

Instituto Tecnológico Superior de Tierra Blanca

Semestre Agosto 2021 Enero 2022

Ingeniería en Sistemas Computacionales

ISIC 2010-224

SCC-1010 Graficación

**Unidad V Introducción a la animación por
computadora**

**Reporte de practica | Colisiones entre mallas de
polígonos**

Blanca Ramírez Cruz 198N0530

Sergio Jared Valencia Cortaza 198N0068

Tierra Blanca Veracruz



Introducción

La implementación de las físicas en los modelos 3D ha sido un impacto muy grande en el mundo de la graficación y el diseño. Claramente nos otorga más posibilidades de experimentar y mejorar los modelos 3D a otro nivel, llevando nuestra imaginación más allá de lo posible.

Esta implementación se ve frecuentemente reflejada en los videojuegos y apps, realidad virtual y realidad aumentada, películas animadas y en gran parte de la industria del entretenimiento. Ya que resulta ser de mucha ayuda para el constante avance del mundo del 3D.

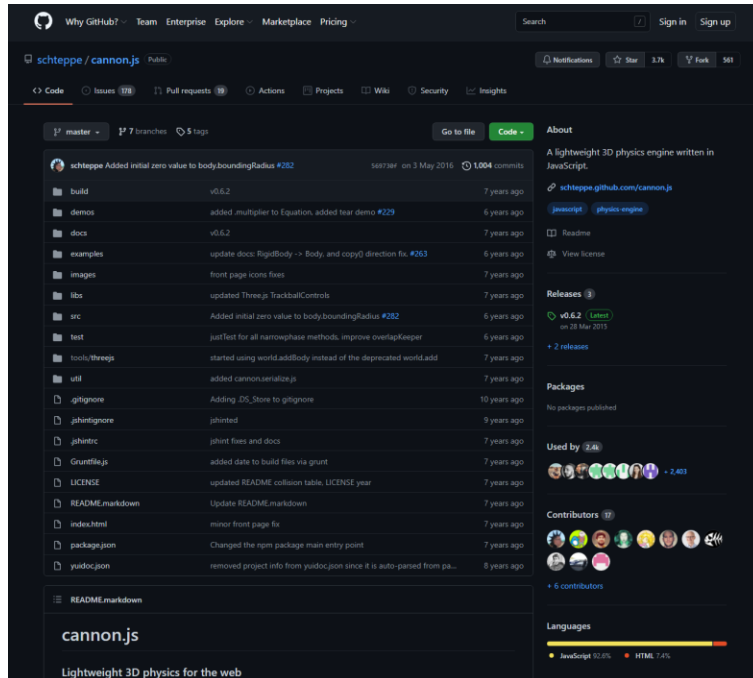
Desarrollo

Para realizar esta práctica utilizamos una biblioteca llamada Cannon.js y que esta creada para ser usada en los modelos 3D para hacer que los objetos que agregamos se muevan e interactúen de manera real.

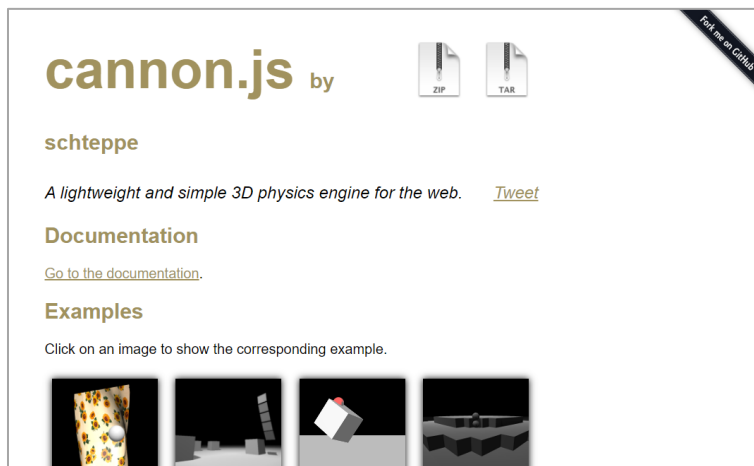
Agregando lo que nosotros conocemos como físicas haciendo una presentación mas realista de nuestros modelos 3D.

Creación del proyecto

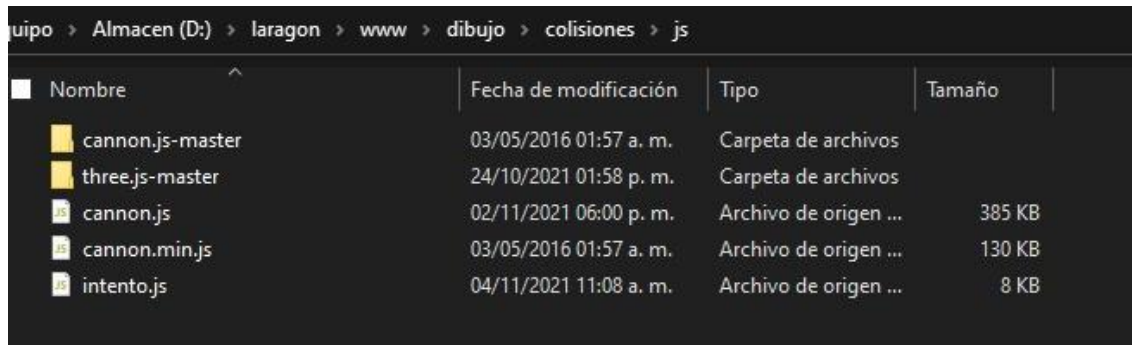
Antes de comenzar lo que hicimos fue descargar la biblioteca de Cannon.js desde la página de github en un archivo zip que después descomprimos para seguir agregando los archivos a nuestro proyecto.



