Interacción
Detección de Colisiones

Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Sistemas Gráficos

Grado en Ingeniería Informática

Curso 2023-2024

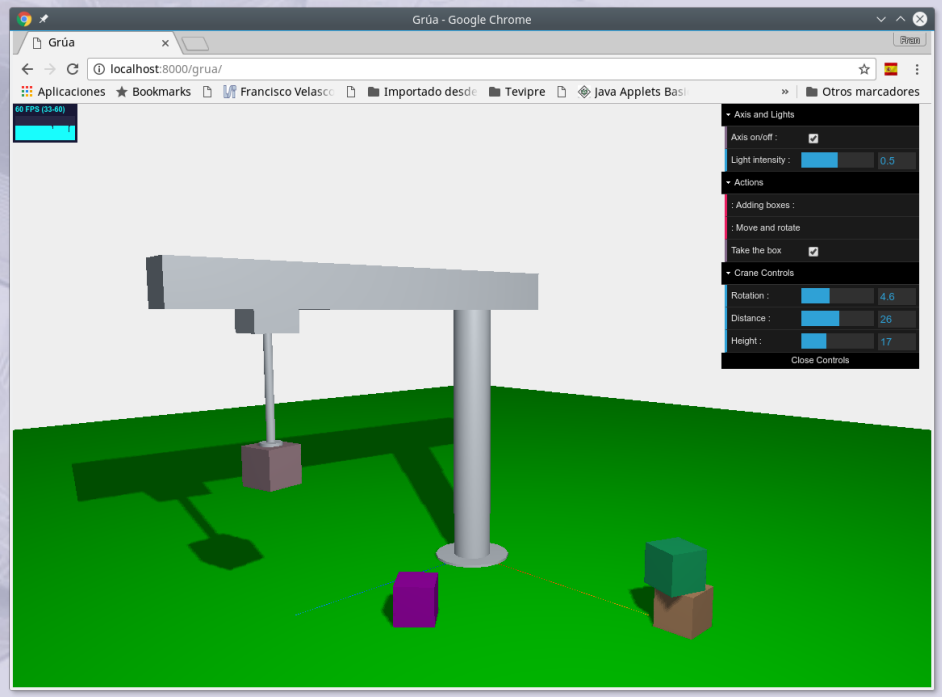
Contenidos

Objetivos

- Conocer las técnicas para que el usuario interactúe con la escena
  - Mediante el ratón
  - Mediante el teclado
- Saber seleccionar objetos de la escena con el ratón: Picking
- Conocer técnicas básicas de detección de colisiones

Introducción

- El usuario puede actuar con la escena:
  - Seleccionando objetos o posiciones (Picking)
  - Modificando objetos (su posición, orientación, forma)
  - Modificando la cámara (es un objeto más)
- Los objetos pueden interactuar entre ellos (Colisiones)
- Implica realizar búsquedas en la escena
  - Más rápidas si se tiene la escena indexada



Mensajes en pantalla

Salida HTML

- Se puede tener en el html una zona para escribir mensajes

HTML: Zona para mostrar mensajes

```
<div style="position:absolute; left:100px; top:10px" id="Messages">
</div>
```

- Y enviarle texto (formateado) desde Javascript

Javascript: Método para escribir en la zona anterior

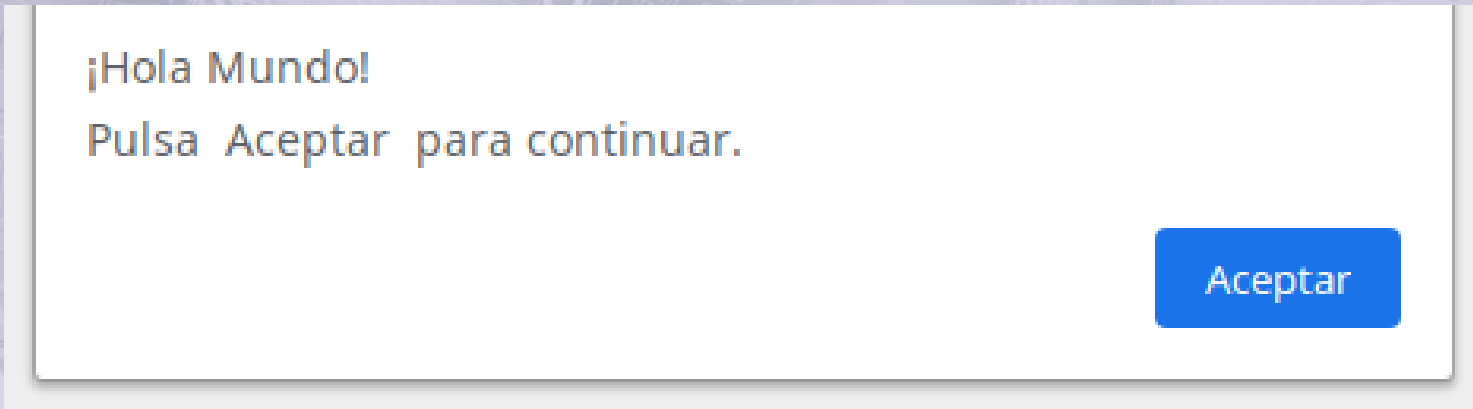
```
setMessage (str) {
  document.getElementById ("Messages").innerHTML = "<h2>"+str+"</h2>";
}
```

