Práctica 1:
Ejemplo Grúa

Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Sistemas Gráficos

Grado en Ingeniería Informática

Curso 2017-2018

Contenidos

1.- Introducción

Se presenta un ejemplo de desarrollo de un Sistema Gráfico mediante el modelo de una grúa de obra que puede manejarse a través de los controles de una interfaz gráfica de usuario.

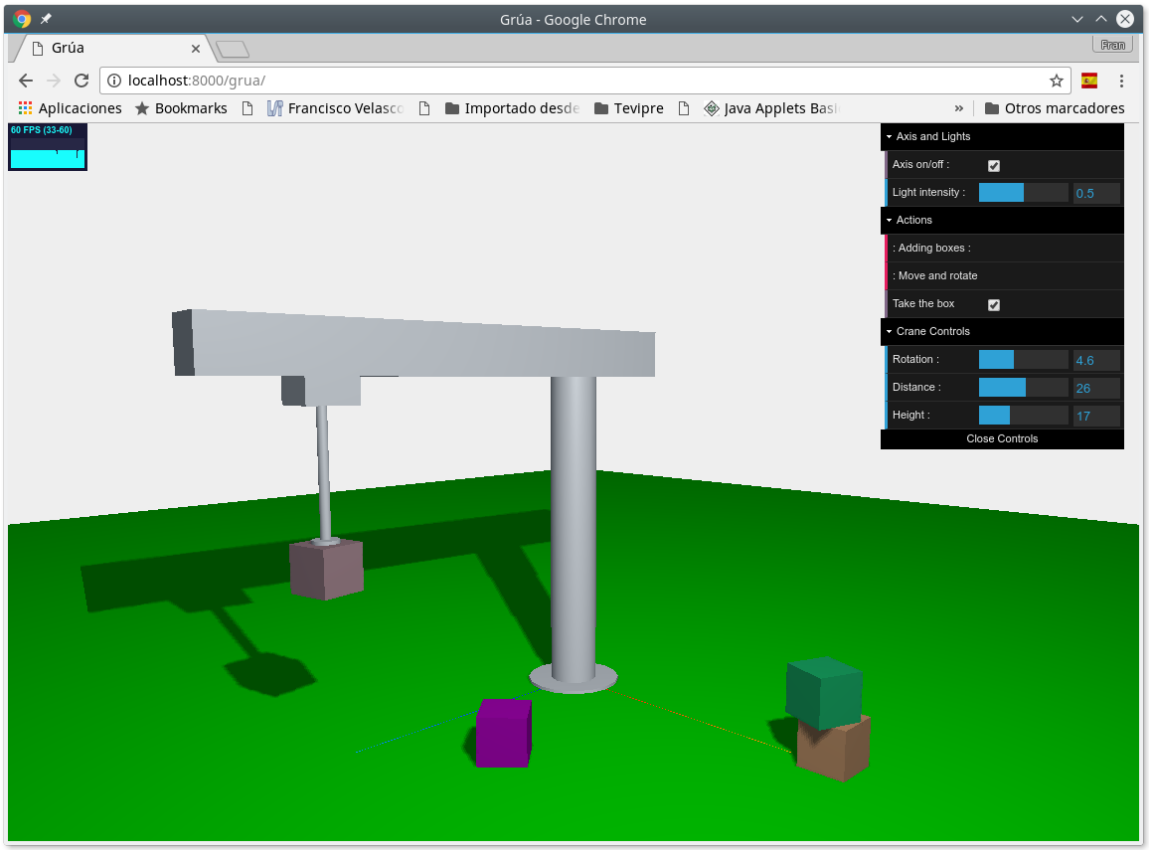
El sistema permite añadir cajas en tiempo de ejecución. La grúa puede enganchar cajas y transportarlas para depositarlas en otro sitio.

El usuario también puede, mediante el uso del ratón, seleccionar y desplazar cajas, apilar unas sobre otras y hacerlas girar sobre su propio eje vertical y central.

No hay que preocuparse si quedan dudas sobre las partes más complejas del ejemplo. Se trata de una toma de contacto con las prácticas y con un sistema gráfico. Los diferentes aspectos que se incluyen en el ejemplo serán tratados en profundidad a lo largo de la asignatura.

La siguiente imagen muestra una captura de la aplicación. En la parte superior derecha se tienen los controles de la interfaz gráfica de usuario, realizada con `dat.gui.js`, que permite:

- Mostrar, o no, los ejes de coordenadas del sistema de referencia universal.
- Controlar la intensidad de la luz focal que hay en la escena.
- Entrar en el modo “Añadir cajas”
- Entrar en el modo “Mover y rotar cajas”
- Enganchar y desenganchar cajas
- Mover la grúa:
  - Rotando el brazo con respecto al mástil
  - Desplazando la pluma a lo largo del brazo
  - Subiendo y bajando el gancho

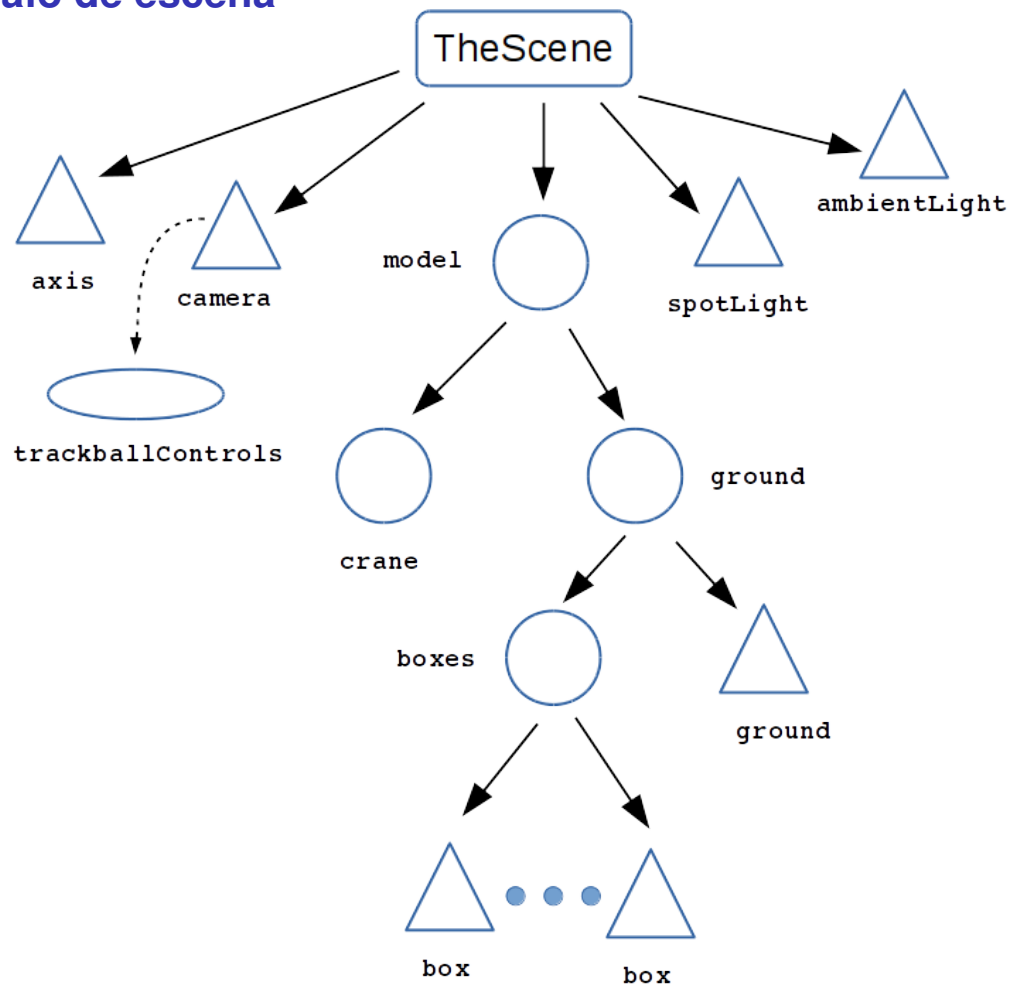


2.- Diseño

La siguiente imagen muestra el grafo de escena del sistema. Se muestran, mediante nodos, los distintos elementos que intervienen en la escena, desde las luces, la cámara, el suelo, la grúa, las cajas, etc. Los arcos de este grafo dirigido muestran las relaciones entre dichos elementos. Por ejemplo, la escena debe contenerlo todo. Lo que se denomina *modelo* contiene la grúa y el suelo, el nodo suelo contiene tanto a la primitiva geométrica que representa el suelo (un plano o una caja estrecha) como a un grupo de cajas.