Dándole al botón verde tendremos el master del proyecto que contiene los 3 posibles módulos diferentes que ofrece la página principal.

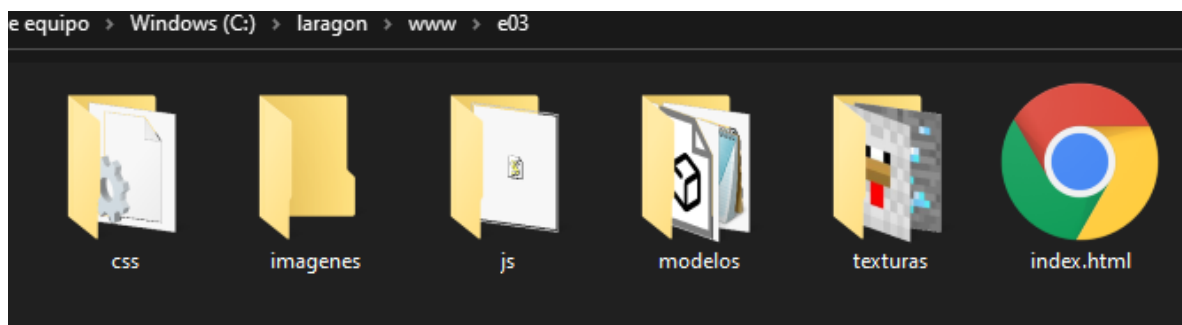


También está la página principal que te permite descargar un archivo comprimido, pero en realidad hay cierta diferencia en su contenido



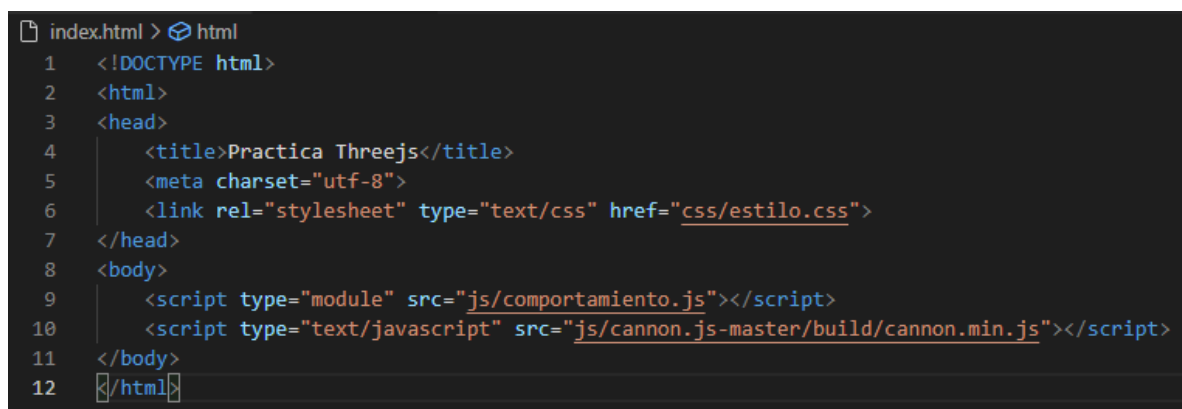
Nombre	Fecha de modificación	Tipo	Tamaño
cannon.js-master	03/05/2016 01:57 a. m.	Carpeta de archivos	
three.js-master	24/10/2021 01:58 p. m.	Carpeta de archivos	
cannon.js	02/11/2021 06:00 p. m.	Archivo de origen ...	385 KB
cannon.min.js	03/05/2016 01:57 a. m.	Archivo de origen ...	130 KB
intento.js	04/11/2021 11:08 a. m.	Archivo de origen ...	8 KB

Hay cierta contradicción entre distintas fuentes oficiales del proyecto y en este caso nos funcionó teniendo las librerías de esta manera.



Como ya sabemos necesitamos de nuestros correspondientes archivos antes de iniciar a hacer el código para evitar complicaciones a la hora de querer comprobar nuestros avances.