Ventanas pop-up

- Se muestran con `window.alert ("Texto")`

Mensajes: Ventana pop-up

```
window.alert ("Hola Mundo!\nPulsa Aceptar para continuar.");
```



Interacción con teclado

Eventos

- Se definen métodos asociados a eventos del teclado
  - `keydown`: Se pulsa una tecla
  - `keyup`: Se suelta
  - `keypress`: Pulsación y suelta.

    Este evento no lo generan las teclas modificadoras.
- En el *main* se añaden los *listener* y se indican los métodos que se ejecutarán cuando se produzca cada evento

Listener: Ejemplo

```
window.addEventListener ("keydown", (event) => scene.onKeyDown(event));
                          evento                  método
```

Interacción con teclado

Lectura de la tecla pulsada

- En dichos métodos, mediante del parámetro recibido `(event)`, se puede consultar la tecla concreta que produjo el evento a través de sus atributos `which` o `key`
  - `var x = event.which || event.key`
    - Así, la lectura del código asociado a dicha tecla funciona en todos los navegadores
  - Mediante `import * as KeyCode from 'keycode.esm.js'`
    - Se puede saber si se ha pulsado una tecla no imprimible
    - Ejemplo: `if (x == KeyCode.KEY_CONTROL)`
  - Para saber si se ha pulsado un carácter imprimible
    - Ejemplo: `if (String.fromCharCode (x) == "A")`

Interacción con teclado

Pulsación mantenida. Pulsación múltiple

- Mediante el uso variables auxiliares booleanas
  - Se evita la latencia entre el primer evento y los siguientes cuando se mantiene pulsada una tecla
  - Se puede procesar la pulsación simultánea de varias teclas

Teclado: Variables auxiliares booleanas

```
// Se tiene una variable booleana por cada tecla a procesar
var up = false;
var right = false;
. . .

// En el evento keydown se ponen a true y en el keyup se ponen a false
if (codigo == KeyControl.KEY_UP) up = true;          // o false según el evento
if (codigo == KeyControl.KEY_RIGHT) right = true;    // o false según el evento
. . .

// En el método update se consultan y se actúa en consecuencia
if (up) coche.avanza();
if (right) coche.giraDerecha();
. . .
```

Interacción con el ratón en la escena

- Se definen métodos asociados a determinados eventos del ratón
- Se definen estados que indican qué se está haciendo con la aplicación en cada momento

- Cada método que procese un evento del ratón
  - Debe consultar el estado actual de la aplicación
  - O si hay alguna tecla pulsada a la vez (ejemplo Ctrl + clic)
  - Realizar el procesamiento correcto

  Por ejemplo, hacer un clic y arrastrar el ratón puede ser:
    - ★ Realizar un movimiento de cámara
    - ★ Añadir un objeto a la escena en una posición
    - ★ Seleccionar y mover un objeto existente
    - ★ *Cualquier otra cosa . . .*

Eventos del ratón que pueden escucharse

- Entre otros, se pueden escuchar los siguientes eventos
  - `mousedown` `mouseup` `mousemove` `wheel`
- En el *main* se añaden los *listener* y se indican los métodos que se ejecutarán cuando se produzca cada evento
- Valores asociados al evento que se pueden consultar
  - `clientX`: La coordenada X del ratón
  - `clientY`: La coordenada Y
    - En coordenadas de dispositivo (int) relativas a la esquina superior izquierda de la ventana
  - `which`: El botón concreto que se ha pulsado
    - `0` (ninguno), `1` (izquierdo), `2` (central), `3` (derecho)

Ratón junto a una tecla modificadora

- Los eventos del ratón pueden realizar distintas acciones si se producen estando pulsada una o varias teclas modificadoras
- En la función que procesa un evento del ratón se puede consultar el estado de dichas teclas para realizar un procesamiento u otro
- Teclas modificadoras que pueden consultarse

