Interacción
Detección de Colisiones

Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Sistemas Gráficos

Grado en Ingeniería Informática

Curso 2019-2020

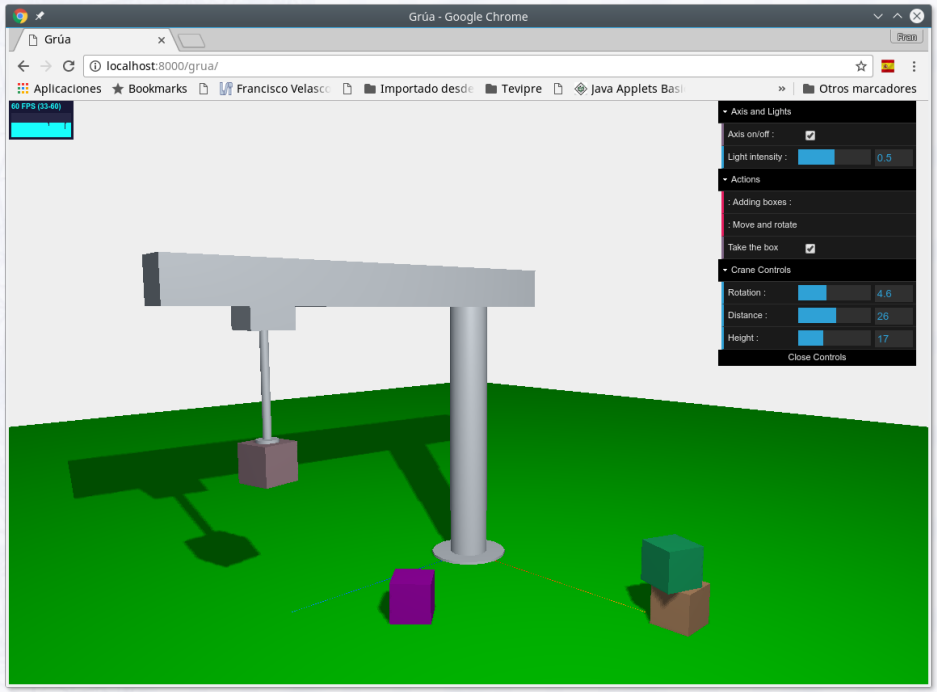
Contenidos

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
- Saber mejorar el picking y la detección de colisiones mediante la indexación espacial de los objetos de la escena

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



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

Listener: Ejemplo

```
window.addEventListener ("mousemove", (event) => scene.onMouseMove(event));
                          evento                    método
```

- Valores asociados al evento que se pueden consultar
  - `clientX`: La coordenada X del ratón (relativa a la ventana)
  - `clientY`: La coordenada Y
  - `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: Movimiento de cámara solo con `Ctrl` pulsado

```
function onMouseDown (event) {
  if (event.ctrlKey) {
    scene.getCameraControls().enabled = true;
  } else {
    scene.getCameraControls().enabled = false;
    // Se desabilitan los movimientos de cámara
    // Se realiza otro procesamiento
    . . .
  }
}
```

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
TheScene.NO_ACTION = 0;
TheScene.ADDING_BOXES = 1;
TheScene.MOVING_BOXES = 2;

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

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

Procesamiento de pulsaciones de teclado

- 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 dichos métodos se lee la tecla que produjo el evento
  - `var x = event.which || event.keyCode`
    - ★ Así, la lectura del código asociado a dicha tecla funciona en todos los navegadores
  - La lista completa de códigos se puede consultar en `https://www.w3schools.com/charsets/ref_html_utf8.asp`
  - Para saber, de manera más cómoda, si se ha pulsado un carácter imprimible se puede usar `if (String.fromCharCode (x) == "A")`