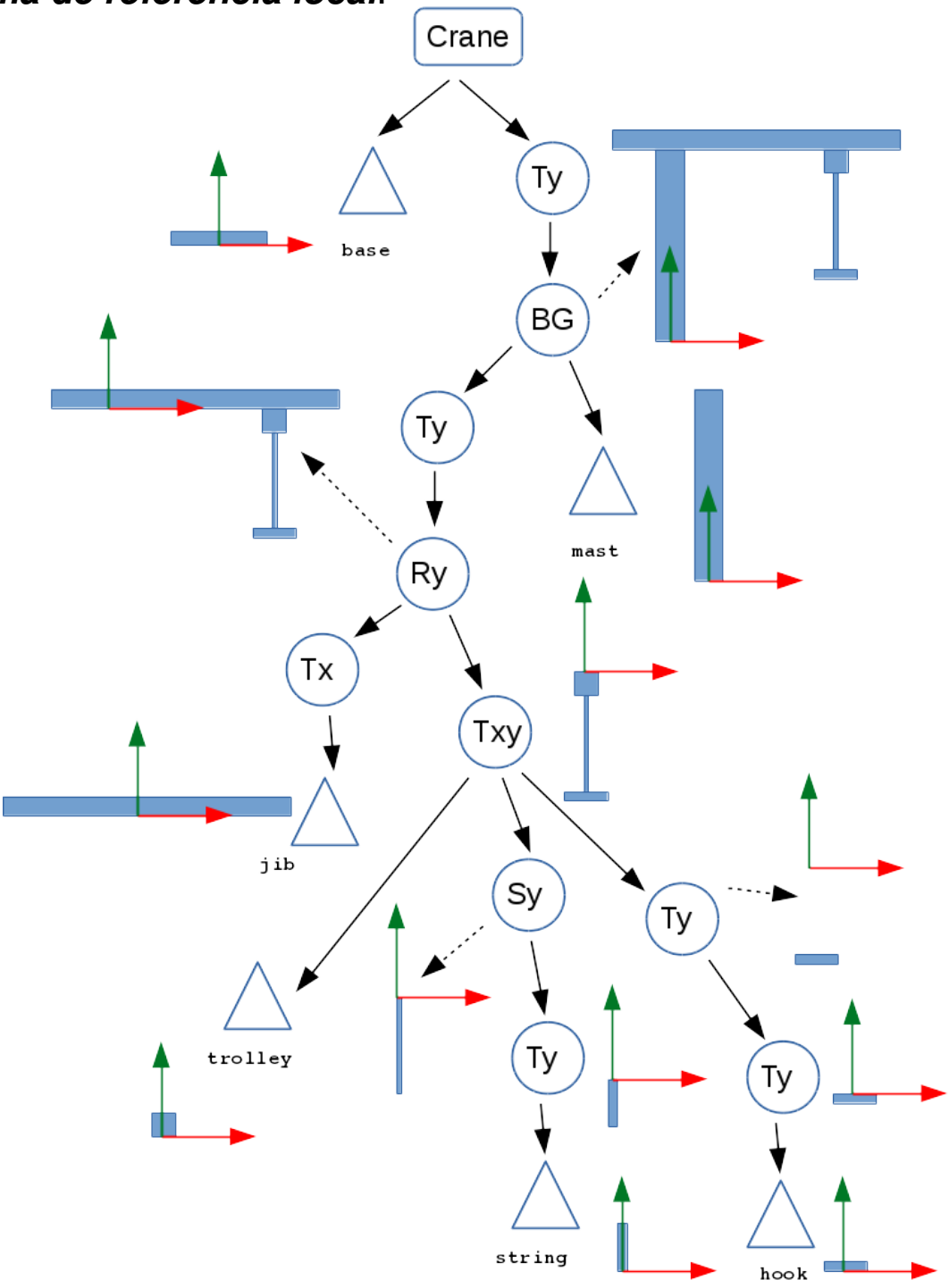
El grafo que representa la grúa se muestra en otra imagen.

Grafo de escena



Grafo de la grúa

Al hacer un grafo de escena **es fundamental**, para no equivocarse, **incluir al lado de cada nodo importante un dibujo de la parte del modelo que está representando, *incluyendo su sistema de referencia local***.

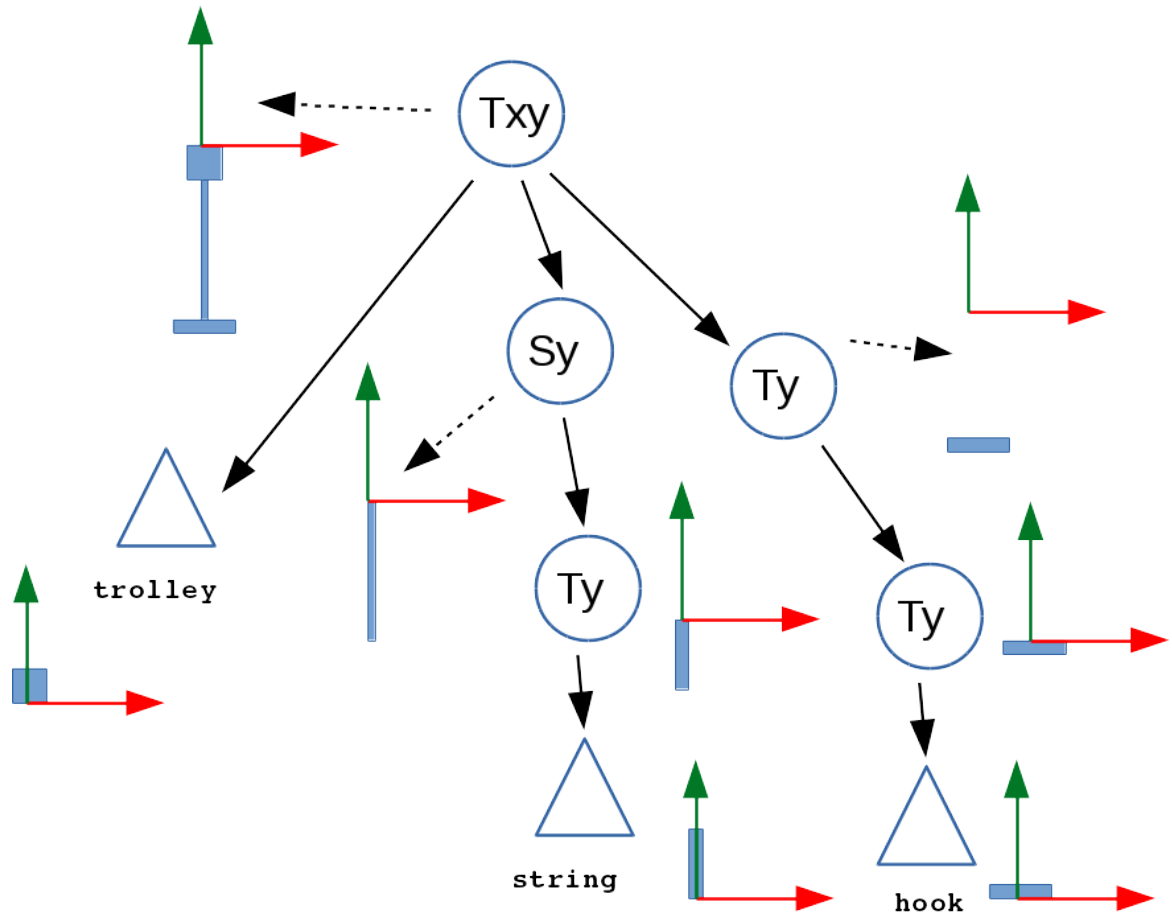


Detalle del conjunto Pluma-Cuerda-Gancho

Se observa cómo la traslación de la pluma (Txy) afecta tanto a la pluma como a la cuerda y al gancho.

Cada transformación que afecte a varios elementos se debe situar en el grafo de manera que su nodo apareza una sola vez e influya sobre todos los elementos a los que deba afectar. **Un nodo de transformación afecta a todos sus descendientes, tanto directos como indirectos**.