  `ctrlKey` `altKey` `shiftKey`

Ejemplo: Diferente funcionalidad dependiendo de `Ctrl`

```
function onMouseDown (event) {
  if (event.ctrlKey) {
    // Se realizan unas acciones ...
  } else {
    // ... u otras
  }
}
```

Estados de la aplicación

- Un atributo en la escena indica qué se está haciendo
- Se puede establecer al elegir una opción del menú
- Se consulta desde los métodos que procesan los eventos del ratón para determinar el procesamiento a realizar

Ejemplo: Definición y usos de estados de aplicación

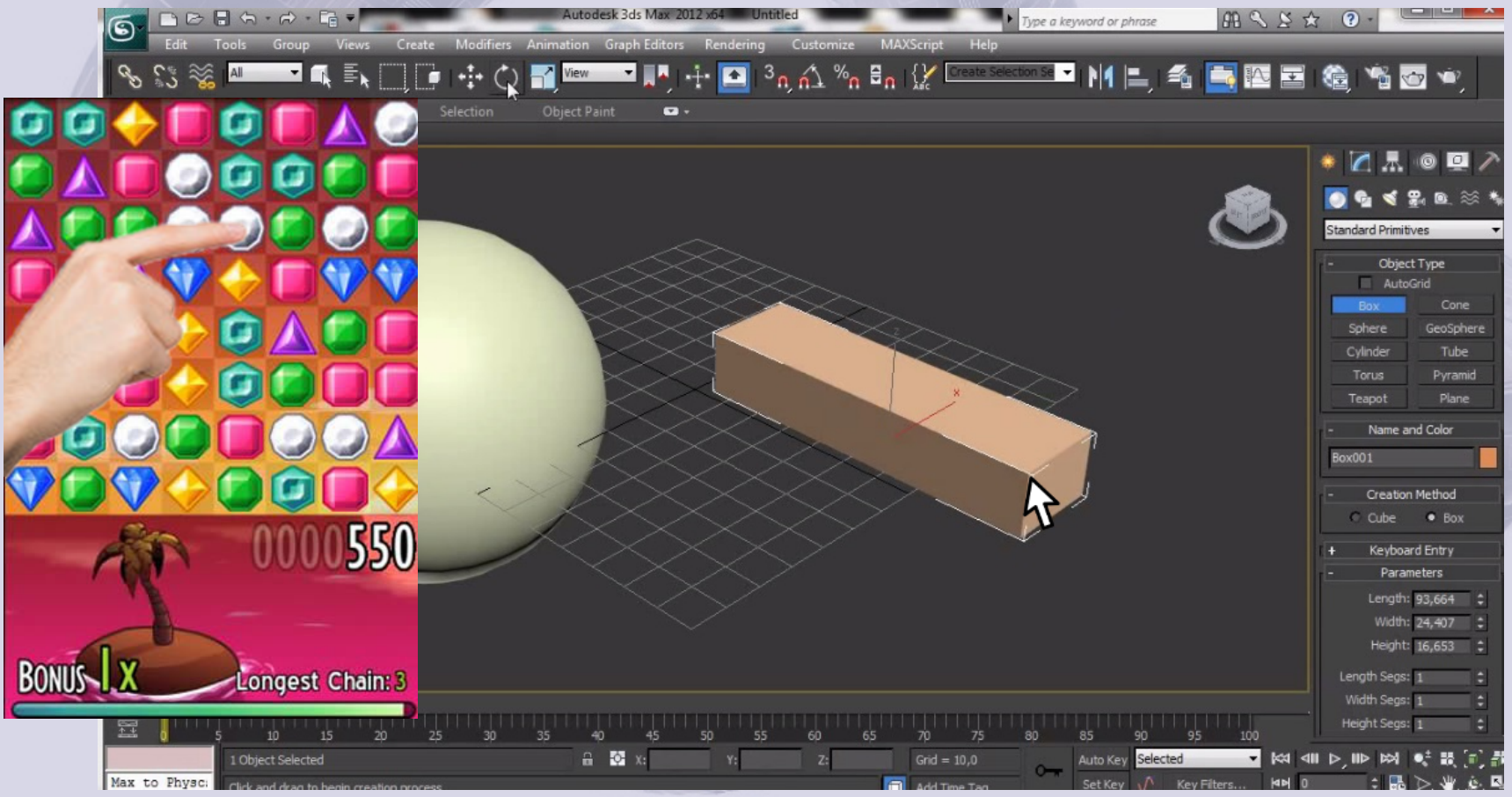
```
// Se definen (normalmente) como constantes numéricas
MyScene.NO_ACTION = 0;
MyScene.ADDING_BOXES = 1;
MyScene.MOVING_BOXES = 2;

// Se usan para darle valor al atributo de estado de la aplicación
this.applicationMode = MyScene.NO_ACTION;

// Se consulta al procesar un evento del ratón
onMouseDown (event) {
  switch (this.applicationMode) {
    case MyScene.ADDING_BOXES :
      // procesamiento para mouseDown y ADDING_BOXES
```

