Introducción a los motores de física

Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Sistemas Gráficos

Grado en Ingeniería Informática

Curso 2021-2022

Contenidos

Objetivos

- Conocer lo que ofrece un motor de física
- Saber qué magnitudes físicas se pueden representar
- Saber definir materiales y objetos físicos
- Saber interactuar con ellos
- Saber procesar colisiones
- Saber crear y configurar restricciones
- Saber diseñar e implementar escenas sencillas

Introducción

- Un motor de física permite:
  - Dotar de gravedad a la escena
  - Masa a los objetos
  - Atributos a los materiales como:
    - ★ Rozamiento
    - ★ Efecto rebote
  - Detectar y procesar colisiones entre objetos
- Requiere bastante cálculo
  - Se suele separar la Física y el Rendering en hebras distintas
- Usaremos la biblioteca **Physijs**
  - Descargable de `github.com/chandlerprall/Physijs`
  - Requiere usar la versión de Three.js que viene con Physijs

Algunos cambios en la estructura de la aplicación

- La biblioteca no usa módulos
  - Requiere cambios en el archivo html
  - Se incluyen en él todos los fuentes
  - No se realizan imports (ni exports)
  - Consultar los ejemplos proporcionados

Archivo html: (fragmento)

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo Básico de Física</title>
    <meta charset="utf-8">
<!-- En esta ocasión no se usan módulos -->
    <script type="text/javascript" src="./physijs/jquery.js"></script>
    <script type="text/javascript" src="./physijs/three.min.js"></script>
    <script type="text/javascript" src="./physijs/physi.js"></script>
    <script type="text/javascript" src="./physijs/dat.gui.js"></script>
    <script type="text/javascript" src="./physijs/TrackballControls.js"></script>
    <script type="text/javascript" src="./physijs/keycode.js"></script>
    <script type="text/javascript" src="MyPhysiScene.js"></script>

    . . .
```

Una escena básica con Physijs Three.js

Physijs: Una escena básica (1/2)

```
// La clase deriva de la escena física
class MyPhysiScene extends Physijs.Scene {

  constructor (myCanvas) {

    // El gestor de hebras
    Physijs.scripts.worker = './physijs/physijs_worker.js'
    // El motor de física de bajo nivel, en el cual se apoya Physijs
    Physijs.scripts.ammo   = './ammo.js'

    // Las dos líneas anteriores DEBEN ejecutarse antes de inicializar Physijs.Scene.
    //   En este caso, antes de llamar a super
    super();

    // Se crea el visualizador,
    //   pasándole el lienzo sobre el que realizar los renderizados. Esto no cambia.
    this.renderer = this.createRenderer(myCanvas);

    // Se establece el valor de la gravedad, negativo en la Y, los objetos caen hacia abajo
    this.setGravity (new THREE.Vector3 (0, -10, 0));

    // Se construye una figura física

    . . .
```

Una escena básica con Physijs Three.js

Physijs: Una escena básica (2/2)

```
    . . .

    // Se construye una figura física

    // Se crea un material físico en base a un material Three
    var mat = new THREE.MeshPhongMaterial ({color: 0xff000000});
    var matFisico = Physijs.createMaterial (mat, 0.9, 0.3);

    // Se crea una geometría Three
    var geom = new THREE.BoxGeometry (1, 3, 2);

    // Se crea un mesh físico
    this.figuraFisica = new Physijs.BoxMesh (geom, matFisico, 25);

    // IMPORTANTE: Los elementos que se desee sean tenidos en cuenta en la FISICA
    //   deben colgar DIRECTAMENTE de la escena. NO deben colgar de otros nodos.

    this.add (this.figuraFisica);
  }

  update() {
    // Entre otras cosas

    this.simulate();
  }
}
```

Propiedades de los objetos

- Rozamiento y rebote
  - Con valores entre 0.0 y 1.0
  - Se indican al definir un material físico

```
var mat = Physijs.createMaterial (
  new THREE.MeshPhongMaterial ({color: 0xff0000}),
  0.9,  // rozamiento
  0.3); // rebote
```

- Masa
  - Se indica al crear el Mesh físico