La longitud variable de la cuerda se ha representado con un escalado. Previamente, el cilindro que representa la cuerda se ha trasladado para que se sitúe en el origen de coordenadas el punto que permanece invariante al alargar y acortar la cuerda, la parte superior de la misma.

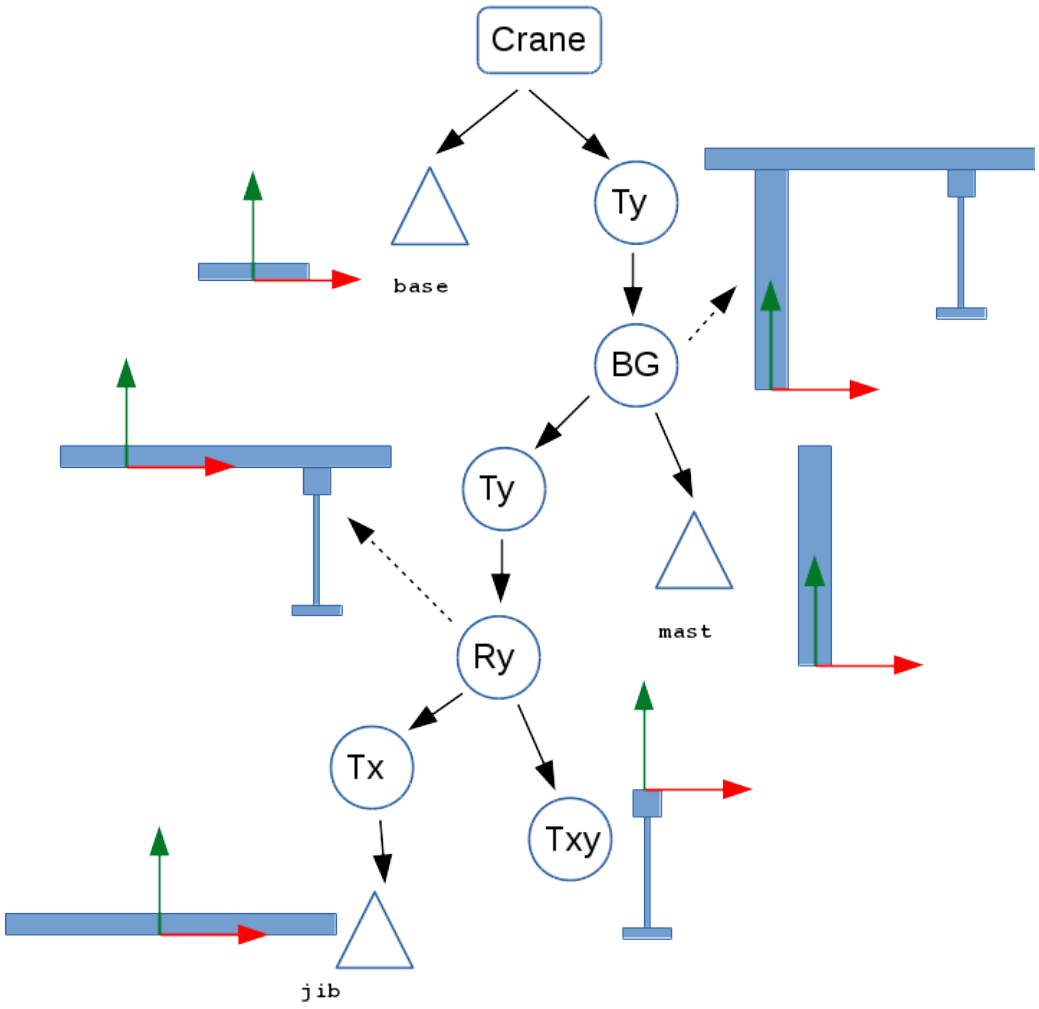


Detalle de la parte superior del grafo

Es una buena estrategia **realizar el diseño en 2 fases**:

En una primera fase, **descendente**, se va *dividiendo el objeto a modelar en partes cada vez menos complejas*. Si el objeto es articulado, un buen punto de división suele ser las articulaciones. Quedando a un lado la parte que permanece fija con respecto a esa articulación y al otro lado la parte que se ve afectada por el movimiento de esa articulación.

En una segunda fase, **ascendente**, se comienza por abajo, realizando los grafos de los componentes más sencillos, *componiendo los subgrafos en grafos cada vez más complejos* mientras se asciende hasta obtener el grafo del objeto completo.



Particularidades de Three.js

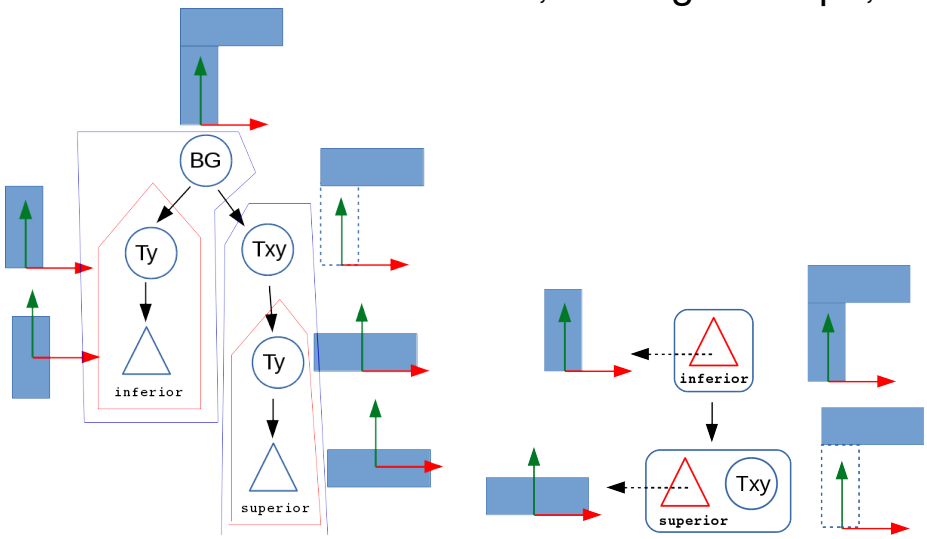
En `Three.js` los nodos de geometría pueden tener hijos y 2 tipos de transformaciones:

- Una transformación que no afecta a los hijos.
  Se utiliza para recolocar la geometría con respecto a su sistema de referencia local.
- Una transformación que afecta a la geometría y a sus descendientes.
  Se utiliza para modelado (colocar esta geometría en un modelo mayor) y animación.

Nodos de geometría: Particularidades

```
geometria = new THREE.Mesh (new THREE.BoxGeometry (. . .), material);
// Transformación que no afecta a los hijos de la geometría
geometria.geometry.applyMatrix (
    new THREE.Matrix4().makeTranslation (. . .));
// Transformaciones de modelado y animación, que sí afecta a sus descendientes
geometria.position.y = 5;
// Hijos de una geometría
geometria.add (otraGeometria);
```

Esa característica permite diseñar grafos más compactos. En la imagen de abajo, las transformaciones de recolocación de la geometría con respecto al sistema de referencia, las del primer tipo, son las Ty; mientras que la transformación de modelado para el elemento horizontal encima del vertical, del segundo tipo, es la Txy.