Selección de objetos

- **Picking**

  Seleccionar un elemento de la escena con un puntero

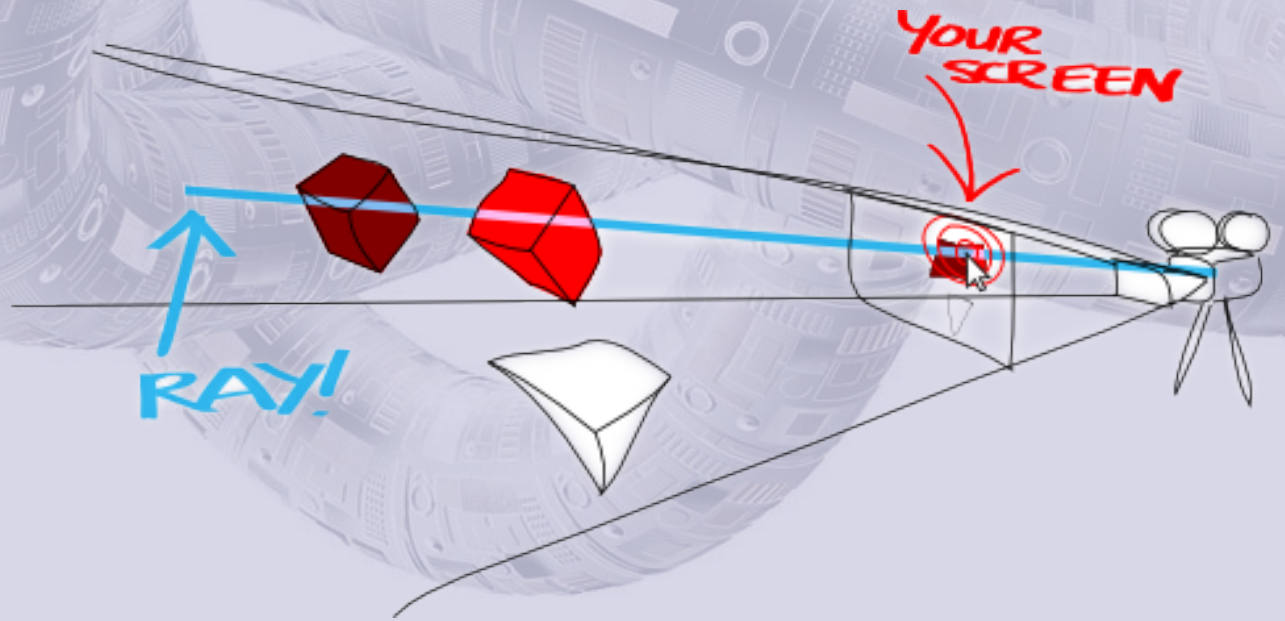
  - Suele ser una operación habitual en muchas aplicaciones gráficas



Selección de objetos (Picking)

Proceso a realizar

1. Saber en qué píxel se ha hecho clic
2. Lanzar un rayo
   - Desde la cámara
   - Que pase por dicho píxel
3. Obtener los objetos alcanzados por ese rayo
   Normalmente el seleccionado es el más cercano



Picking

Three.js

Ejemplo: Picking

```
onDocumentMouseDown (event) {
  // Suponemos que se tienen las siguientes variables
  // mouse = new THREE.Vector2 ();
  // raycaster = new THREE.Raycaster ();
  // Reutilizamos esos objetos, evitamos construirlos en cada pulsación

  // Se obtiene la posición del clic
  // en coordenadas de dispositivo normalizado
  // - La esquina inferior izquierda tiene la coordenada (-1,-1)
  // - La esquina superior derecha tiene la coordenada (1,1)

  mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
  mouse.y = 1 - 2 * (event.clientY / window.innerHeight);

  // Se actualiza un rayo que parte de la cámara (el ojo del usuario)
  // y que pasa por la posición donde se ha hecho clic
  raycaster.setFromCamera (mouse, camera);

  // Hay que buscar qué objetos intersecan con el rayo
  // Es una operación costosa, solo se buscan intersecciones
  // con los objetos que interesan en cada momento
  // Las referencias de dichos objetos se guardan en un array
```

