Working with Geometries in Three.js

Eclipse Group
Amelia Mumtazah Karimah - 5025201128
Farzana Afifah Razak - 5025201130
Rahel Cecilia Purba - 5025201155

Table of Content

The Kinds Geometries provided by Three.js

2D Geometries

2D objects look like flat objects and, as the name implies, only have two dimensions. In this geometries, we'll first look at the 2D geometries :

- **`THREE.CircleGeometry`**
- **`THREE.RingGeometry`**
- **`THREE.PlaneGeometry`**
- **`Three.ShapeGeometry`**

3D Geometries

3D objects object or shape that has three dimensions—length, width, and height.. In this geometries, we'll first look at the 3D geometries :

- **`THREE.ConeGeometry`**
- **`THREE.CapsuleGeometry`**
- **`THREE.CylinderGeometry`**
- **`THREE.TorusGeometry`**

Properties in THREE.Geometry

With a THREE.Geometry, the shape of an object is defined by the following properties :

Property	Description
`vertices`	The vertices array defines the position of each individual vertex of the model. Vertices are the points that make up a model.
`faces`	When combining three vertices together, you get a face. This array contains all the faces of the model.

2D Geometry

2D geometry in computer graphics is the creation of an image object using 2 points as a reference, namely x and y. Can be used in applications originally developed on traditional printing and image technologies, such as typography, cartography, technical drawing, advertising, and others. In applications, two-dimensional images are not just representations of real-world objects, but independent artifacts with value-added semantics; Therefore, two-dimensional models are preferred, as they provide more direct image control than 3D computer graphics, an approach that is more similar to photography than typography.

2D Geometries

1. Circle Geometry

CircleGeometry is a simple shape of Euclidean geometry. It is constructed from a number of triangular segments that are oriented around a central point and extend as far out as a given radius. It is built counter-clockwise from a start angle and a given central angle. It can also be used to create regular polygons, where the number of segments determines the number of sides.

Code

```
const geometry = new THREE.CircleGeometry( 5, 32 );
const material = new THREE.MeshBasicMaterial({color: 0xffff00});
const circle = new THREE.Mesh( geometry, material );
scene.add( circle );
```

2D Geometries

2. Ring Geometry

In mathematics, rings are algebraic structures that generalize fields: multiplication need not be commutative and multiplicative inverses need not exist. In other words, a ring is a set equipped with two binary operations satisfying properties analogous to those of addition and multiplication of integers. Ring elements may be numbers such as integers or complex numbers, but they may also be non-numerical objects such as polynomials, square matrices, functions, and power series.

Code

```
const geometry = new THREE.RingGeometry( 1, 5, 32 );
const material = new THREE.MeshBasicMaterial
( { color: 0xffff00, side: THREE.DoubleSide } );
const mesh = new THREE.Mesh( geometry, material );
scene.add( mesh );
```

2D Geometries

3. Plane Geometry

PlaneGeometry deals with flat shapes which can be drawn on a piece of paper. These include lines, circles & triangles of two dimensions. Plane geometry is also known as two-dimensional geometry.All the two-dimensional figures have only two measures such as length and breadth. It does not deal with the depth of the shapes. Some examples of plane figures are square, triangle, rectangle, circle, and so on.

Code

```
const geometry = new THREE.PlaneGeometry( 1, 1 );
const material = new THREE.MeshBasicMaterial( {color:
0xffff00, side: THREE.DoubleSide} );
const plane = new THREE.Mesh( geometry, material );
scene.add( plane );
```

2D Geometries

4. Shape Geometry

A shape or figure is a graphical representation of an object or its external boundary, outline, or external surface, as opposed to other properties such as color, texture, or material type. A plane shape or plane figure is constrained to lie on a plane, in contrast to solid 3D shapes. A two-dimensional shape or two-dimensional figure may lie on a more general curved surface (a non-Euclidean two-dimensional space).

Code

```
const x = 0, y = 0; const heartShape = new THREE.Shape(); heartShape.moveTo( x + 5, y + 5 );
heartShape.bezierCurveTo( x + 5, y + 5, x + 4, y, x, y ); heartShape.bezierCurveTo( x - 6, y, x - 6, y +
7,x - 6, y + 7 ); heartShape.bezierCurveTo( x - 6, y + 11, x - 3, y + 15.4, x + 5, y + 19 );
heartShape.bezierCurveTo( x + 12, y + 15.4, x + 16, y + 11, x + 16, y + 7 );
heartShape.bezierCurveTo( x + 16, y + 7, x + 16, y, x + 10, y ); heartShape.bezierCurveTo( x + 7, y, x +
5, y + 5, x + 5, y + 5 );
const geometry = new THREE.ShapeGeometry( heartShape );
const material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
const mesh = new THREE.Mesh( geometry, material ) ; scene.add( mesh );
```

3D Geometry

3D Geometry in computer graphics are graphics that use a three-dimensional representation of geometric data (often Cartesian) that is stored in the computer for the purposes of performing calculations and rendering 2D images. The resulting images may be stored for viewing later (possibly as an animation) or displayed in real time. The objects in 3D computer graphics are often referred to as 3D models. Unlike the rendered image, a model's data is contained within a graphical data file. A 3D model is a mathematical representation of any three-dimensional object; a model is not technically a graphic until it is displayed. A model can be displayed visually as a two-dimensional image through a process called 3D rendering, or it can be used in non-graphical computer simulations and calculations.