Grafo de la grúa adaptado a las particularidades de Three.js

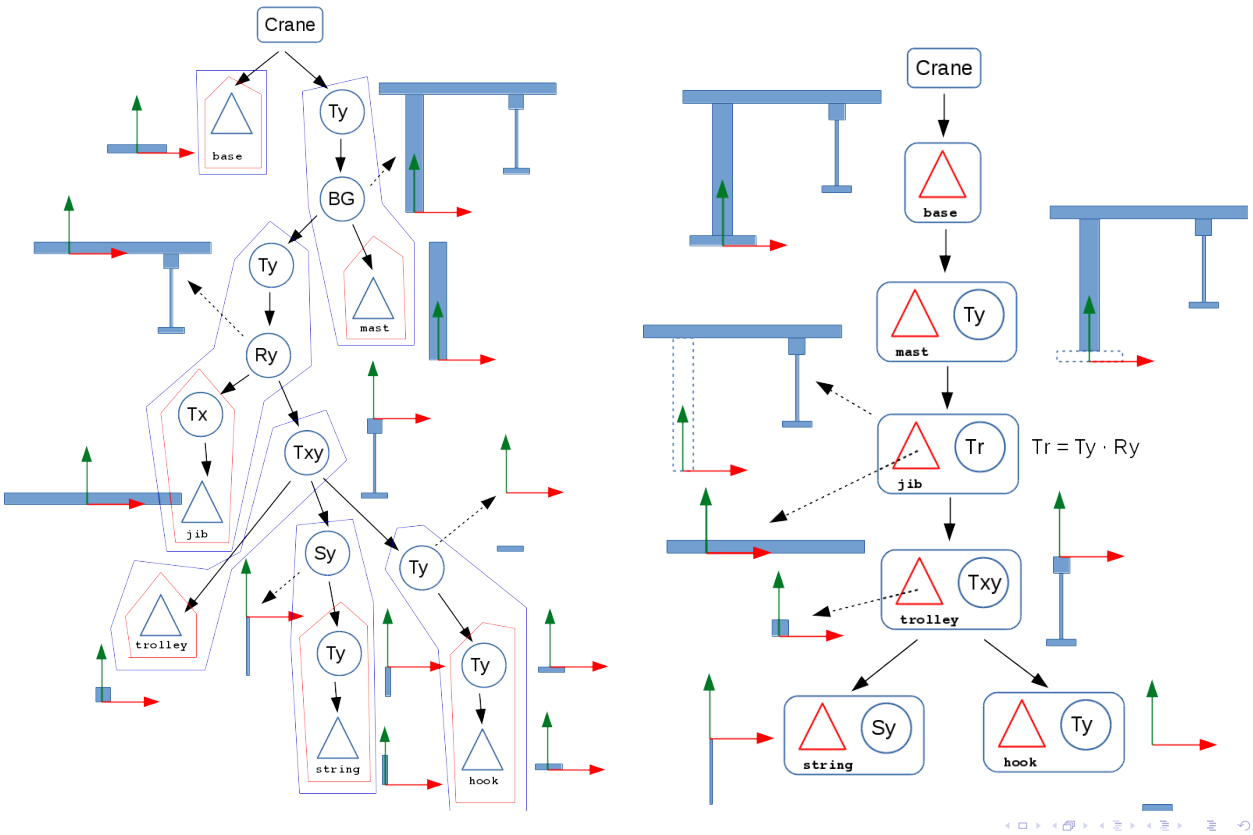
Teniendo en cuenta lo dicho anteriormente se queda un grafo para la grúa más compacto, incluyendo las mismas transformaciones.

No hay que preocuparse si han quedado dudas con respecto al grafo de la grúa. El diseño de grafos de escena y de objetos articulados será tratado con profundidad en la asignatura.

Sin embargo, **sí es importante que se asimilen 2 puntos**:

1. **Diseño en 2 fases**: una descendente de descomposición del objeto y otra ascendente de composición de subgrafos.
2. **Es fundamental acompañar a los nodos de dibujos, con sistema de referencia local incluido**, que muestren la parte del objeto que están representando.

No hacerlo así no va a ahorrar tiempo, al contrario, resultará un diseño con errores que llevará mucho más tiempo corregir una vez se haya implementado.

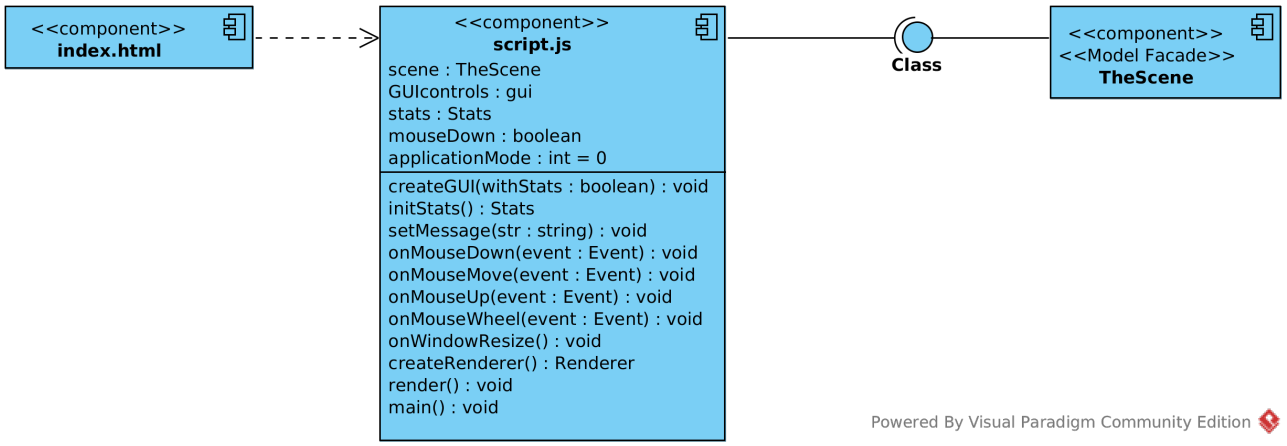


Estructura de la aplicación

La aplicación:

- Tendrá interfaz gráfica mediante `dat.gui.js`
- Permitirá la interacción directa con el ratón sobre la escena
- Tendrá un control de cámara predefinido
- Tendrá dos luces:
  - Una ambiental
  - Una focal de intensidad variable. Genera sombras.
- Se basará en una clase fachada, `TheScene`

Diagrama de componentes

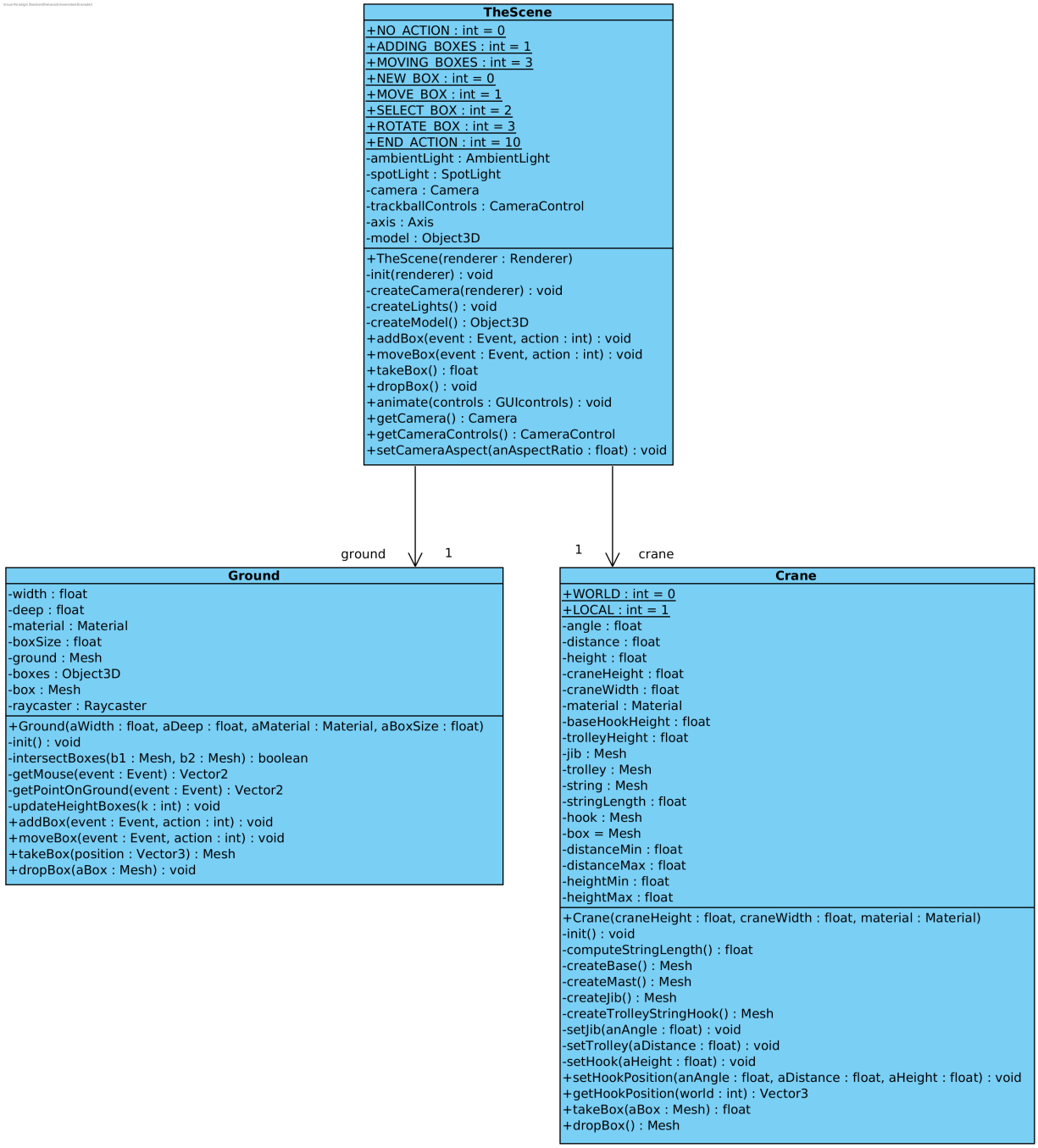


Se trata de una aplicación para la web. El archivo html correspondiente incluye un script javascript que contiene el “main” que es el que crea la escena y realiza el renderizado de cada frame.