Con la intención de cargar texturas creamos la ya común carpeta en Laragon que debe estar iniciada para ejecutar el proyecto.



```
index.html > html
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Practica Threejs</title>
5   <meta charset="utf-8">
6   <link rel="stylesheet" type="text/css" href="css/estilo.css">
7 </head>
8 <body>
9   <script type="module" src="js/comportamiento.js"></script>
10  <script type="text/javascript" src="js/cannon.js-master/build/cannon.min.js"></script>
11 </body>
12 </html>
```

```
D:\laragon\www\dibujos\colisiones\juego.html - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

juego.html
1
2 <!DOCTYPE html>
3 <html>
4   <head>
5     <meta charset="utf-8">
6     <title>Intento de juego</title>
7     <style rel="stylesheet" type="text/css">
8       body{
9         margin: 0;
10      }
11    </style>
12  </head>
13  <body>
14    <script type="module" src="js/intento.js"></script>
15    <script type="text/javascript" src="js/cannon.js"></script>
16  </body>
17 </html>
```

Primeramente y algo muy importante es que debemos de agregar cannon.js o cannon.min.js en el HTML.

Browser install

Just include `cannon.js` or `cannon.min.js` in your html and you're done:

```
<script src="cannon.min.js"></script>
```

De nuevo basándonos en la explicación de GitHub. Tomando esto en cuenta, no hay necesidad de importarlo desde el código y sorprendentemente funciona

```
intento.js - js - Visual Studio Code
JS intento.js x
JS intento.js > Main > meterBloques > forma
1
2 import * as THREE from './three.js-master/build/three.module.js'
3 import { OrbitControls } from './three.js-master/examples/jsm/controls
4 //import * as CANNON from './cannon.js-master/build/cannon.js'
5
```

Ya se puede trabajar con la inicialización de las físicas.

```

class Main{
    constructor(){
        this.initAtributos()
        this.initMateriales()
        this.initFisica()
        this.initEscena()
        this.initCamara()
        this.initComponentes()
    }
}

```

```

initAtributos(){
    this.escena=null
    this.camarap=null
    this.camarao=null
    this.rbtctrls=null
    this.renderizador=null
    this.ayudasombra=null
    this.gmtrs={}
    this.mallas={}
    this.colores={}
    this.luces={}
    this.fongos={}
    this.fisicos=[]
    this.cuadros=[]
    this.terreno=new THREE.Group()
}

```

El proyecto no es tan diferente de los otros en este también agregaremos nuestros constructores, y atributos. Prácticamente siendo indispensables para el funcionamiento de nuestro código.

```

initEscena(){
    this.escena=new THREE.Scene()
}
initCamara(){
    this.camarap=new THREE.PerspectiveCamera(
        100, window.innerWidth/window.innerHeight, 0.1, 1000
    );
    this.renderizador=new THREE.WebGLRenderer()
    this.renderizador.setSize(window.innerWidth, window.innerHeight)
    this.renderizador.shadowMap.enabled=true
    document.body.appendChild(this.renderizador.domElement)
    this.rbtctrls=new OrbitControls(this.camarap, this.renderizador.domElement)
    this.rbtctrls.enableDamping=true
    this.rbtctrls.dampingFactor=0.05
    this.rbtctrls.minDistance=10
    this.rbtctrls.listenToKeyEvents( window )
}

```

```

initMateriales(){
  this.colores[0]=new THREE.MeshBasicMaterial({color:"white"})
  this.colores[1]=new THREE.MeshBasicMaterial({color:"black"})
  this.colores[2]=new THREE.MeshBasicMaterial({color:"green"})

  const textura2=new THREE.TextureLoader().load('texturas/suelo-natural.jpg')
  const textura3=new THREE.TextureLoader().load('texturas/gallina.jpg')
  const textura4=new THREE.TextureLoader().load('texturas/diamante.jpg')
  const textura5=new THREE.TextureLoader().load('texturas/pasto.jpg')

  this.fongos["tierra"]=new THREE.MeshPhongMaterial({map:textura2})
  this.fongos["gallina"]=new THREE.MeshPhongMaterial({map:textura3})
  this.fongos["diamante"]=new THREE.MeshPhongMaterial({map:textura4})
  this.fongos["pasto"]=new THREE.MeshPhongMaterial({map:textura5})
  this.luces[0]=new THREE.PointLight(this.colores[0],1,100)
  this.luces[1]=new THREE.PointLight(this.colores[0],1,100)
  this.luces[2]=new THREE.AmbientLight(this.colores[0],0.4)
  this.ayudasombra=new THREE.CameraHelper(this.luces[0].shadow.camera)
  this.gmtrs[0]=new THREE.PlaneGeometry(2,2,1,1)
  this.gmtrs[1]=new THREE.BoxGeometry(4,4,4)
  this.gmtrs[2]=new THREE.SphereGeometry(8,32,32)
  this.mallas["pelota"]=new THREE.Mesh(this.gmtrs[2],this.fongos["gallina"])
  this.mallas["cubo"]=new THREE.Mesh(this.gmtrs[1],this.fongos["diamante"])
}

```

Iniciamos nuestra cámara y escena incluyendo los controles de orbita, y por supuesto agregamos los materiales que utilizaremos.

```

crearTerreno(ancho, altura){
  this.escena.add(this.terreno)
  for(let i=0;i<ancho;i++){
    for(let j=0;j<altura;j++){
      let m=new THREE.Mesh(this.gmtrs[0],this.fongos["pasto"])
      m.receiveShadow=true
      this.terreno.add(m)
      let equis=(-ancho)+(i*2);
      let ye=(altura)-(j*2)
      m.position.set(equis,ye,0)
    }
  }
  console.log(this.terreno)
}

```

Creamos nuestro terreno para poder trabajar en él y ajustamos su posición.