```
var suelo = new Physijs.BoxMesh (
  new THREE.BoxGeometry (60,1,60), mat,
  0); // masa
```

  - El valor 0 hace que no le afecte la gravedad, necesario en aquellos objetos como el suelo, paredes que no se caen, etc.

A tener en cuenta

- Importante:
  Las figuras físicas **deben colgar de la raíz de la escena**
- Los valores para la masa, rozamiento y rebote deben ponerse con sentido
  - Los cuerpos ligeros son más *dinámicos* que los pesados

Formas disponibles

- Se debe usar la forma Physijs que mejor se adapte a la geometría creada
- Las formas más usuales de Physijs son:
  - Physijs.BoxMesh
  - Physijs.SphereMesh
  - Physijs.CylinderMesh
  - Physijs.ConeMesh
  - Physijs.ConvexMesh : Para aquellas geometrías que no encajen bien en las otras formas
    - ★ Es más lenta. Debe evitarse en la medida de lo posible.

Objetos compuestos

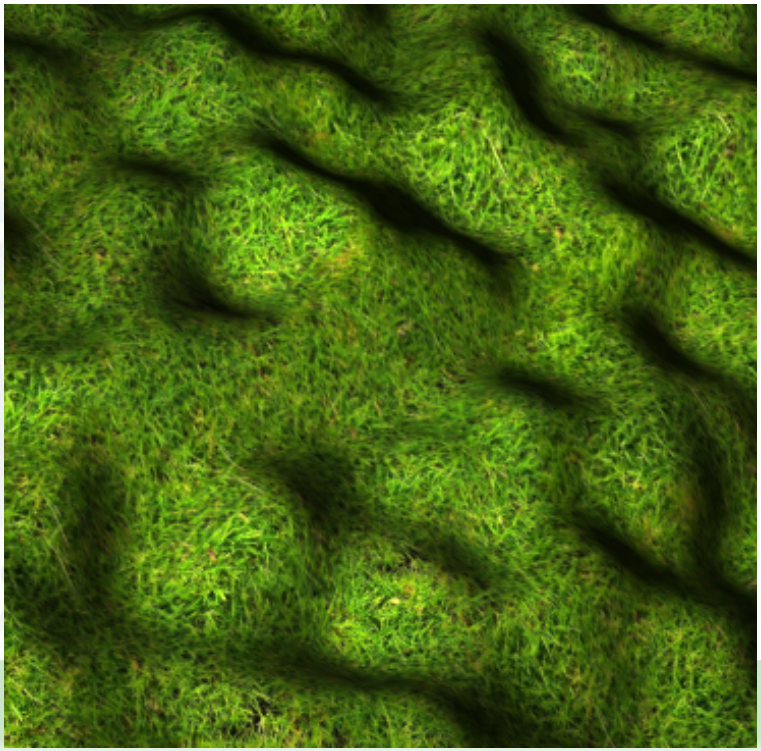
- Para que un objeto compuesto por varios elementos sea tratado como un objeto único por Physijs
  - Debe haber una relación jerárquica entre los elementos
  - La jerarquía debe estar hecha **antes** de añadir la raiz de la jerarquía a la raíz de la escena

Physijs: Objetos compuestos tratados como un todo

```
var suelo = new Physijs.BoxMesh (
    new THREE.BoxGeometry (60,1,60), materialSuelo, 0);

var paredIzq = new Physijs.BoxMesh (
    new THREE.BoxGeometry (2,6,60), materialParedes, 0);
paredIzq.position.x = -30;
paredIzq.position.y = 2.5;

suelo.add (paredIzq);    primero se monta la jerarquía

scene.add (suelo);       luego se cuelga en la raíz de la escena
```

Formas especiales

- Physijs.HeightfieldMesh
  - Crea un plano con alturas
  - Sirve para crear terrenos ondulados



Physijs: Plano con alturas