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

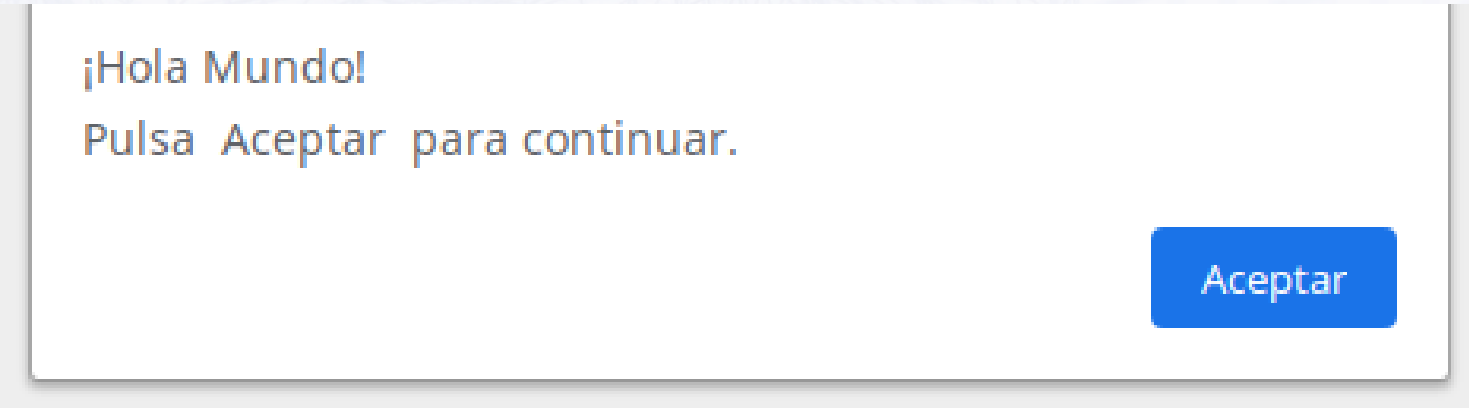
```
setMessage (str) {
  document.getElementById ("Messages").innerHTML = "<h2>"+str+"</h2>";
}
```