El script javascript también se encarga de asignar funciones a los distintos eventos del ratón que se desean capturar y procesar.

Diagrama de clases

El diagrama de clases es sencillo, una clase para representar a la escena y que tiene la responsabilidad de ser la clase fachada y que se relaciona tanto con la clase de la grúa como con la clase de suelo, la cual incluye la colección de cajas como atributo.



3.- Implementación

Archivo `index.html`

Básicamente se trata de un archivo que incluye los javascript de las bibliotecas utilizadas y las de las clases de la aplicación.

También se define el área que después se usará para mostrar el render. En este caso se le ha dado el identificador `WebGL-output`

Archivo: `index.html` **(fragmento)**

```
<!DOCTYPE html>
<html>
<head>
    <title>Grúa</title>
    <meta charset="utf-8">
    <script type="text/javascript" src="../libs/three.js"></script>
    <!-- inclusión de otras bibliotecas ... -->
    . . .
    <!-- ... de nuestras clases y del script con el "main" -->
    <script type="text/javascript" src="Crane.js"></script>
    <script type="text/javascript" src="TheScene.js"></script>
    <script type="text/javascript" src="script.js"></script>
</head>
<body>
  <!-- Div which will hold the Output -->
  <div id="WebGL-output">
  </div>
. . .
```

Archivo `script.js`

Incluye la función principal, la cual crea el renderer WebGL, enlaza su salida con el área que se ha definido en el html para tal fin. Se crea la escena, la interfaz gráfica de usuario y se llama la primera vez a la función encargada de hacer el render.

`script.js`: Función ‘‘ main’’ (fragmento)

```
$(function () {
  renderer = createRenderer();
  $("#WebGL-output").append(renderer.domElement);
  scene = new TheScene (renderer.domElement);
  createGUI(true);
  render();
});
```

Este archivo también incluye la función que crea el renderer, que no es más que una instancia del renderer WebGL de THREE. El cual se configura poniéndole un color de fondo, un tamaño, que se toma del tamaño de la ventana en el navegador, y se habilita la gestión de sombras arrojadas.

script.js: Creación del renderer

```
function createRenderer () {
  var renderer = new THREE.WebGLRenderer();
  renderer.setClearColor(new THREE.Color(0xEEEEEE), 1.0);
  renderer.setSize(window.innerWidth, window.innerHeight);
  renderer.shadowMap.enabled = true;
  return renderer;
}
```

Por su parte, la función que realiza el render encola una nueva llamada a sí misma, si no, solo se haría un único render y se desea que esta función se esté ejecutando continuamente.

Los elementos que hayan podido cambiar desde el render del frame anterior deben actualizarse. En este caso, se actualiza la posición de la cámara y la propia escena.

Finalmente se le solicita al renderer que realice la visualización de la escena para una cámara concreta.

script.js: Función render

```
function render() {
  requestAnimationFrame(render);
  scene.getCameraControls().update ();
  scene.animate(controls);
  renderer.render(scene, scene.getCamera());
}
```

Este archivo también incluye la implementación de la interfaz gráfica de usuario (GUI).

Interfaz Gráfica de Usuario con `dat.gui.js`

Se define una función-objeto con un atributo por cada parámetro a controlar, el tipo de dato determina el tipo del control en la GUI

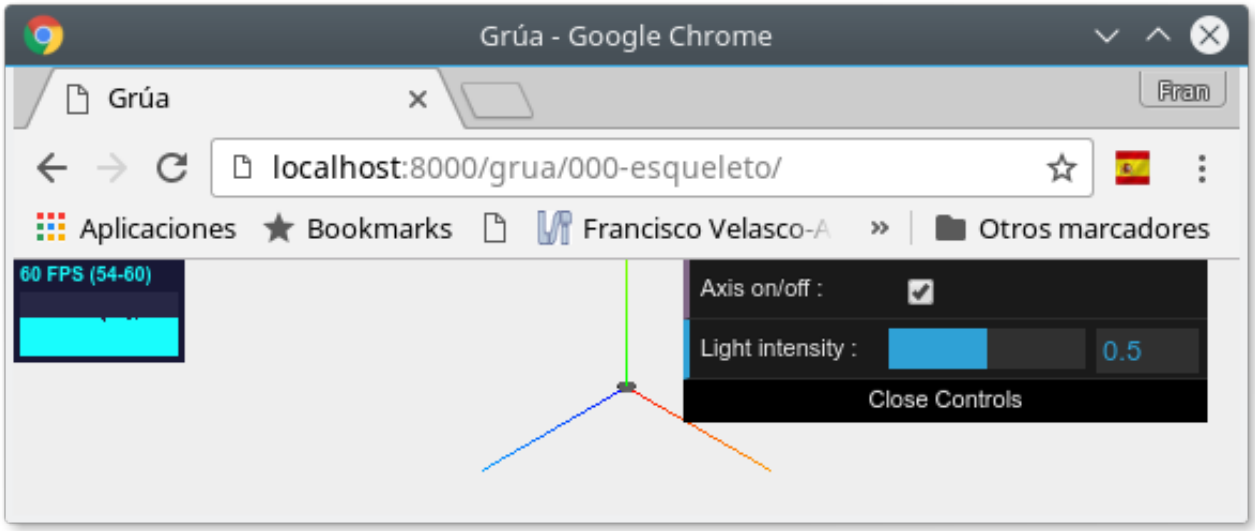
- Booleano, para un control tipo *checkbox*
- Número, para un control tipo *slider*
- Función, para un control de tipo *button*
  La propia función es el código que se ejecutará al pulsar el botón

Posteriormente se conforma la interfaz, agrupando opciones, poniendo etiquetas, límites, etc. Cuando sea necesario, se pueden consultar los atributos de dicha función-objeto para actuar sobre los elementos que deban verse afectados por dichos atributos.

Más información en `https://github.com/dataarts/dat.gui`

Ejemplo de GUI

La imagen siguiente muestra un ejemplo de GUI para mostrar u ocultar los ejes del sistema de referencia universal y para controlar la intensidad de la luz.



Para implementar dicha interfaz se define un objeto función con esos dos atributos, uno booleano y otro de tipo real.

Se define la interfaz a partir de una instancia de la clase `dat.GUI` a la que se le añaden los controles concretos a usar relancionándolos con los atributos del objeto función.

Finalmente, en la función `animate` que se llamará para cada render, se actualizan los elementos de la escena en según de los atributos el objeto función.

GUI preliminar: Mostrar/Ocultar ejes y control de la luz

```
controls = new function() {
  this.axis = true;
  this.lightIntensity = 0.5;
}

var gui = new dat.GUI();
gui.add(controls, 'axis').name('Axis on/off :');
gui.add(controls, 'lightIntensity', 0, 1.0).name('Light intensity:');

this.animate = function (controls) {
  axis.visible = controls.axis;
  spotLight.intensity = controls.lightIntensity;
}
```

Clase `TheScene`

Esta clase es la clase fachada del modelo. Es la que recibe, en primera instancia, los mensajes que se generan con la interacción del usuario, encargándose de *transmitir órdenes* al resto de las clases del modelo.

Igualmente, posee la cámara de la escena y el control de la misma, así como las luces de la escena.

Sus atributos y métodos, por tanto, se responsabilizan de esta funcionalidad.

Creación de la cámara y su control

Para crear una cámara se le indica el ángulo de visión, su ratio de aspecto, que se le da teniendo en cuenta el ratio de aspecto de la ventana del navegador, y sus planos de recorte cercano y lejano.

También se indica dónde se sitúa la cámara y hacia dónde mira.