Iluminación

Esta parte de la iluminación es importante ya que su principal objetivo es que la escena y todo lo que haya en ello se pueda ver claramente además de que será más fácil ver los movimientos generados cuando se agreguen las físicas a los objetos.

```
this.escena.add(this.luces[0])
this.escena.add(this.luces[1])
this.escena.add(this.luces[2])

this.luces[0].position.set(40,35,0)
this.luces[1].position.set(-40,35,0)
this.luces[0].castShadow=true
this.luces[1].castShadow=true
```

Los atributos de cast y receive shadow habilitan que cada malla reciba o genere sombras.

```
65 this.renderizador=new THREE.WebGLRenderer()
66 this.renderizador.setSize(window.innerWidth, window.innerHeight)
67 this.renderizador.shadowMap.enabled=true
68 document.body.appendChild(this.renderizador.domElement)
69 this.orbitctrls=new OrbitControls(this.camara, this.renderizador.domElement)
```

En el objeto renderizador le colocamos como verdadera a la habilidad de “grabar con la cámara” todas las sombras que generen los objetos.

```
47 this.fongos[0]=new THREE.MeshPhongMaterial({map:texture})
48 this.luces[0]=new THREE.PointLight(this.colores[0],1,100)
49 this.luces[1]=new THREE.PointLight(this.colores[0],1,100)
50 this.luces[2]=new THREE.AmbientLight(this.colores[0],0.4)
51 this.ayudasombra=new THREE.CameraHelper(this.luces[0].shadow.camera)
52 this.plano[0]=new THREE.PlaneGeometry(2,2,1,1)
```

Se debe ejecutar un ayudante de cámara donde solo necesitas poner el punto de vista de la sombra de una luz vista desde una cámara.


```

102     initComponents() {
103         this.meterBloques()
104         this.crearTerreno(50,50)
105         this.escena.add(this.luces[0])
106         this.escena.add(this.luces[1])
107         this.escena.add(this.luces[2])
108         //this.escena.add(this.mallas["cubo"])
109         this.escena.add(this.mallas["pelota"])
110         //this.terreno.rotateX(Math.PI*3/2)
111         //this.terreno.position.set(0,-5,0)
112         this.luces[0].position.set(40,35,0)
113         this.luces[1].position.set(-40,35,0)
114         this.luces[0].castShadow=true
115         this.luces[1].castShadow=true
116         //this.mallas["cubo"].castShadow=true
117         this.mallas["pelota"].castShadow=true
118         this.mallas["pelota"].receiveShadow=true
119         this.camarap.position.set(-40,10,70)
120     }

```

Cada objeto añadido a escena se le debe de determinar si produce sombra o si las recibe, usamos 2 luces laterales porque una sola no alcanzara todo lo que haremos y una ambiental para que los lugares con sombras no se vean completamente en negro.

Es algo perfecto para el proyecto que estamos haciendo, donde habrá físicas realistas como las del mundo al que estamos acostumbrados.

CANNON.js

El uso de Cannon.js es considerablemente difícil comparado contra el uso de Three.js.

```
⋮ README.markdown

// Setup our world
var world = new CANNON.World();
world.gravity.set(0, 0, -9.82); // m/s2

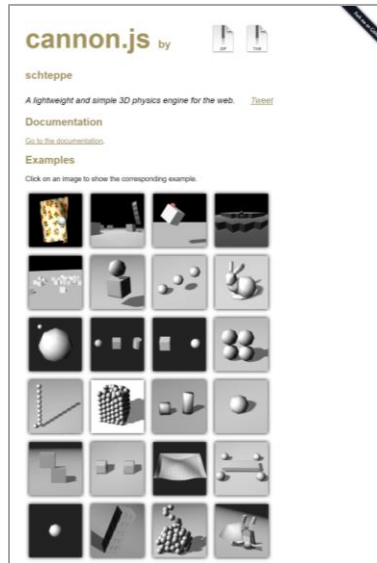
// Create a sphere
var radius = 1; // m
var sphereBody = new CANNON.Body({
  mass: 5, // kg
  position: new CANNON.Vec3(0, 0, 10), // m
  shape: new CANNON.Sphere(radius)
});
world.addBody(sphereBody);

// Create a plane
var groundBody = new CANNON.Body({
  mass: 0 // mass == 0 makes the body static
});
var groundShape = new CANNON.Plane();
groundBody.addShape(groundShape);
world.addBody(groundBody);

var fixedTimeStep = 1.0 / 60.0; // seconds
var maxSubSteps = 3;

// Start the simulation loop
var lastTime;
(function simloop(time){
  requestAnimationFrame(simloop);
  if(lastTime !== undefined){
    var dt = (time - lastTime) / 1000;
    world.step(fixedTimeStep, dt, maxSubSteps);
  }
  console.log("Sphere z position: " + sphereBody.position.z);
  lastTime = time;
})();
```

Tan solo hay dos ejemplos básicos de código en su página de Github de donde se descarga.