Ventanas pop-up

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

Mensajes: Ventana pop-up

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

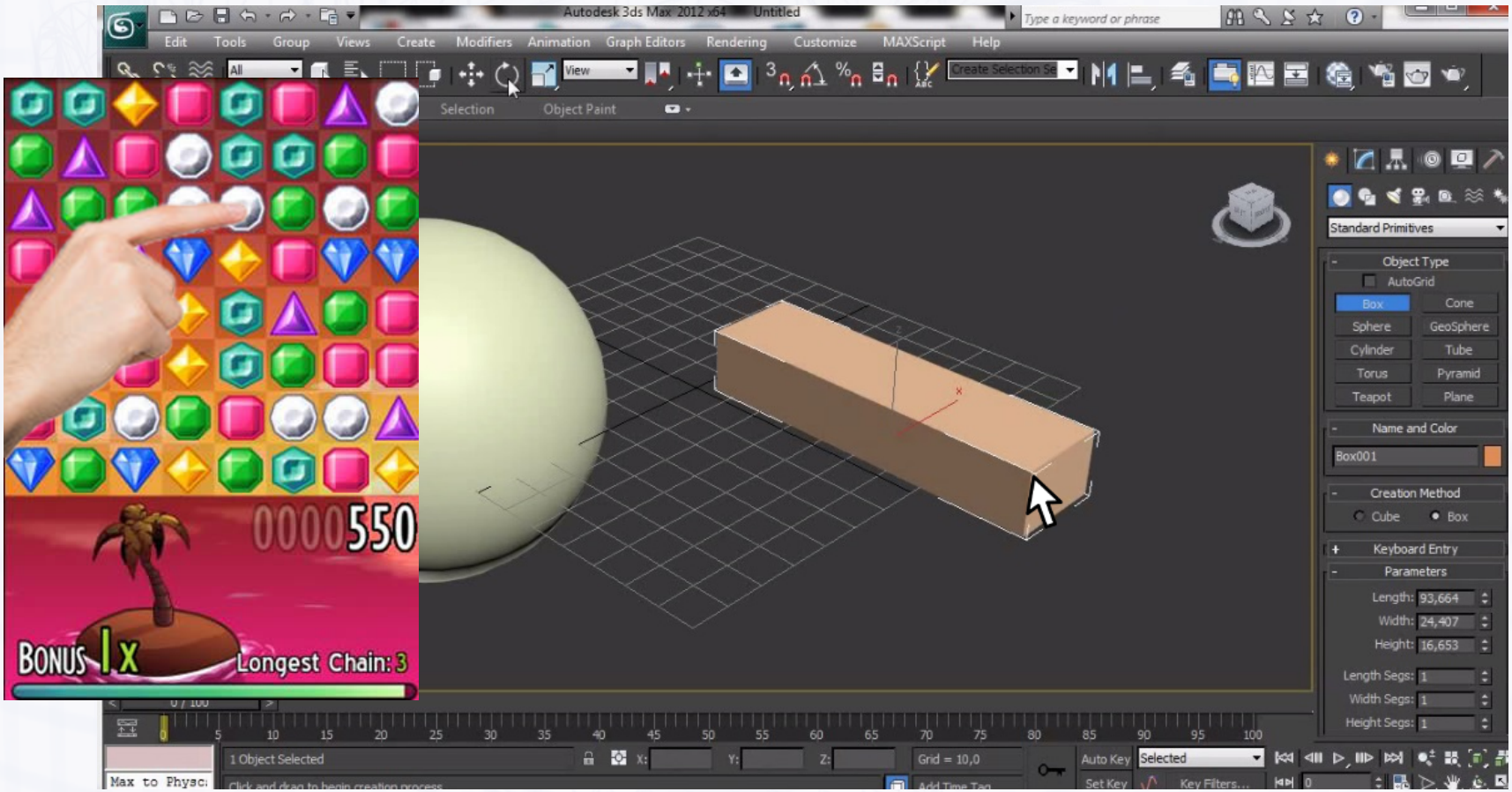


Selección de objetos

- **Picking**

  Seleccionar un elemento de la escena con un puntero

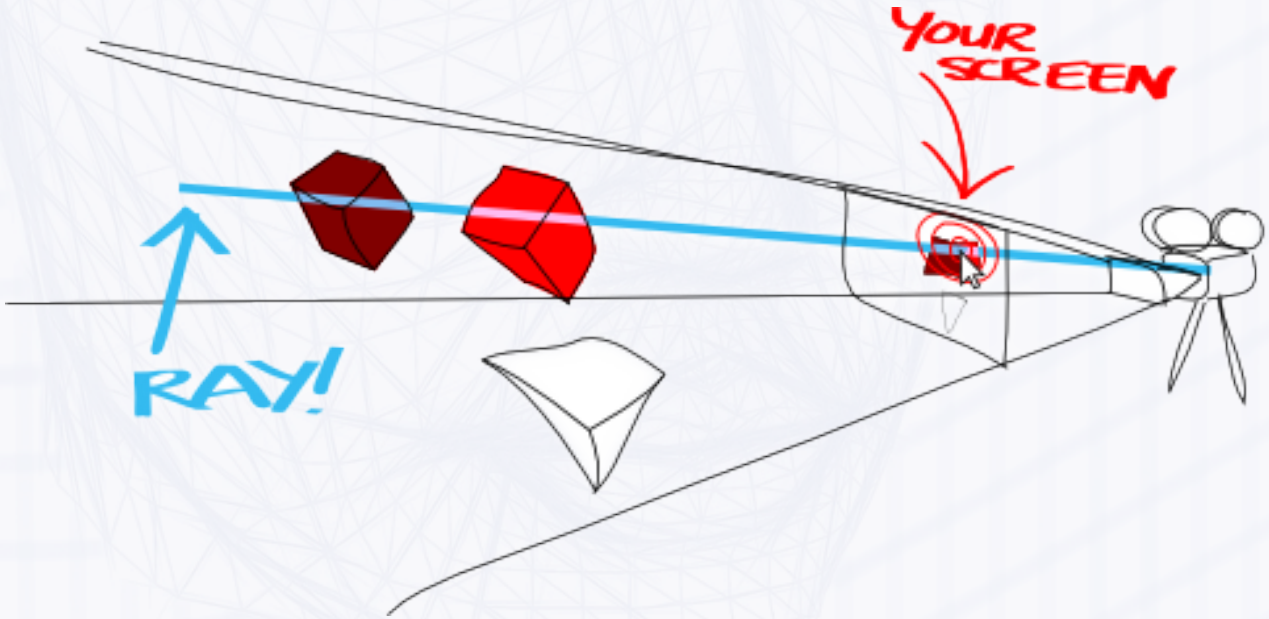
  - Suele ser una operación habitual en muchas aplicaciones gráficas