Picking
(continuación)

Ejemplo: Picking

```
    // pickableObjects    vector de objetos donde se van a buscar intersecciones con el rayo
    // pickedObjects      vector donde se almacenan los Meshes intersecados por el rayo
    //                    ordenado desde el más cercano a la cámara hasta el más lejano

    var pickedObjects = raycaster.intersectObjects (pickableObjets, true);
    // El parámetro  true  indica que se deben buscar intersecciones en los nodos del vector y
         sus descendientes

    if (pickedObjects.length > 0) {    // hay algún Mesh clicado

      // Se puede referenciar el Mesh clicado
      selectedObject = pickedObjects[0].object;

      // E incluso el punto concreto, en coordenadas del mundo, donde se ha hecho clic
      selectedPoint =  pickedObjects[0].point;
      ...
    }
  }
```

Selección del objeto global

Uso del atributo `userData`

- Pick devuelve el `Mesh` 'clicado'
- Se puede desear acceder a la raíz del árbol de la figura
- Se usa el atributo `userData` de `Mesh`
  - En cada `Mesh` se hace que `userData` apunte a la raíz
  - Tras el Pick, se accede a la raíz mediante `userData`

Selección del objeto global

Uso del atributo `userData`

Ejemplo: Uso del atributo `userData` al construir

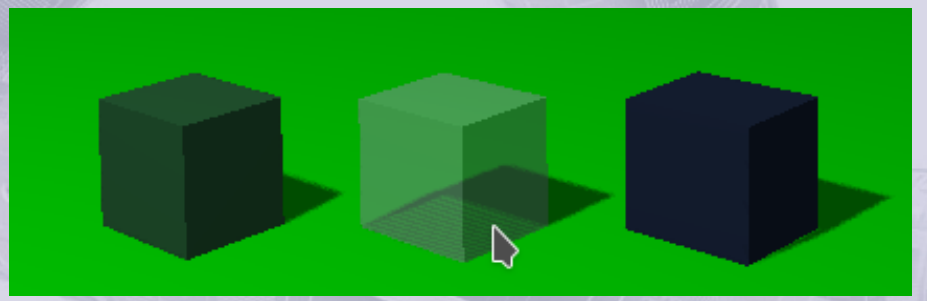
```
class DarthVader extends THREE.Object3D {
  constructor() {
    . . .
    this.cabeza = new THREE.Mesh ( . . . );
    this.cabeza.userData = this;
    . . .
  }

  recibeClic (meshConcreto) {
    . . .
  }
  . . .
}
```

Ejemplo: Uso del atributo `userData` al hacer clic

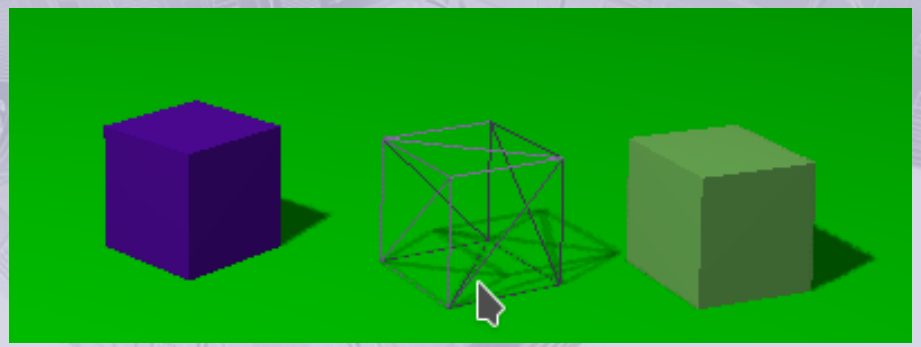
```
var meshClicado = pickedObjects[0].object;
if (meshClicado.userData) {    // Está definido?
  meshClicado.userData.recibeClic (meshClicado);
}
```

Feedback

- El objeto concreto seleccionado debe indicarse al usuario
- Se realiza con un cambio en su aspecto
  - Transparencias, cambio de color, modo alambre, etc.



