going to use Float32Array

Typed array, can store only floats, fixed length (determined length on creation), easy for comp to handle. A typed array can only store one type of data.

Here is one way to do this

const positionsArray = new Float32Array(9);

positionsArray[0] = 0;

positionsArray[1] = 0;

positionsArray[2] = 0;

positionsArray[3] = 0;

positionsArray[4] = 1;

positionsArray[5] = 0;

positionsArray[6] = 1;

positionsArray[7] = 0;

positionsArray[8] = 0;

Or like this:

const positionsArray = new Float32Array([0, 0, 0, 0, 1, 0, 1, 0, 0]);

now we need to convert this to a buffer attribute

const positionsAttribute = new THREE.BufferAttribute(positionsArray, 3)

the 3 is to show that one vertex contain three values, x y and z

Now we send the attribute to the geometry.

const geometry = new THREE.BufferGeometry();

geometry.setAttribute("position", positionsAttribute);

the ‘position” is just a name we send in for the geometry. Although, this is also the name that will be used inside of the shaders. We will learn more about these later. So jk we actually have to use the name position

Also, we did not provide faces.

Now lets create a bunch of triangles!

Basic setup will look like

const geometry = new THREE.BufferGeometry();

const count = 50;

const positionsArray = new Float32Array(count \* 3 \* 3);

if we want 50 triangles, we need 50 \* 3 vertex, \* 3 values

so

const positionsAttribute = new THREE.BufferAttribute(positionsArray, 3);

Takes in an array of values and needs to know how many vertices and values per triangle or whatever it is making.

We already have our

const geometry = new THREE.BufferGeometry();

and we set that to

geometry.setAttribute("position", positionsAttribute);

our buffer attribute with the name position.

Here is full code with the array loop we made:

const geometry = new THREE.BufferGeometry();

const count = 50;

const positionsArray = new Float32Array(count \* 3 \* 3);

for (let i = 0; i < count \* 3 \* 3; i++) {

  positionsArray[i] = (Math.random() - 0.5) \* 3;

}

const positionsAttribute = new THREE.BufferAttribute(positionsArray, 3);

geometry.setAttribute("position", positionsAttribute);

final note about index. Some geometry have faces that share common vertices.

Some geometrys can share a vertex and you can provide an index to save space and the computer from having to reproduce stuff. We wont do that in this course.