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) {
  // 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)
  var mouse = new THREE.Vector2 ();
  mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
  mouse.y = 1 - 2 * (event.clientY / window.innerHeight);

  // Se construye un rayo que parte de la cámara (el ojo del usuario)
  // y que pasa por la posición donde se ha hecho clic
  var raycaster = new THREE.Raycaster ();
  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   es el array de objetos donde se van a buscar intersecciones con el
         rayo
    // Los objetos alcanzados por el rayo, entre los seleccionables,
    // se devuelven en otro array
    var pickedObjects = raycaster.intersectObjects
        (pickableObjets, true);

    // pickedObjects es un vector ordenado desde el objeto más cercano
    if (pickedObjects.length > 0) {
      // Se puede referenciar el Mesh clicado
      selectedObject = pickedObjects[0].object;
      // E incluso el punto concreto donde se ha hecho clic
      selectedPoint =  new THREE.Vector3 (pickedObjects[0].point);
      . . .
    }
}
```

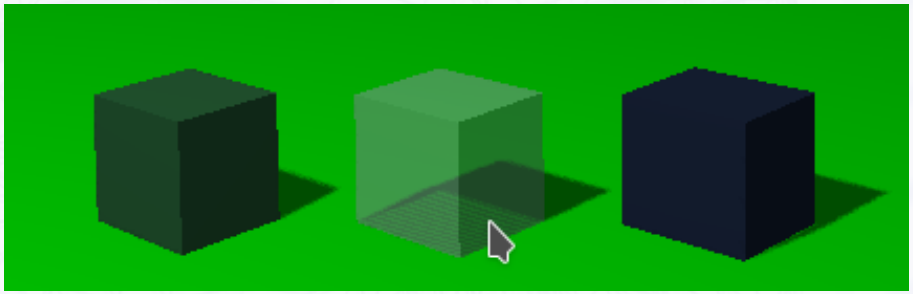
Selección del objeto global

Uso del atributo `userData`

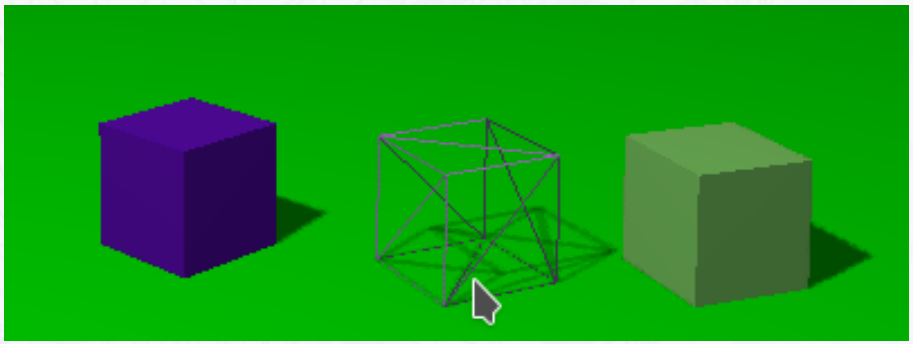
- Pick devuelve el `Mesh` 'clicado'
- Se puede desear acceder a la raíz del árbol de la figura
- Se usa el atributo `userData` de `Mesh`
  - En cada `Mesh` se hace que `userData` apunte a la raíz
  - Tras el Pick, se accede a la raíz mediante `userData`

Feedback

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

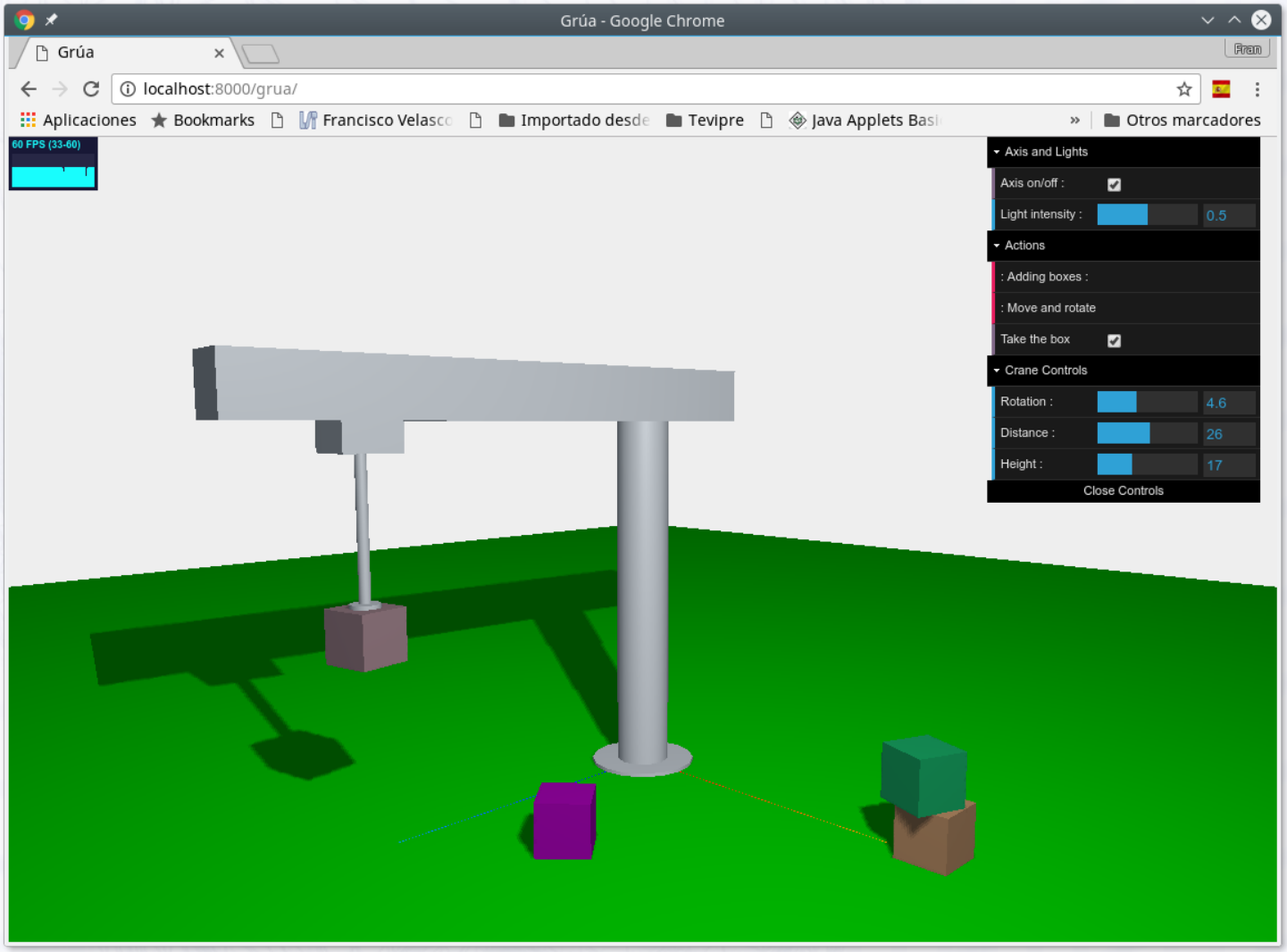


Atributo del material `opacity = 0.5` y `transparent = true`



Atributo del material `wireframe = true`

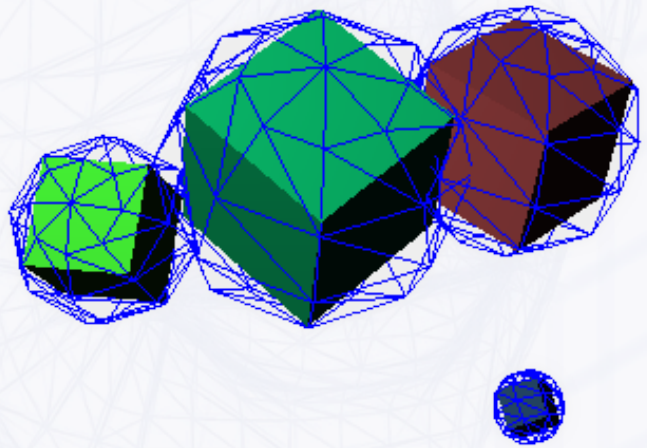
Ejemplo que incluye estas técnicas



Veamos algunos fragmentos del código (disponible completo en Prado)

Detección de colisiones

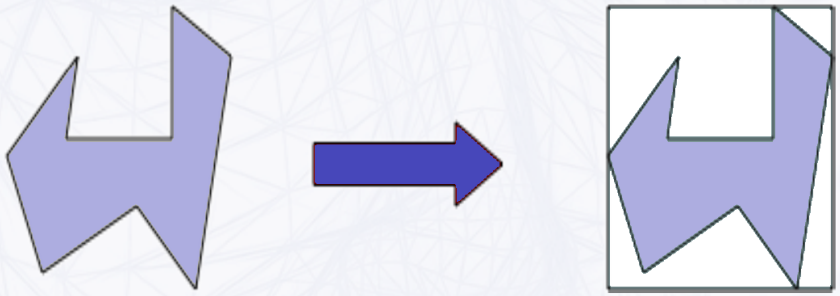