Y despues en su página, aunque parecen haber muchos utilizan un módulo llamado `CANNON.demo()`, para comprender como fueron programados podemos hacer trampa y presionar `Ctrl+U` para ver su código fuente.

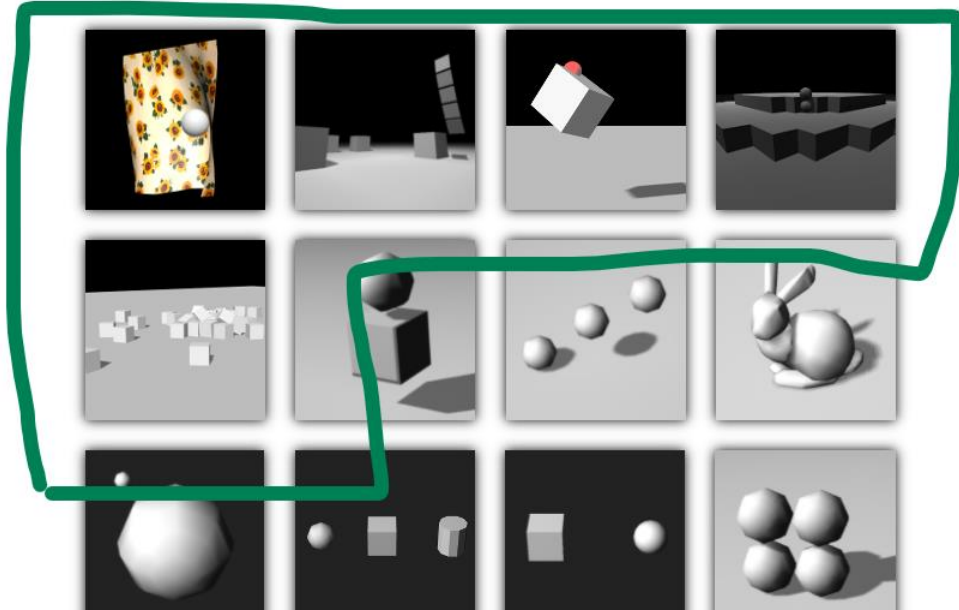
```
20 var demo = new CANNON.Demo();
21
22 var postStepHandler;
23
24 demo.addScene("Tween box",function(){
25     var world = demo.getWorld();
26
27     // Inputs
28     var startPosition = new CANNON.Vec3(5, 0, 2);
29     var endPosition = new CANNON.Vec3(-5, 0, 2);
30     var tweenTime = 3; // seconds
31
32     var body = new CANNON.Body({
33         mass: 0,
34         type: CANNON.Body.KINEMATIC,
35         position: startPosition
36     });
37     body.addShape(new CANNON.Box(new CANNON.Vec3(1,1,1)));
38     world.add(body);
39     demo.addVisual(body);
40 }
```

Este ofrece su propio tipo es escenario tridimensional, poner materiales cannon que son grises.

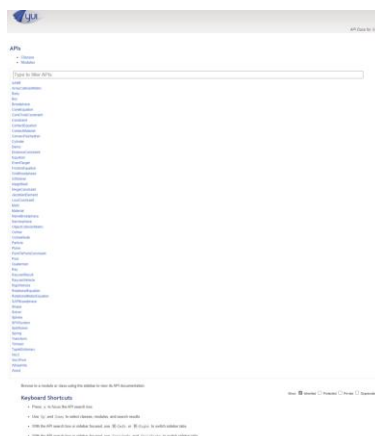
Ademas de que a un body solo se le necesita meter en una función `demo.addVisual(body)` y automáticamente con eso tienes un objeto realista.

Examples

Click on an image to show the corresponding example.



Pero esto no nos sirve porque queremos colocarles texturas a los componentes y en toda la pagina solo estos pocos utilizan realmente mallas de Three.js, por eso decimos que es difícil aprender a usar Cannon.js.



Por si fuera poco, la documentación no es nada estética y es difícil encontrar lo que buscas. Sin mencionar que casi no hay tutoriales en la web de como implementarlo.

```
initFisica(){
  this.mundo = new CANNON.World
  this.mundo.gravity.set(0,-70,0)
  this.mundo.broadPhase= new CANNON.NaiveBroadphase()
```

Ahora sí procedemos a crear el mundo usando el CANNON.World. Luego, configuramos la gravedad que estará afectando a los objetos que pongamos en la escena.

Luego se agrega el “broadPhase” al mundo para que pueda encontrar el objeto en colisión.

```
this.crpplano=new CANNON.Body({
  mass:0,
  position:new CANNON.Vec3(1,0,0),
});
```

```
this.crpplano.quaternion.setFromAxisAngle(new CANNON.Vec3(1,0,0),Math.PI*27/16)
this.crpplano.addShape(new CANNON.Plane())
```

Ahora que tenemos nuestro mundo físico, podemos comenzar a agregar los objetos.