Cámara: Creación

```
this.camera = new THREE.PerspectiveCamera (45,
    window.innerWidth / window.innerHeight, 0.1, 1000);

this.camera.position.set (60, 30, 60);

var look = new THREE.Vector3 (0,20,0);
this.camera.lookAt(look);
```

Para controlar el movimiento de la cámara se usa una instancia de una clase de THREE dedicada a tal fin. Con dicho controlador se hace zoom moviendo la rueda del ratón, se hacen giros arrastrando el botón izquierdo, y se hacen reencuadres arrastrando el botón derecho.

En esta aplicación en concreto, solo funciona el movimiento de cámara si se realiza mientras se tiene pulsado `Ctrl`. Dado que se va a poder manejar la aplicación con el ratón, no se desea que cualquier arrastre del ratón, por ejemplo para mover una caja, provoque también un giro de cámara.

Esta característica de discriminar una funcionalidad u otra del ratón en función de la pulsación de una tecla se implementará en las funciones que procesen los eventos del ratón. Se verá más adelante en este documento.

Cámara: Control

```
this.trackballControls = new THREE.TrackballControls (this.camera, renderer);
this.trackballControls.rotateSpeed = 5;
this.trackballControls.zoomSpeed = -2;
this.trackballControls.panSpeed = 0.5;
this.trackballControls.target = look;
```

Iluminación

Se usa una luz focal que proyecta sombras arrojadas más una luz ambiental para todos los objetos tengan algo de iluminación aunque no estén directamente iluminados por la luz focal.

Para la gestión de las sombras, además de habilitarlas en el renderer, hay que habilitarlas en las fuentes de luz e indicar qué objetos las generan y cuáles las reciben.

Luces: Definición

```
this.ambientLight = new THREE.AmbientLight(0xccddee, 0.35);

this.spotLight = new THREE.SpotLight( 0xffffff );
this.spotLight.position.set( 60, 60, 40 );
this.spotLight.castShadow = true;
```

Añadido de elementos a la escena

Todos los elementos, ya sean luces, cámara, geometría, etc., para que formen parte de la escena, deben ser añadidos a la misma, en este caso el objeto de la clase TheScene.

De esa forma son tenidos en cuenta al hacer el rendering de cada frame. Lo que no se haya añadido a la escena es como si no existiera.

Se añaden mediante el método `add`.

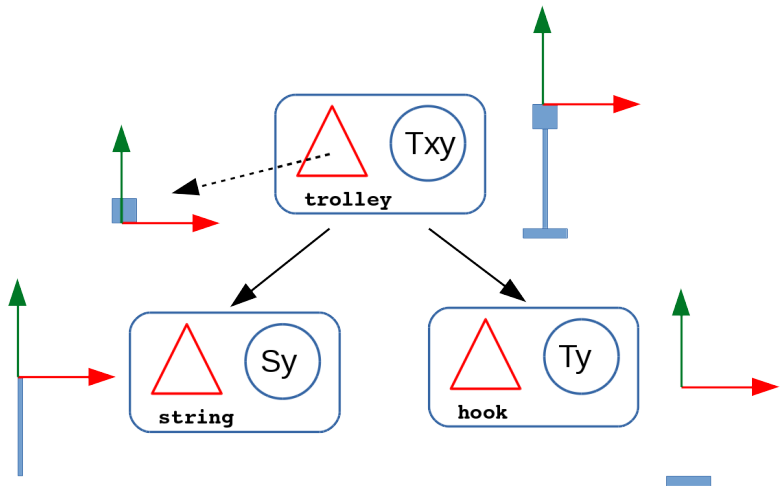
TheScene: Añadido de elementos a la escena

```
// En la clase TheScene this referencia al objeto de la escena

// Observar cómo la creación de cada elemento concluye con su
//    añadido a   this

this.add (this.ambientLight);
this.add (this.spotLight);
this.add (this.camera);
this.add (this.model);
```

Implementación de la Grua

Una vez hecho el diseño del grafo, lo difícil, queda lo fácil, traducir a código lo especificado en el gráfo. *Observar qué objetos, atributos y métodos intervienen en el proceso*. Los dibujos de cada nodo ayudan a visualizar lo que se está implementando.

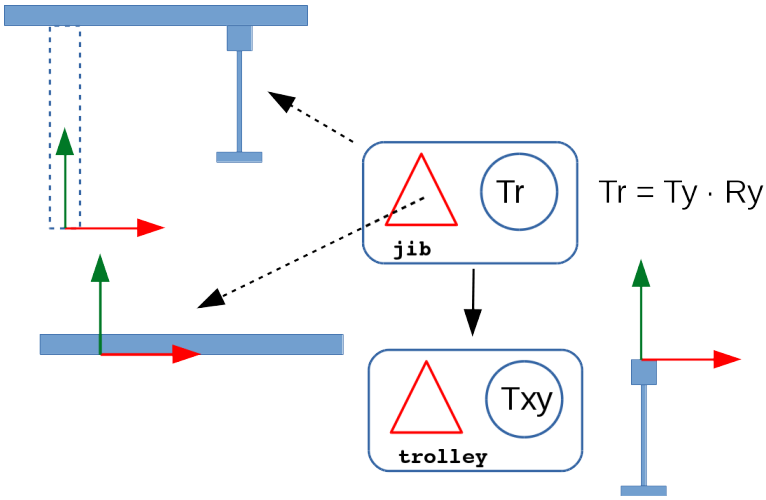
Conjunto Pluma-Cuerda-Gancho



Grúa: Conjunto Pluma-Cuerda-Gancho

```
createTrolleyStringHook () {
  // Pluma
  // Las primitivas en THREE se crean centradas en el origen
  this.trolley = new THREE.Mesh (new THREE.BoxGeometry (. . .), this.material);
  // Se transforma para situar el sistema de coordenadas donde interesa
  this.trolley.geometry.applyMatrix (
      new THREE.Matrix4().makeTranslation (0, this.trolleyHeight/2, 0));
  this.trolley.castShadow = true;    // Genera sombras
  this.trolley.position.y = -this.trolleyHeight;    // Baja la pluma y sus hijos
  this.trolley.position.x = this.distanceMin;   // Desplaza la pluma y sus hijos

  // Cuerda
  this.string = new THREE.Mesh (new THREE.CylinderGeometry (. . .), this.material);
  this.string.geometry.applyMatrix (
    new THREE.Matrix4().makeTranslation (0, -0.5, 0));
  this.string.castShadow = true;
  this.stringLength = this.computeStringLength();
  this.string.scale.y = this.stringLength;    // Longitud de la cuerda: escalado
  this.trolley.add (this.string);    // La cuerda es hija de la pluma

  // Gancho
  this.hook = new THREE.Mesh ( new THREE.CylinderGeometry (. . .), this.material);
  this.hook.geometry.applyMatrix (
    new THREE.Matrix4().makeTranslation (0, -this.baseHookHeight/2, 0));
  this.hook.castShadow = true;
  this.hook.position.y = -this.stringLength;    // Posición del gancho: traslación
  this.trolley.add (this.hook);    // El gancho es hijo de la pluma
  return this.trolley;
}
```

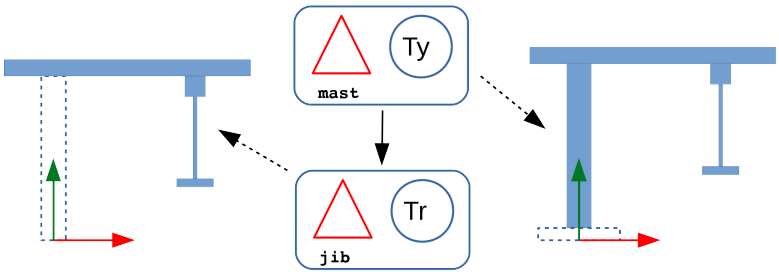
Brazo



Grua: Brazo

```
createJib () {
  this.jib = new THREE.Mesh (new THREE.BoxGeometry (. . .), this.material);

  // Tras crear la caja, se transforma para hacer coincidir el sistema de coordenadas
     donde interesa. El eje Y coincide con el eje de giro del brazo
  this.jib.geometry.applyMatrix (new THREE.Matrix4().makeTranslation (. . .));
  this.jib.castShadow = true;

  // El brazo y sus hijos se elevan para situarse encima del mástil
  this.jib.position.y = this.craneHeight;

  // El brazo y sus hijos giran con respecto al Eje Y
  this.jib.rotation.y = this.angle;

  // Al brazo se le añaden como hijos el conjunto pluma-cuerda-gancho
  this.jib.add (this.createTrolleyStringHook());

  return this.jib;
}
```