3D Geometries

1. Cone Geometry

A **cone** is a three-dimensional shape in geometry that narrows smoothly from a flat base (usually circular base) to a point(which forms an axis to the centre of base) called the apex or vertex. We can also define the cone as a pyramid which has a circular cross-section, unlike pyramid which has a triangular cross-section. These cones are also stated as a circular cone.

Code

```
const geometry = new THREE.ConeGeometry( 5, 20, 32 );
const material = new THREE.MeshBasicMaterial
( {color: 0xffff00} );
const cone = new THREE.Mesh(geometry, material );
scene.add( cone );
```

3D Geometries

2. Capsule Geometry

A **capsule** (from Latin capsula, "small box or chest"), or stadium of revolution, is a basic three dimensional geometric shape consisting of a cylinder with hemispherical ends. Another name for this shape is spherocylinder.

Code

```
const geometry = new THREE.CapsuleGeometry( 1, 1, 4, 8 );
const material = new THREE.MeshBasicMaterial( {color:
0x00ff00} );
const capsule = new THREE.Mesh( geometry, material );
scene.add( capsule );
```

3D Geometries

3. Cylinder Geometry

A **cylinder** is a 3D solid shape that consists of two identical and parallel bases linked by a curved surface. These bases are like circular disks. The line passing from the center or joining the centers of two circular bases is called the axis of the cylinder shape.

Code

```
const geometry = new THREE.CylinderGeometry( 5, 5, 20, 32);
const material = new THREE.MeshBasicMaterial( {color:
0xffff00} );
const cylinder = new THREE.Mesh( geometry, material );
scene.add( cylinder );
```

3D Geometries

4. Torus Geometry

In Mathematics, a **torus** is a doughnut-shaped object such as an O ring. It is a surface of an object formed by revolving a circle in three-dimensional space about an axis that lies in the same plane as the circle. If the axis of revolution does not touch the circle, the surface forms a ring shape known as ring torus or simply torus if the ring shape is implicit. As the distance from the axis of revolution minimizes, then the ring torus transforms into a horn torus.

Code

```
const geometry = new THREE.TorusGeometry( 10, 3, 16, 100);
const material = new THREE.MeshBasicMaterial( { color:
0xffff00 } );
const torus = new THREE.Mesh( geometry, material );
scene.add( torus );
```

Examples

1. Before we start, Before you can use three.js, you need somewhere to display it. Save the following HTML to a file on your computer, along with a copy of three.js in the js/ directory, and open it in your browser.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>My first three.js app</title>
        <style>
            body { margin: 0; }
        </style>
    </head>
    <body>
        <script src="js/three.js"></script>
        <script>
            // Our Javascript will go here.
        </script>
    </body>
</html>
```

2. ***Create Scene*** to actually be able to display anything with three.js, we need three things: scene, camera and renderer, so that we can render the scene with camera.

```
const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera( 75, window.innerWidth /
window.innerHeight, 0.1, 1000 );

const renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );
```

There are a few different cameras in three.js. For now, let's use a PerspectiveCamera. ***The first*** attribute is the field of view. FOV is the extent of the scene that is seen on the display at any given moment. The value is in degrees. ***The second one*** is the aspect ratio. You almost always want to use the width of the element divided by the height, or you'll get the same result as when you play old movies on a widescreen TV - the image looks squished. The next two attributes are the near and far clipping plane. What that means, is that objects further away from the camera than the value of far or closer than near won't be rendered. In addition to creating the renderer instance, we also need to set the size at which we want it to render our app. It's a good idea to use the width and height of the area we want to fill with our app - in this case, the width and height of the browser window. For performance intensive apps, you can also give setSize smaller values, like window.innerWidth/2 and window.innerHeight/2, which will make the app render at quarter size.

3. **Create Geometry**

```
const geometry = new THREE.BoxGeometry( 1, 1, 1 ); const material = new
THREE.MeshBasicMaterial( { color: 0x00ff00 } ); const cube = new THREE.Mesh(
geometry, material ); scene.add( cube ); camera.position.z = 5;
```

To create a cube we need a BoxGeometry. This is an object that contains all the points (vertices) and fill (faces) of the cube. In addition to the geometry, we need a material to color it. Three.js comes with several materials, but we'll stick to the MeshBasicMaterial for now. All materials take an object of properties which will be applied to them. To keep things very simple, we only supply a color attribute of (0x00ff00), which is green. This works the same way that colors work in CSS or Photoshop (hex colors). The third thing we need is a Mesh. A mesh is an object that takes a geometry, and applies a material to it, which we then can insert to our scene, and move freely around.

4. **Rendering the scene**

```
function animate() { requestAnimationFrame( animate ); renderer.render( scene,
camera ); } animate();
```

This will create a loop that causes the renderer to draw the scene every time the screen is refreshed (on a typical screen this means 60 times per second). If you're new to writing games in the browser, you might say *"why don't we just create a setInterval ?"* The thing is - we could, but requestAnimationFrame has a number of advantages. Perhaps the most important one is that it pauses when the user navigates to another browser tab, hence not wasting their precious processing power and battery life.

5. **Animating The Cube**

```
cube.rotation.x += 0.01; cube.rotation.y += 0.01;
```

Add the following code right above the renderer.render call in this animate function. This will be run every frame (normally 60 times per second), and give the cube a nice rotation animation. Basically, anything you want to move or change while the app is running has to go through the animate loop. You can of course call other functions from there, so that you don't end up with an animate function that's hundreds of lines.