Primero crearemos nuestro plano. Estableciendo la masa en cero, ya que si ponemos otro valor el plano se verá afectado por la gravedad, poniendo el valor en cero asegurará que el plano se quede quieto en su lugar.

```
this.crpbola=new CANNON.Body({
  mass:400,
  position:new CANNON.Vec3(20,35,-30),
  shape:new CANNON.Sphere(8)
});
```

```
this.mundo.addBody(this.crpplano)
this.mundo.addBody(this.crpbola)
```

Creamos una forma de esfera y le establecemos el valor de la masa por encima de 0 para que esta si pueda ser afectada por la gravedad.

```

let forma=new CANNON.Box(new CANNON.Vec3(2,2,2))
for(let i=0;i<90;i++){
  let equis=(Math.random()*60)-30;
  let seta=(Math.random()*60)-30;
  let ye=(Math.random()*20)+10;
  let m=new THREE.Mesh(this.gmtrs[1],this.fongos["diamante"])
  let crpbox=new CANNON.Body({
    mass:Math.random()*10,
  });
  crpbox.addShape(forma)
  m.castShadow=true
  m.receiveShadow=true
  crpbox.position.set(equis,ye,seta)
  m.position.set(equis,ye,seta)
  this.mundo.addBody(crpbox)
  this.escena.add(m)
  this.fisicos.push(crpbox)
  this.cuadros.push(m)
}

```

Para no agregar los objetos de uno en uno escribimos un ciclo for que nos ayudará a agregar una cantidad de objetos a nuestro plano que tendrán un valor aleatorio en su masa.

```

animando(){
  requestAnimationFrame(()=>{this.animando()})
  this.mallas["pelota"].position.copy(this.crpbola.position)
  this.mallas["pelota"].quaternion.copy(this.crpbola.quaternion)
  this.terreno.position.copy(this.crpplano.position)
  this.terreno.quaternion.copy(this.crpplano.quaternion)
  console.log("p "+this.crpplano.position)
  for(let c=0;c<this.fisicos.length;c++){
    this.cuadros[c].position.copy(this.fisicos[c].position)
    this.cuadros[c].quaternion.copy(this.fisicos[c].quaternion)
  }
  this.renderizador.render(this.escena,this.camara)
  this.rbtctrls.update()
  this.mundo.step(1.0/60.0)
}

```

Todo esto lleva una animación así que agregamos lo debido al método de animación. Sin olvidar que al mundo de canon se le asignan los fotogramas por segundo en este método.

Three.js

One of the most convenient ways of using Cannon.js with [Three.js](#) is by enabling use of quaternions:

```
mesh.useQuaternion = true;
```

Note: On the latest version of Three.js (r59) useQuaternion is true by default.