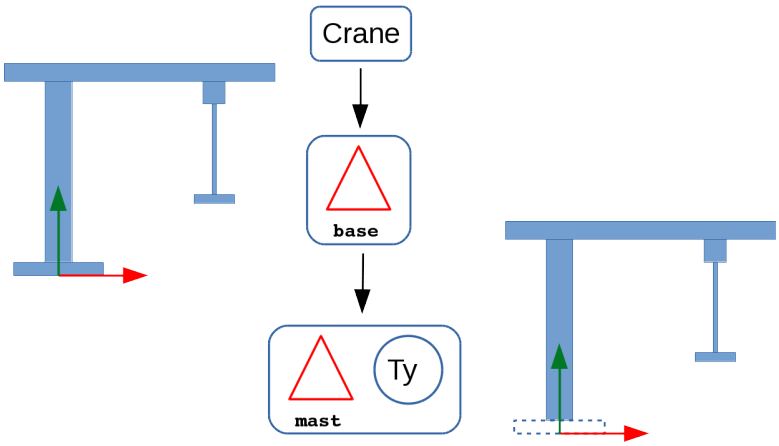
Mástil



Grua: Mástil

```
createMast () {
  var mast = new THREE.Mesh (new THREE.CylinderGeometry(. . .), this.material);
  mast.geometry.applyMatrix (new THREE.Matrix4().makeTranslation (. . .));
  mast.castShadow = true;
  mast.position.y = this.baseHookHeight; // Mástil e hijos se elevan encima de la base
  mast.autoUpdateMatrix = false;         // Esta transformación no va a cambiar más
  mast.updateMatrix();                   // Pero debe aplicarse la primera vez
  mast.add(this.createJib());            // Se le añade el brazo como hijo
  return mast;
}
```

Base

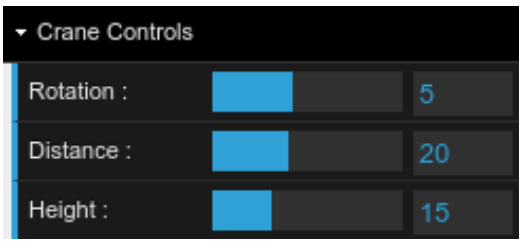


Grua: Base

```
createBase () {
  var base = new THREE.Mesh (new THREE.CylinderGeometry(. . .), this.material);
  base.geometry.applyMatrix (new THREE.Matrix4().makeTranslation (
      0, this.baseHookHeight/2, 0));
  base.castShadow = true;
  base.autoUpdateMatrix = false;    // Este nodo no se transforma nunca
  base.add(this.createMast());
  return base;
}
```

Moviendo la Grúa

Controles en la interfaz gráfica de usuario



GUI: Agrupamiento de controles

```
// En   script.js
GUIcontrols = new function() {
  . . .
  this.rotation = 6;
  this.distance = 10;
  this.height   = 10;
  . . .
}
var craneControls = gui.addFolder ('Crane Controls');
craneControls.add(controls,'rotation',0,12,0.01).name('Rotation:');
craneControls.add(controls,'distance',0,50,0.1).name('Distance:');
craneControls.add(controls,'height',0,50,0.1).name('Height:').listen();
```

Uso de los controles para mover la grua

Grúa: Establecimiento de la posición de la grúa

```
// En  TheScene.js
animate (controls) {
  . . .
  this.crane.setHookPosition (controls.rotation, controls.distance, controls.height);
}

// En   Crane.js
setHookPosition (anAngle, aDistance, aHeight) {
  this.setJib (anAngle);
  this.setTrolley (aDistance);
  this.setHook (aHeight);
}

// En diversos métodos se realiza
this.jib.rotation.y = this.angle;
this.trolley.position.x = this.distance;
// Se calcula la longitud de la cuerda según la altura al suelo del gancho
this.stringLength = this.computeStringLength ();
this.string.scale.y = this.stringLength;
this.hook.position.y = -this.stringLength;
```

Interacción con el ratón en la escena

Se definen *estados* que indican *qué se está haciendo* con la aplicación en cada momento.

Se definen métodos que procesen los eventos del ratón y se asocian dichos métodos a sus correspondientes eventos.

Cada método que procese un evento del ratón, debe consultar el estado actual de la aplicación para realizar el procesamiento correcto, ya que unas veces un clic de ratón será para añadir cajas, otras veces para seleccionar una caja y hacer algo con ella.

Se definen los siguientes estados:

- `NO_ACTION`: No se está haciendo nada
- `ADDING_BOXES`: Se ha ordenado añadir cajas
- `MOVING_BOXES`: Se ha ordenado mover cajas

Se definen también las siguientes acciones:

- `NEW_BOX`: Se va a añadir una caja
- `MOVE_BOX`: Se está moviendo una caja
- `SELECT_BOX`: Se va a seleccionar una caja
- `ROTATE_BOX`: Se está girando una caja
- `END_ACTION`: Se ha acabado la acción actual

Se van a escuchar y procesar los siguientes eventos: `mousedown`, `mouseup`, `mousemove` y `wheel`

En el *main* se añaden los *listener* y se indican las funciones que se ejecutarán cuando se produzca cada evento.

Listener: Ejemplo

```
window.addEventListener ("mousemove", onMouseMove);
```

Control de la cámara

La cámara se controla con el ratón

- El botón izquierdo orbita la cámara alrededor de su target
- La rueda hace zoom
- El botón derecho reencuadra

La aplicación también se va a manejar con el ratón. Para evitar que cualquier movimiento del ratón mueva la cámara, sólo estará activo el control del cámara si se tiene pulsado `Ctrl`

Esto se realiza en los métodos que procesan los eventos del ratón realizando la consulta sobre la pulsación de dicha tecla para hacer unas cosas u otras.

Cámara: Control de la misma solo con `Ctrl` pulsado

```
function onMouseDown (event) {
  if (event.ctrlKey) {
    scene.getCameraControls().enabled = true;
  } else {
    scene.getCameraControls().enabled = false;
    // Y ahora se incluyen el resto de acciones asociadas a este evento
  }
}
```

Añadido de cajas

Esta operación implica:

- Añadir el correspondiente botón a la GUI
- El procesamiento asociado a dicho botón, que será solo establecer ese estado en la aplicación.
- Incluir un *case* en la función que procesa el *clic-down* del ratón que le indica a la escena que se añada una caja.
- Incluir un *case* en la función que procesa el *movimiento* del ratón que le indica a la escena que se está moviendo la nueva caja por el suelo.
- Incluir un *case* en la función que procesa el *clic-up* del ratón para indicarle a la escena que ha finalizado el añadido de una caja.

GUI: Añadido de un *botón*

```
controls = new function() {
  . . .
  this.addBox    = function () {
    // Un mensaje de realimentación que se muestra en la página
    setMessage ("Añadir cajas clicando en el suelo");

    // La función solo establece el estado actual
    applicationMode = TheScene.ADDING_BOXES;
  };
  . . .
}
```

Añadido de cajas: `mousedown`

```
function onMouseDown (event) {
    if (event.button === 0) {    // Left button
      mouseDown = true;
      switch (applicationMode) {
        case TheScene.ADDING_BOXES :    // Se envía la orden a la fachada
          scene.addBox (event, TheScene.NEW_BOX);
          break;
        . . .
```

Añadido de cajas: Procesamiento en TheScene

```
addBox(event, action) { // La escena redirige la acción al suelo
  this.ground.addBox(event, action);
}
```

Añadido de cajas: Procesamiento en Ground

```
addBox (event, action) {
  . . .
  // Se obtiene las coordenadas del punto en el suelo sobre el que se ha hecho clic
  var pointOnGround = this.getPointOnGround (event);
  if (pointOnGround !== null) {
    if (action === TheScene.NEW_BOX) {
      // Se crea la nueva caja, centrada en el origen
      this.box = new THREE.Mesh (
        new THREE.BoxGeometry (. . .),
        new THREE.MeshPhongMaterial (. . .));
      // Se coloca el origen en la base, subiendo un poco la caja
      this.box.geometry.applyMatrix (
        new THREE.Matrix4().makeTranslation (0, this.boxSize/2, 0));
      box.position.x = pointOnGround.x;
      this.box.position.y = 0;
      // Se actualiza la posición de la caja
      this.box.position.z = pointOnGround.y;
      this.box.receiveShadow = true;
      this.box.castShadow = true;
      // Se añade a la lista de cajas
      this.boxes.add (this.box);
    }
  }
}
```