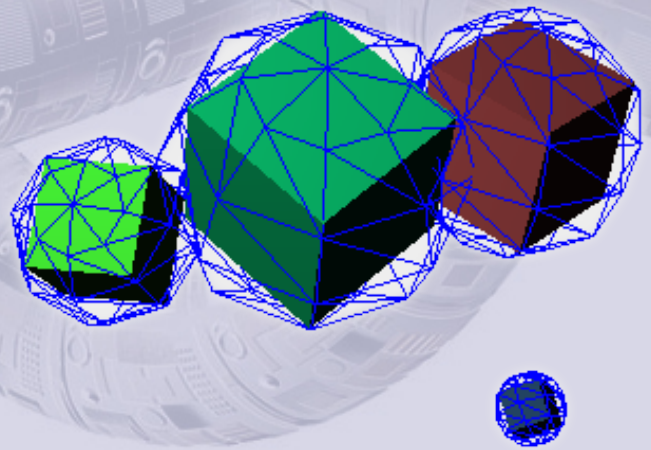
Atributo del material `opacity = 0.5` y `transparent = true`



Atributo del material `wireframe = true`

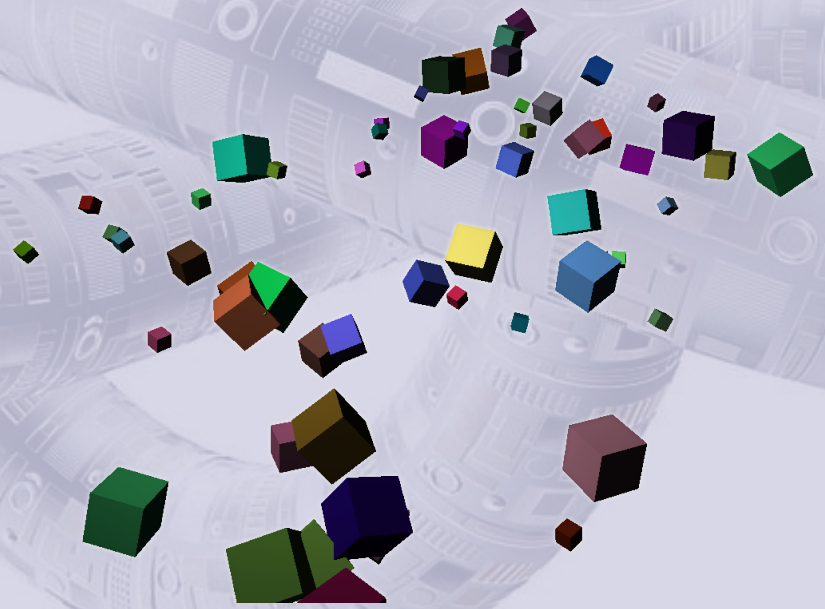
Detección de colisiones

- Suele ser necesario saber cuándo 2 objetos colisionan
- La detección de colisiones se realiza en dos fases
  - Fase gruesa:
    Se descartan rápidamente los elementos que no colisionan
  - Fase fina:
    Se determina con exactitud si 2 elementos están colisionando
- En la fase gruesa se usa:
  - Indexación espacial
  - Cajas o esferas englobantes



Indexación espacial de la escena

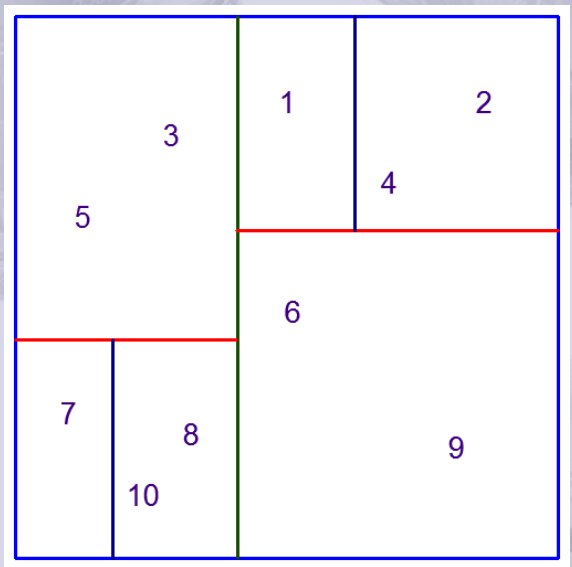
- Cuando se tienen muchos objetos en la escena es importante saber indentificarlos con rapidez
  - Cuando se lanzan rayos para Ray Tracing o Picking
  - Cuando se buscan colisiones entre objetos
- Para ello se usan estructuras de descomposición espacial