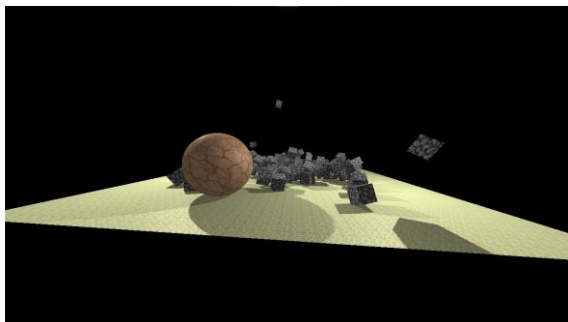
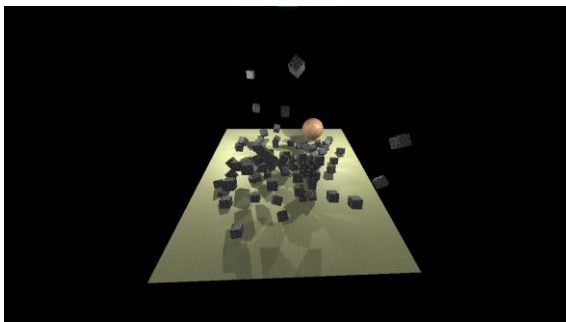
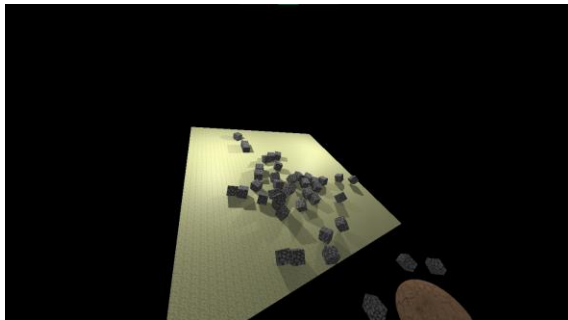
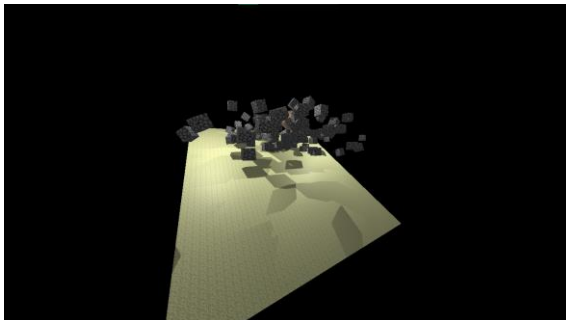
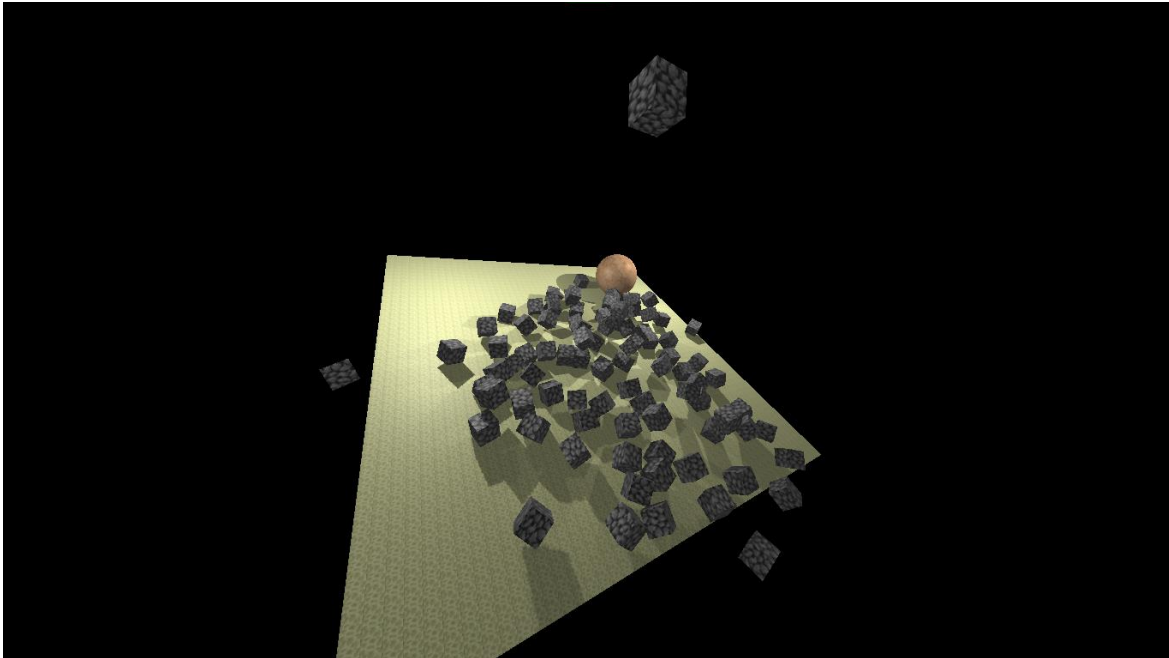
Then it gets really simple to copy over position+orientation data to the Three.js mesh:

```

mesh.position.x = body.position.x;
mesh.position.y = body.position.y;
mesh.position.z = body.position.z;
mesh.quaternion.x = body.quaternion.x;
mesh.quaternion.y = body.quaternion.y;
mesh.quaternion.z = body.quaternion.z;
mesh.quaternion.w = body.quaternion.w;

```

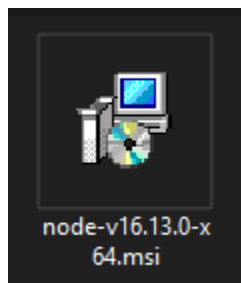
Primero se hizo el intento de esta manera, pero funcionaba de manera extraña.



Lo hice inclinado porque asi hay más movimiento.

Casos de estudio

Los módulos de Cannon.js necesitan Node.js según el propio creador de Github y se comprueba porque si no lo tienes no funciona ninguna de sus clases, así que primero lo buscamos para instalarlo.



Tras haber ejecutado el instalador anterior podrás abrir una consola o terminal para correr el siguiente comando.

```
Node.js install

Install the cannon package via NPM:

npm install --save cannon

Alternatively, point to the Github repo directly to get the very latest version:

npm install --save schteppe/cannon.js
```

En una lectura rápida nos enteramos que NPM es una especie de gestor de paquetes que necesita Cannon.js (entre otros) para estar trabajando completamente.


```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\zelmo> npm install --save cannon

added 1 package, and audited 2 packages in 3s

found 0 vulnerabilities
npm notice
npm notice New patch version of npm available! 8.1.0 -> 8.1.2
npm notice Changelog: https://github.com/npm/cli/releases/tag/v8.1.2
npm notice Run npm install -g npm@8.1.2 to update!
npm notice
PS C:\Users\zelmo> npm -v
8.1.0
PS C:\Users\zelmo>
```

```
Administrador: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\WINDOWS\system32> npm install --save cannon
npm ERR! ECONNRESET
npm ERR! ECONNRESET
npm ERR! request to https://registry.npmjs.org/cannon failed, reason: socket hang up
npm ERR! This is a problem related to network connectivity.
npm ERR! In most cases you are behind a proxy or have bad network settings.
npm ERR!
npm ERR! If you are behind a proxy, please make sure that the
npm ERR! 'proxy' config is set properly. See: 'npm help config'
npm ERR! A complete log of this run can be found in:
npm ERR! C:\Users\Blanca ramirez cruz\AppData\Local\npm-cache\_logs\2021-11-05T02_09_37_521Z-debug.log
PS C:\WINDOWS\system32> npm install --save cannon

added 1 package, and audited 2 packages in 3m

found 0 vulnerabilities
PS C:\WINDOWS\system32> npm -v
8.1.0
PS C:\WINDOWS\system32>
```