```
var sueloGeometria = new THREE.PlaneGeometry (60, 50, 100, 100);
for (var i = 0; i < sueloGeometria.vertices.length; i++) {
  sueloGeometria.vertices[i].z = // se le da la altura deseada
}
sueloGeometria.computeFaceNormals();      // Necesario
sueloGeometria.computeVertexNormals();    //   al cambiar las Z
var suelo = new Physijs.HeightfieldMesh (sueloGeometria,
    sueloMaterial, 0,    // masa
    100, 100);
```

Modificaciones manuales

- La posición, orientación y velocidad de un objeto vienen determinadas por:
  - El efecto de la gravedad
  - Impulsos (empujones)
  - La interacción con otros objetos
    - Colisiones, rozamientos, rebotes, etc.
- Si se desea modificar la posición y/u orientación manualmente
  - Se modifican los atributos `position` y/o `rotation`
  - Se le indica al motor de física para que lo tenga en cuenta
    - Atributo `__dirtyPosition = true`
    - Atributo `__dirtyRotation = true`
- Si se desea modificar la velocidad manualmente
  - `figura.setLinearVelocity (velocidad);`
  - `figura.setAngularVelocity (velocidad);`
    - El parámetro `velocidad` es un `THREE.Vector3`

Empujar objetos

- A las figuras se les puede dar un *empujón* con el método `applyCentralImpulse`

Physijs: Impulsos

```
// La fuerza que se desea aplicar
var fuerza = 10;

// La dirección
var offset = new THREE.Vector3 (1,2,3);

// Hay que descomponer la fuerza en un vector 3D
var effect = offset.normalize().multiplyScalar(fuerza);

// Se aplica el impulso
objetoFisico.applyCentralImpulse (effect);

// Si se quiere aplicar en la dirección opuesta solo hay que negar el vector
objetoFisico.applyCentralImpulse (effect.negate());
```

Colisiones

- El motor detecta y procesa las colisiones
  - Rebotes, cambios de dirección, etc.
- Se puede programar una función
  - Emitir un sonido, quitar una vida, etc.
- Para que esa función sea llamada, la figura física debe colgar **directamente** de la escena

Physijs: Listener de colisiones

```
elMesh.addEventListener ('collision',
    (elOtroObjeto, velocidad, rotacion, normal) => {
      // el procesamiento a realizar
    });
```

Restricciones a los movimientos

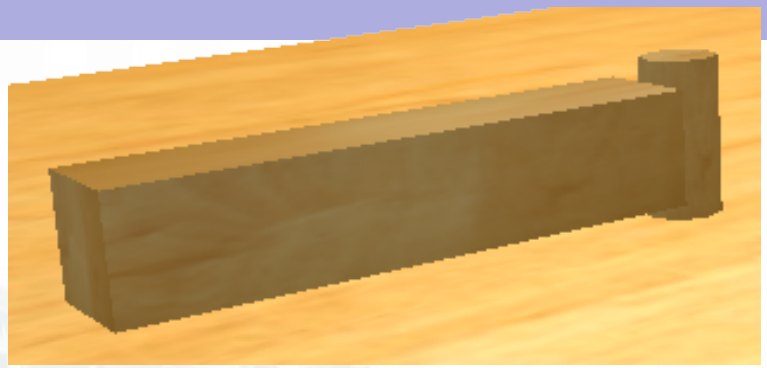
- Se pueden añadir restricciones a los movimientos que tiene un objeto por la gravedad y las colisiones
- PointConstraint
  - Se fija la posición de un objeto respecto a otro
  - Si uno se mueve, el otro también lo hará
  - Manteniendo la distancia y la orientación

Physijs: Restricción objeto a objeto

IMPORTANTE

```
// Los objetos obj1 y obj2 deben estar en la escena ANTES
//   de definir la restricción
var restric = new Physijs.PointConstraint (
      obj1, obj2, obj2.position);

// Las restricciones también deben añadirse a la escena
// Pero con otro método
scene.addConstraint (restric);
```

Restricción tipo bisagra



- HingeConstraint

Physijs: Restricción tipo bisagra

```
var restric = new Physijs.HingeConstraint (
      objMovil, objFijo, objFijo.position,
      new THREE.Vector3 (0,1,0));    // el eje de la bisagra
scene.addConstraint (restric);
// PRIMERO: Se añade la restricción a la escena, LUEGO se configura
// Configuración de la restricción:
// - Angulo mínimo y máximo
// - Con qué fuerza intenta evitar salirse de límites < 0.5
// - Rebote al llegar a límites
retric.setLimits (-Math.PI/2, Math.PI/2, 0.1, 0.1);