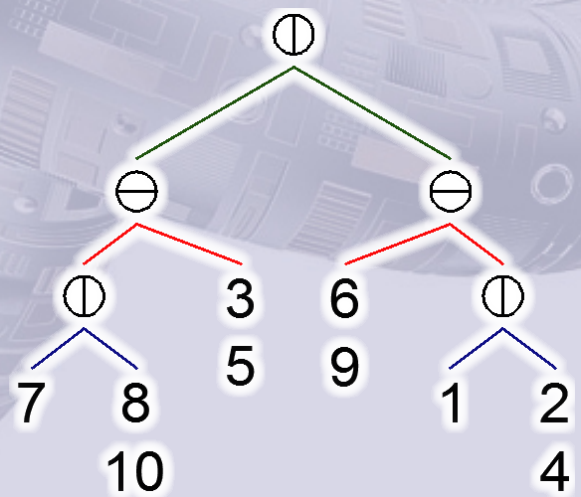


Estructuras para indexación espacial

KD-Trees

- El espacio se subdivide en 2 semiespacios por un plano
  - Solo si tiene más elementos que un límite
  - El plano está alineado con los ejes
  - Situado de manera que quede un árbol balanceado
  - En cada nivel se cambia el eje de división, alternativamente $X \rightarrow Y \rightarrow Z \rightarrow X \cdots$

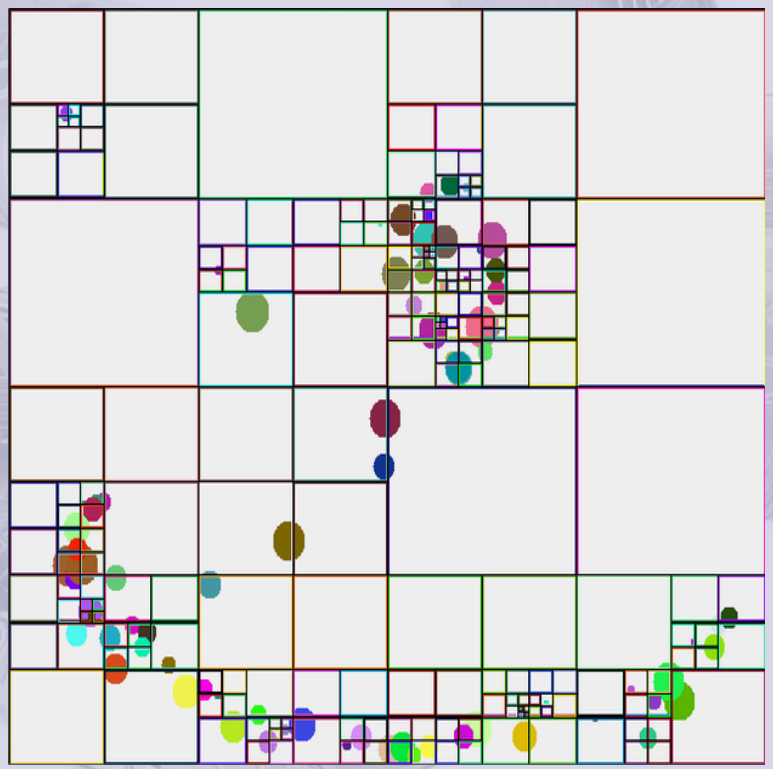




Estructuras para indexación espacial

Octrees

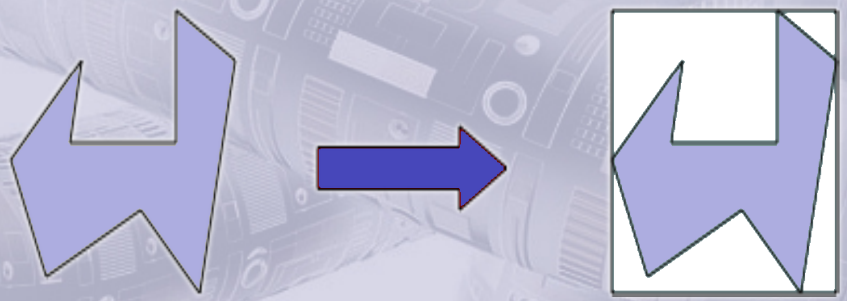
- El espacio se subdivide jerárquicamente en octantes
- Un octante solo se subdivide si contiene más objetos que el límite establecido

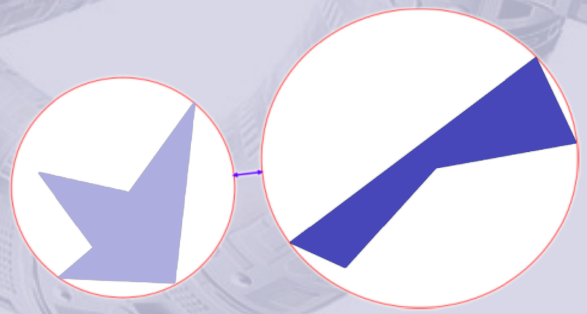


Cajas y esferas englobantes

- Formas sencillas que engloban complentamente al objeto
- Permiten saber rápidamente cuando 2 objetos no colisionan
- Según la precisión exigida, se usan para determinar la colisión