En este caso usamos Windows Powershell como consola, pero tambien funciona correctamente en CMD.

```
120
121 meterBloques(){
122     let forma=new CANNON.Box(new CANNON.Vec3(2,2,2))
123     for(let i=0;i<90;i++){
```

```
        this.crpbola=new CANNON.Body({
            mass:400,
            position:new CANNON.Vec3(20,30,20),
            shape:new CANNON.Sphere(8)
        });
```

```
52 this.gmtrs[0]=new THREE.PlaneGeometry(2,2,1,1)
53 this.gmtrs[1]=new THREE.BoxGeometry(4,4,4)
54 this.gmtrs[2]=new THREE.SphereGeometry(8,32,32)
55 this.mallas["pelota"]=new THREE.Mesh(this.gmtrs[2],this.for
```

Algo confuso es que en los cubos la geometría de Three.js debe ser el doble del tamaño de la geometría de Cannon.js, si las pones iguales los cubos no se tocan entre si visualmente.

Pero en la esfera no aplica, si colocas el doble de radio a la geometría de Three.js veras que esta pasa las paredes, aunque la razon podría ser que en la esfera de cannon en vez del radio asignamos el diámetro... no se puede saber con certeza.

```
var dt = 1/60;
function animate() {
  requestAnimationFrame( animate );
  if(controls.enabled){
    world.step(dt);

    // Update ball positions
    for(var i=0; i<balls.length; i++){
      ballMeshes[i].position.copy(balls[i].position);
      ballMeshes[i].quaternion.copy(balls[i].quaternion);
    }

    // Update box positions
    for(var i=0; i<voxels.bboxes.length; i++){
      boxMeshes[i].position.copy(voxels.bboxes[i].position);
      boxMeshes[i].quaternion.copy(voxels.bboxes[i].quaternion);
    }
  }

  controls.update( Date.now() - time );
  renderer.render( scene, camera );
  time = Date.now();
}
```

```
// Add boxes
var halfExtents = new CANNON.Vec3(1,1,1);
var boxShape = new CANNON.Box(halfExtents);
var boxGeometry = new THREE.BoxGeometry(halfExtents.x*2,halfExtents.y*2,halfExtents.z*2);
for(var i=0; i<7; i++){
  var x = (Math.random()-0.5)*20;
  var y = 1 + (Math.random()-0.5)*1;
  var z = (Math.random()-0.5)*20;
  var boxBody = new CANNON.Body({ mass: 5 });
  boxBody.addShape(boxShape);
  var boxMesh = new THREE.Mesh( boxGeometry, material );
  world.add(boxBody);
  scene.add(boxMesh);
  boxBody.position.set(x,y,z);
  boxMesh.position.set(x,y,z);
  boxMesh.castShadow = true;
  boxMesh.receiveShadow = true;
  boxes.push(boxBody);
  boxMeshes.push(boxMesh);
}
```

```
var dt = 1/60;
function animate() {
  requestAnimationFrame( animate );
  if(controls.enabled){
    world.step(dt);

    // Update ball positions
    for(var i=0; i<balls.length; i++){
      ballMeshes[i].position.copy(balls[i].position);
      ballMeshes[i].quaternion.copy(balls[i].quaternion);
    }

    // Update box positions
    for(var i=0; i<boxes.length; i++){
      boxMeshes[i].position.copy(boxes[i].position);
      boxMeshes[i].quaternion.copy(boxes[i].quaternion);
    }
  }

  controls.update( Date.now() - time );
  renderer.render( scene, camera );
  time = Date.now();
}
```

Los métodos de actualización de posición y el modo de acceso a los datos físicos fueron sacados de los ejemplos de la pagina de Cannon.js y sin estos es prácticamente imposible hacer una correcta implementación de esta biblioteca añadiendo texturas con Three.js.

Conclusión

A pesar de las complicaciones que se nos presentaron en el proceso para realizar nuestra práctica logramos resolver el problema y continuar haciendo el modelo.

Hay varias herramientas que nos ayudan a implementar lo que son las físicas en un modelo 3D pero en esta ocasión utilizamos CANNON.js ya que contaba con los materiales necesarios para poder entender mejor y aunque la forma en la que se implementa la biblioteca de CANNON.js no es tan fácil como parece una vez que se hace esta implementación puedes hacer cosas geniales con ella, pasando de ser un simple modelo aburrido a la vista de quien lo ve, a algo totalmente impresionante y atractivo con lo que hasta se puede interactuar de manera lista.

Sin duda resulta ser muy útil la herramienta de CANNON.js para mejorar notablemente un modelo 3D y gracias a esta práctica nos dimos cuenta de eso.