// Para moverlo intencionadamente
// Velocidad positiva mueve en un sentido, negativa en el opuesto
restric.enableAngularMotor (velocidadMaxima, aceleracion);

// Para desactivar el motor
//    (solo se movería por gravedad o colisiones)
restric.disableMotor();
```

Restricción de deslizamiento

- SliderConstraint

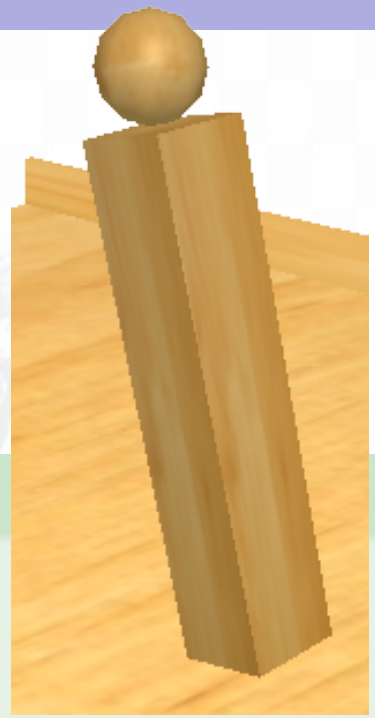
Physijs: Restricción tipo deslizamiento

```
var restric = new Physijs.SliderConstraint (
      objMovil, objFijo, objMovil.position,
      new THREE.Vector3 (0,1,0));    // el eje del deslizamiento (1)
scene.addConstraint (restric);
// Límites al movimiento, distancia mínima y máxima
retric.setLimits (-10, 10);
// Para moverlo intencionadamente
restric.enableLinearMotor (velocidad, aceleracion);
// Para desactivar el motor
//    (solo se movería por gravedad o colisiones)
restric.disableMotor();
```

- (1) - Ojo
  - Eje X, poner (0, 1, 0)
  - Eje Y, poner (0, 0, Math.PI/2)
  - Eje Z, poner (Math.PI/2, 0, 0)

    no me preguntéis por qué

Restricción de péndulo



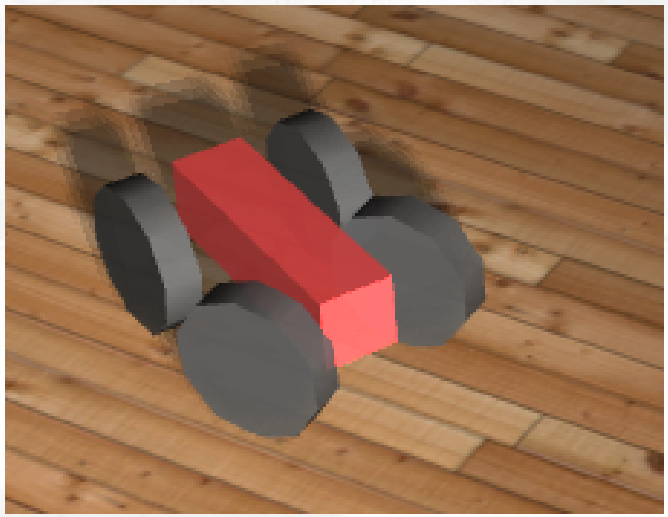
- ConeTwistConstraint

Physijs: Restricción de péndulo

```
var restric = new Physijs.ConeTwistConstraint (
      objMovil, objFijo, objFijo.position);
scene.addConstraint (restric);

// Límites al movimiento, 3 ángulos para los 3 ejes
retric.setLimits (0.5*Math.PI, 0, 0.5*Math.PI);

// Para moverlo intencionadamente unos determinados ángulos por eje
restric.enableMotor ();
restric.setMotorTarget (new THREE.Vector3 (1, 0, 1.5));