- Suele ser necesario saber cuándo 2 objetos colisionan
- La detección de colisiones se realiza en dos fases
  - Fase gruesa:
    Se descartan rápidamente los elementos que no colisionan
  - Fase fina:
    Se determina con exactitud si 2 elementos están colisionando
- En la fase gruesa se usa:
  - Indexación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

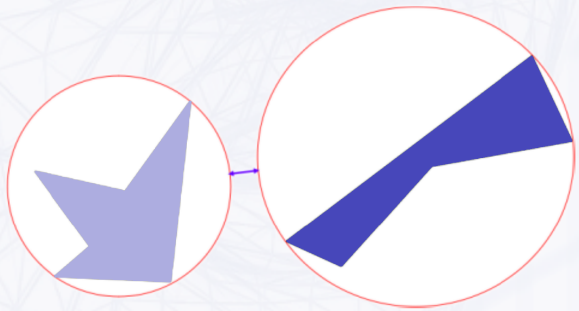


Cajas y esferas englobantes

- 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

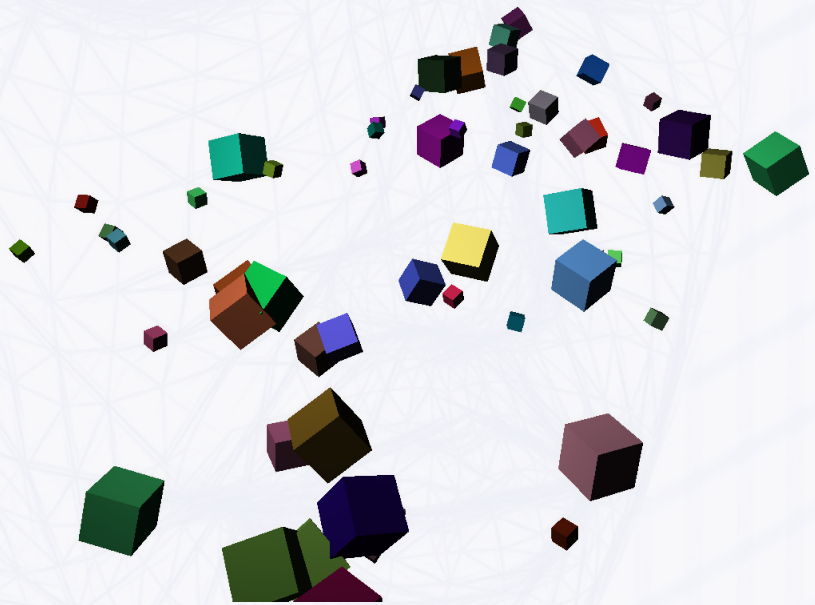
Ejercicio: Diseñar sendos algoritmos para calcular la caja y la esfera englobante de un objeto cualquiera a partir de su lista de vértices





Indexación espacial de la escena

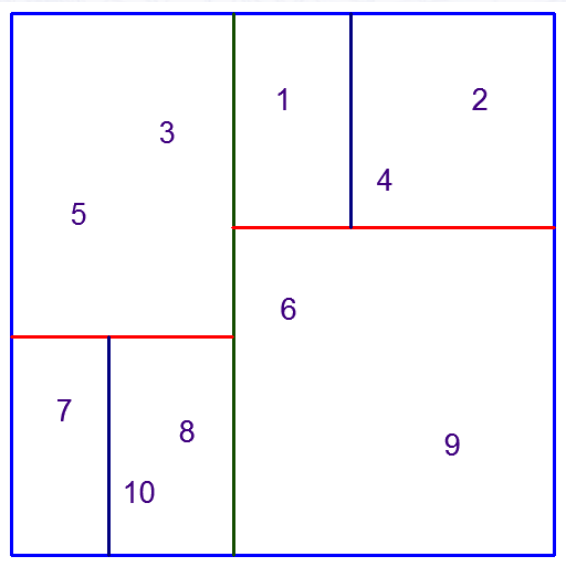
- Cuando se tienen muchos objetos en la escena es importante saber indentificarlos con rapidez
  - Cuando se lanzan rayos para Ray Tracing o Picking
  - Cuando se buscan colisiones entre objetos