Añadido de cajas: Cálculo de la posición del ratón en el suelo

```
getPointOnGround (event) {
  var mouse = this.getMouse (event);

  // Se obtiene la semirecta que parte desde la cámara hacia la escena pasando por el
     puntero del ratón
  this.raycaster.setFromCamera (mouse, scene.getCamera());

  // Solo buscamos la intersección del rayo con el suelo
  var surfaces = [this.ground];
  var pickedObjects = this.raycaster.intersectObjects (surfaces);
  if (pickedObjects.length > 0) {

    // Si existe la intersección, se devuelve sus coordenadas
    return new THREE.Vector2 (pickedObjects[0].point.x,
                              pickedObjects[0].point.z);
  } else
    return null;
}
```

Arrastrado de la nueva caja que se está añadiendo

Añadido de cajas: `mousemove`

```
function onMouseMove (event) {
  if (mouseDown) {
    switch (applicationMode) {
      case TheScene.ADDING_BOXES :
        scene.moveBox (event, TheScene.MOVE_BOX);
        break;
      . . .
    }
  }
}
```

Arrastrado de cajas: Procesamiento en `Ground`

```
moveBox (event, action) {
  switch (action) {
    case TheScene.MOVE_BOX :
      var pointOnGround = this.getPointOnGround (event);
        if (pointOnGround !== null) {
          if (this.box !== null) {
            this.box.position.x = pointOnGround.x;
            this.box.position.z = pointOnGround.y;
          }
        }
      break;
  }
}
```

Finalización de la acción

Añadido de cajas: Se finaliza la acción, `mouseup`

```
function onMouseUp (event) {
  if (mouseDown) {
    switch (applicationMode) {
      case TheScene.ADDING_BOXES :
        scene.addBox (event, TheScene.END_ACTION);
        break;
      . . .
    }
    mouseDown = false;
  }
}
```

Enganchar y soltar cajas

Enganchar y soltar cajas tiene una implementación similar en cuanto al añadido de un control en la GUI, etc.

En esta sección se mostrará solo el código relevante a esta acción de enganchar y soltar una caja.

Enganchar cajas

El procesamiento de esta acción en la escena es sencillo. A partir de la posición del *gancho* se le pide al *suelo* que nos dé una *caja*. Si la hay, se le da a la *grúa*.

Enganchar cajas: `takeBox` en `TheScene`

```
takeBox () {
  var box = this.ground.takeBox (this.crane.getHookPosition());
  if (box === null)
    return 0;
  else
    return this.crane.takeBox (box);
}
```

Enganchar cajas: `takeBox` en `Ground`

```
takeBox (position) {
  if (this.boxes.children.length === 0)
    return null;    // Si no hay cajas, no hay nada que buscar
  // Se buscaría la caja más cercana a  position  y se comprueba, además, que está lo
      suficientemente cerca del gancho. Su distancia debe ser menor de un umbral.
  ...
  if (minDistance < threshold) {
    var boxCrane = this.boxes.children[nearestBox];
    this.boxes.remove (boxCrane);  // Se quita la caja de la lista de cajas del suelo
    return boxCrane;    // Se le da a la grúa
  }
  return null;
}
```

Enganchar cajas: `takeBox` en `Crane`

```
takeBox (aBox) {
  if (this.box === null) { // No hay ya una caja colgada
    this.box = aBox;
    // La caja, con su posición y rotación actualizadas, se añade como hija del gancho
    this.hook.add (this.box);
    return newHeight;
  }
  return 0;
}
```

Soltar cajas

Este procesamiento también es fácil. Se le pide a la *grua* que devuelva la *caja*, se le actualiza su posición con respecto al mundo, y se le da al suelo.

Soltar cajas: dropBox en TheScene

```
dropBox () {
  var box = this.crane.dropBox ();
  if (box !== null) {
    box.position.copy (this.crane.getHookPosition());
    box.position.y = 0;
    this.ground.dropBox (box);
  }
}
```

Soltar cajas: dropBox en Crane

```
dropBox () {
  if (this.box !== null) {
    var theBox = this.box;
    this.hook.remove (this.box);
    this.box = null;    // Ya no hay caja enganchada
    theBox.rotation.y += this.jib.rotation.y;
    this.heightMin = 0;
    return theBox;
  } else
    return null;
}
```

Soltar cajas: dropBox en Ground

```
dropBox (aBox) {
  this.boxes.add (aBox);
}
```

Mover y rotar cajas

Para mover y/o rotar una caja, primero hay que seleccionarla con el ratón y después actualizar su posición en el caso del movimiento o actualizar su orientación en el caso del giro.

Selección de una caja

Mover y Rotar Cajas: mousedown

```
function onMouseDown (event) {
    if (event.button === 0) {   // Left button
      mouseDown = true;
      switch (applicationMode) {
        case TheScene.MOVING_BOXES :
          scene.moveBox (event, TheScene.SELECT_BOX);
          break;
        . . .
```

Mover y Rotar Cajas: Procesamiento en TheScene

```
moveBox (event, action) {
  this.ground.moveBox(event, action);
}
```

Mover y Rotar Cajas: Procesamiento de la Selección en Ground

```
moveBox (event, action) {
  switch (action) {
    case TheScene.SELECT_BOX :
      var mouse = this.getMouse (event);
      this.raycaster.setFromCamera (mouse, scene.getCamera());
      // Se lanza un rayo desde el observador, la cámara, que pase por la posición del
          ratón buscando cajas
      var pickedObjects = this.raycaster.intersectObjects(this.boxes.children);
      // Si se encuentra alguna, se pone semitransparente y se referencia
      if (pickedObjects.length > 0) {
        this.box = pickedObjects[0].object;
        this.box.material.transparent = true;
        this.box.material.opacity = 0.7;
      }
      break;
    . . .
  }
}
```

Moviendo cajas

No hay diferencia entre mover una caja que se está añadiendo, o mover una caja existente que se ha seleccionado. Por tanto se usan los mismos métodos para actualizar la posición de la caja referenciada por `box`. La única diferencia es que al añadir una caja, `box` referencia a la ultima caja añadida, y al mover una caja existente, `box` referencia a la caja que se ha seleccionado.

Rotando cajas

En esta ocasión, el evento del ratón que se captura es el movimiento de la rueda del ratón. Se le hace llegar la orden a la clase `Ground` que actualiza la orientación de la caja referenciada por `box` a partir del movimiento realizado en la rueda del ratón.

Rotar Cajas: `mousewheel`

```
function onWheel (event) {
  if (mouseDown) {
    switch (applicationMode) {
      case TheScene.MOVING_BOXES :
        scene.moveBox (event, TheScene.ROTATE_BOX);
        break;
    }
  }
}
```

Rotar cajas: Procesamiento de la Rotación en `Ground`

```
moveBox (event, action) {
  switch (action) {
    case TheScene.ROTATE_BOX :
      if (this.box !== null) {
        this.box.rotation.y += event.wheelDelta / 100;
      }
      break;
    ...
  }
}
```

Apilamiento de cajas

Al añadir, soltar o mover una caja puede interferir con otras cajas. Si esto ocurre, la caja en movimiento se apilará sobre las otras.

También, si se quita una caja que tenía otras encima éstas caen, no se quedan en el aire.

Llevar a cabo esta funcionalidad se realiza sacando de la lista la caja que se está moviendo y llevándola al final de la lista. A continuación se recalculan las alturas de todas las cajas de la lista, desde la posición que ocupaba la caja que se ha movido (`k`) hasta el final de la lista.

Apilamiento de cajas: Cálculo de las nuevas alturas

```
updateHeightBoxes (k) {
  for (var i = k; i < this.boxes.children.length; i++) {
    this.boxes.children[i].position.y = 0;
    for (var j = 0; j < i; j++) {
      if (this.intersectBoxes (this.boxes.children[j], this.boxes.children[i])) {
        this.boxes.children[i].position.y = this.boxes.children[j].position.y +
          this.boxes.children[j].geometry.parameters.height;
      }
    }
  }
}
```