// Para desactivar el motor
//    (solo se movería por gravedad o colisiones)
restric.disableMotor();
```

Restricción de grado de libertad

- Permite controlar de manera exacta los movimientos angulares y lineales de un objeto
  - Definiendo límites
  - Modificando los límites cuando sea necesario
  - Moviendo elementos bajo demanda activando algún motor
  - Desactivando algún motor para que deje de actuar
- Veamos esta restricción definiendo un coche



Ejemplo del coche

Construcción del coche

Coche: Su construcción

```
// El coche se construye de la manera habitual,
//   pero usando figuras físicas

var carroceria = new Physijs.BoxMesh ( . . . );
scene.add (carroceria);

// Rueda Front Right
var ruedaFR = new Physijs.CylinderMesh ( . . . );

// Se gira porque el cilindro sale 'tumbado'
ruedaFR.rotation.x = Math.PI/2;

// Se posiciona correctamente con respecto a la carrocería
ruedaFR.position.set ( . . . );

// Las ruedas se añaden a la escena, no a la carrocería
scene.add (ruedaFR);

// Y así con las otras 3 ruedas
```

Ejemplo del coche

Restricciones en las ruedas

Coche: Restricciones en las ruedas: Su giro natural

```
// Restricción para la Rueda Front Right
var restriccionFR = new Physijs.DOFConstraint (
  ruedaFR, carroceria, ruedaFR.position);

// La rueda se 'pega' a la carrocería en la posición indicada
// y serán inseparables

// Se añade la restricción a la escena
scene.addConstraint (restriccionFR);

// Se definen sus límites para los movimientos libres
restriccionFR.setAngularLowerLimit({ x: 0, y: 0, z: 0.1 });
restriccionFR.setAngularUpperLimit({ x: 0, y: 0, z: 0 });

// Solo se permite girar por el eje z,
// y como el límite inferior es mayor que el límite superior,
// se permite el giro completamente libre

// Igual para las otras 3 ruedas
```

Ejemplo del coche

Marcha adelante y atrás

Coche: Marcha adelante y atrás

```
// Es un coche de propulsión trasera y por tanto
// se actúa solo en las restricciones de las ruedas traseras
// Se hace ante una petición del usuario (ha pulsado una tecla)

// Se configura un motor angular
// en la Restricción de la Rueda Rear Right y se activa
restriccionRR.configureAngularMotor (2, 0.1, 0, velocidad, fuerza)

// Primer parámetro: Es el eje de este motor, (0 = x, 1 = y, 2 = z)
// 2º y 3er parámetros: Límite inferior > superior (giro completo)
// Si  velocidad > 0, hacia adelante; si < 0,  hacia atrás
// Si  fuerza  es grande, el coche puede hacer 'caballitos'

// El motor hay que activarlo indicando el eje
restriccionRR.enableAngularMotor(2)

// Igual para la Restricción de la Rueda Rear Left
```

Ejemplo del coche

Frenar

Coche: Frenar

```
// Como en el mundo real:
// 1º Se deja de pisar el acelerador
//    El coche sigue avanzando por la inercia
// 2º Se pisa el freno para frenar

// 1º Se desactivan los motores que hubiera activos
//    En todas las ruedas que tengan el motor activo
restriccionRR.disableAngularMotor(2);

// Las ruedas siguen girando por la inercia

// 2º Si se desean frenar se pone a cero su velocidad angular
ruedaRR.setAngularVelocity (new THREE.Vector3(0,0,0));

// En todas las ruedas
```

Ejemplo del coche

Giros de volante

Coche: Giros de volante

```
// Se modifican los límites en las restricciones
//   de las ruedas directrices, en el eje vertical, el Y

restriccionFR.setAngularLowerLimit ({x:0, y:angulo, z:0.1});
restriccionFR.setAngularUpperLimit ({x:0, y:angulo, z:0});

// El ángulo se da en radianes

// Si  angulo > 0  gira a la izquierda; si < 0, a la derecha

// Se sigue permitiendo el giro libre en el eje Z

// Lo mismo en la Restricción de la Rueda Front Left
```

Introducción a los motores de física

Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Sistemas Gráficos

Grado en Ingeniería Informática

Curso 2021-2022