- Para ello se usan estructuras de descomposición espacial

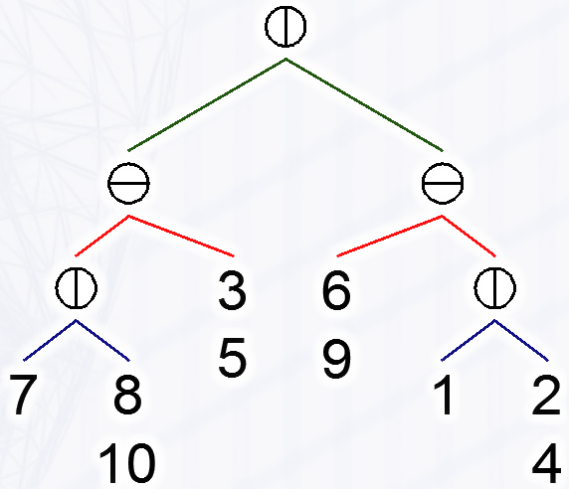


Estructuras para indexación espacial

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 $X \rightarrow Y \rightarrow Z \rightarrow X \cdots$

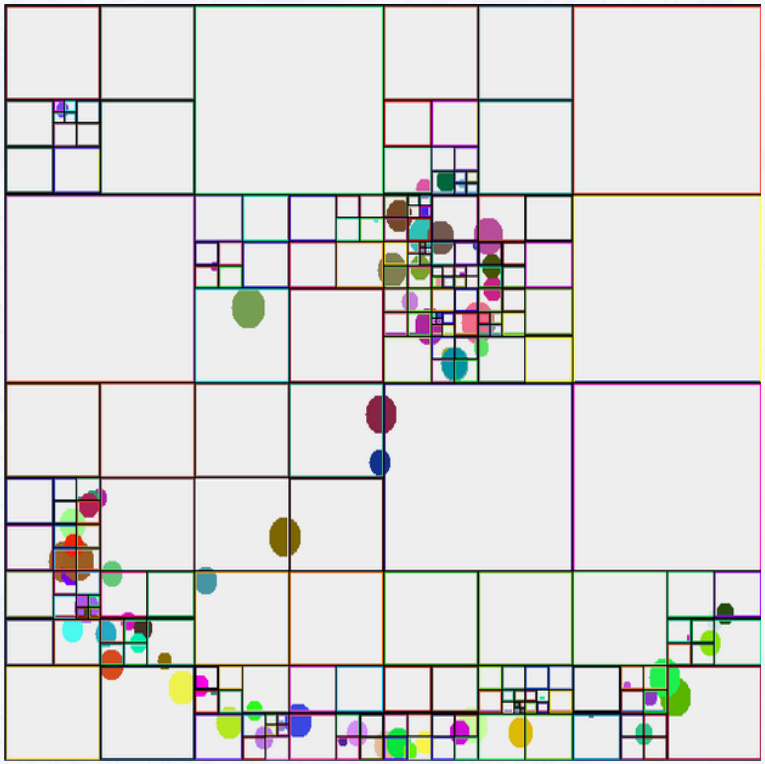




Estructuras para indexación espacial

Octrees

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



Indexación espacial mediante Octree Three.js

- Se usa la biblioteca Octree.js
  - Se encuentraba en versiones anteriores de Three.js
  - Se ha proporcionado junto con la práctica 1
- Proceso
  1. Costruir el árbol, vacío
  2. Añadirle los elementos que se quieren indexar (después se harán búsquedas sobre ellos)
  3. Realizar las búsquedas
     - ★ Por ray casting, un rayo atraviesa la escena (picking)
     - ★ Por cercanía a una posición (detección de colisiones)

Indexación espacial mediante Octree Three.js

Construcción del árbol

Octree: Construcción del árbol