Ejercicio: Diseñar sendos algoritmos para calcular la caja y la esfera englobante de un objeto cualquiera a partir de su lista de vértices



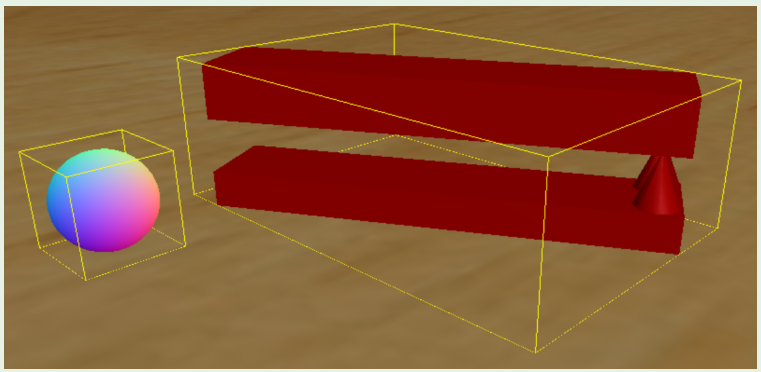


Caja englobante en Three.js

- Está representada por la clase Box3
- Se actualiza a partir de un Object3D (y sus descendientes) con el método setFromObject (object: Object3D)
  - Hay que reactualizar el `Box3` cada vez que el `Object3D` cambie
- Se puede comprobar la intersección de dos cajas enblobantes con el método intersectsBox (box: Box3): boolean

Ejemplo: Test de colisión entre 2 Object3D

```
// Suponemos que las variables figura1 y figura2 referencian a los  Object3D  a comprobar
// Suponemos que ya tenemos en las variables cajaFigura1 y cajaFigura2 las cajas englobantes
      que usaremos para detectar la colision de las figuras anteriores
var cajaFigura1 = new THREE.Box3();
var cajaFigura2 = new THREE.Box3();

// Se actualizan las cajas englobantes
cajaFigura1.setFromObject (figura1);
cajaFigura2.setFromObject (figura2);

// Se comprueba la colisión
if (cajaFigura1.intersectsBox(cajaFigura2)) {
  . . .
}
```



Caja englobante visible

- El `Box3` es invisible
- Para mostrar una caja en alambre, para depuración como en la página anterior, se recurre a la clase Box3Helper
- El `BoxHelper` hay que añadirlo al grafo de escena
- Se puede mostrar, o no, mediante su atributo `visible`

Ejemplo: Caja englobante visible

```
// Suponemos que la variable caja es un Box3 de una caja englobante
var cajaVisible = new THREE.Box3Helper(cajaFigura1, 0xFFFF00); // un color, opcional
escena.add(cajaVisible);
// Se muestra, o no
cajaVisible.visible = true; // o false
```

Detección de colisiones por raycasting

- Cuando una figura se mueve, puede 'lanzar un rayo' en esa dirección para saber con qué puede colisionar
- Se restringe la busqueda a una distancia cercana

Ejemplo: Detección de colisiones por raycasting

```
// Construimos una sola vez un  rayo  que usaremos para detectar colisiones
this.rayo = new THREE.Raycaster (
      // una posición y dirección cualquiera, se actualizan después
  new THREE.Vector3(), new THREE.Vector3(0,0,1),    0, distancia);
      // se indica la  distancia  máxima de búsqueda

// Suponemos que  figura      es un objeto que se mueve según el vector  direccion
// Suponemos que  posicion    es un Vector3 ya creado
// Suponemos que  candidatos  es un array de figuras para detectar colisión

// Configuramos el rayo y buscamos colisiones dentro de la distancia indicada anteriormente
figura.getWorldPosition (posicion);
this.rayo.set (posicion, direccion.normalize());  // La dirección debe estar normalizada

impactados = rayo.intersectObjects (candidatos, true);
if (impactados.length > 0) { // El rayo ha impactado con algún Mesh en esa distancia
  // en impactados[0].object  está el Mesh concreto
  // en impactados[0].object.userData  está la raiz de la figura a la que pertenece ese Mesh
       (en el caso de que hayamos configurado userData adecuadamente)
}
```

Interacción
Detección de Colisiones

Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Sistemas Gráficos

Grado en Ingeniería Informática

Curso 2023-2024