```
octree = new THREE.Octree ({
  undeferred: false,
    // Si undeferred es true, los objetos añadidos al árbol
    //   se insertan inmediatamente. Si es  false,
    //   se insertan cuando se haga  octree.update()
  depthMax: Infinity,
    // Se puede establecer una profundidad máxima
  objectsThreshold: 4,
    // Numero de objetos para subdividir un nodo
  overlapPct: 0.2
    // Porcentaje de solapamiento entre nodos
    //   Facilita la gestión de los objetos que están
    //   en la frontera de un nodo
  });
```

Octree: Añadido de objetos a indexar

```
octree.add ( elObjeto, {useFaces:true});
  // Considerar las caras del objeto mejora las búsquedas
```

Indexación espacial mediante Octree Three.js

Búsquedas

Octree: Picking a través del octree

```
// Se parte de un objeto THREE.Raycaster
//   construido según la cámara actual y la posición del ratón
     (consultar la sección Picking en esta presentación)

// Se obtiene el conjunto de candidatos para el picking
octreeObjects = octree.search (
  raycaster.ray.origin,
  raycaster.ray.far,
  true,
  raycaster.ray.direction
);

// Se busca cuales de esos candidatos son atravesados por el rayo
intersections = raycaster.intersectOctreeObjects (octreeObjects);

// Si hay alguno, el más cercano está en  intersections[0]
```

Indexación espacial mediante Octree Three.js

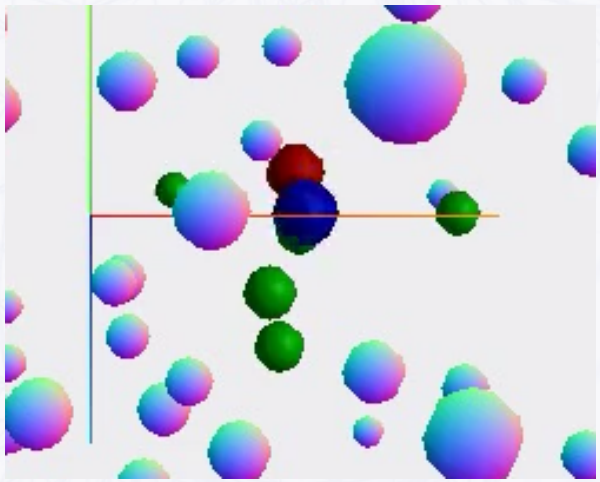
Búsquedas

Octree: Detección de colisiones a través del octree

```
// Suponer que el objeto que se está moviendo es
//   objetoMovil  y tiene un radio aproximado de  1 unidad

// Se obtiene el conjunto de candidatos para la búsqueda de colisiones
octreeObjects = octree.search (objetoMovil.position, 1, true);

// Se busca la posible colisión del objeto móvil
//   solamente con los objetos de la lista
```



Interacción
Detección de Colisiones

Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Sistemas Gráficos

Grado en Ingeniería Informática

Curso 2019-2020