técnicas avanzadas de gráficos

ingeniería multimedia

Seminario 5

Motor 2.0

Departamento de Ciencia de la Computación e Inteligencia Artificial
Universidad de Alicante

Motor 2.0

¿Qué le falta al motor para ser un verdadero motor gráfico “pro”?

Motor 2.0

- Las mejoras se orientan a mejorar la eficiencia de la visualización e incorporar animaciones
- La mayoría relacionadas con las técnicas de optimización que se ven en Videojuegos II
- Aspectos a tratar:
  - Cálculo de colisiones: volúmenes envolventes (*Bounding Volumes*)
  - Partición del espacio: *Tiles*
  - Niveles de detalle: *Level of Detail* (*LOD*)
  - Animaciones

Volúmenes envolventes

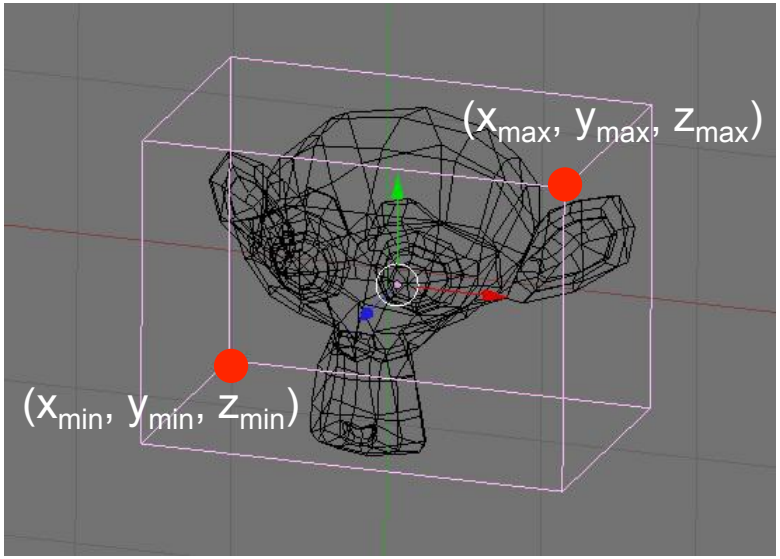
- Un volumen envolvente o *bounding volume* es un volumen cerrado, de geometría generalmente simple, que contiene completamente a un objeto
- Se utilizan para simplificar operaciones geométricas, por ejemplo, intersecciones
- Volúmenes más usuales
  - Cajas o *bounding boxes*
  - Esferas o *bounding spheres*
  - Otros: Cilindros, elipsoides, poliedros convexos…

Bounding boxes (BB)

- Son los volúmenes más utilizados por su simplicidad de cálculo y de intersección
- Dos aproximaciones:
  - *Bounding box* paralelo a los ejes
  - *Bounding box* mínimo

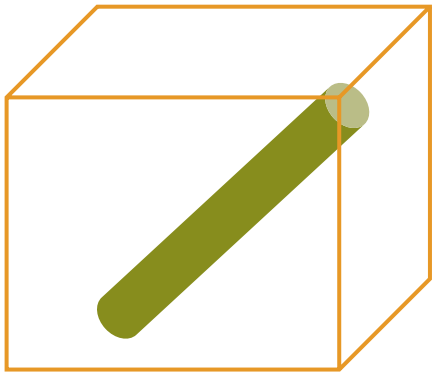
BB paralelo a los ejes

- Basta con almacenar dos puntos: las esquinas de la caja de mayor y menor (x,y,z)
- Se calcula muy fácilmente durante la lectura de la malla, guardando el mayor y el menor valor de (x,y,z) para cada punto



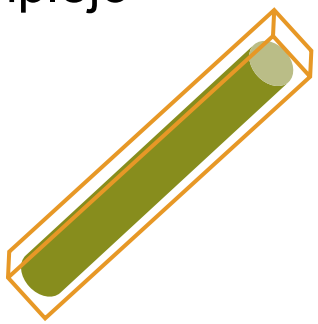
BB paralelo a los ejes

- Ventajas:
  - Muy fácil de calcular
  - Test de interioridad muy sencillo: un punto es interior al BB si todas sus coordenadas están entre los dos puntos
- Inconvenientes:
  - Pueden dejar mucho espacio vacío, por lo que los cálculos son poco precisos



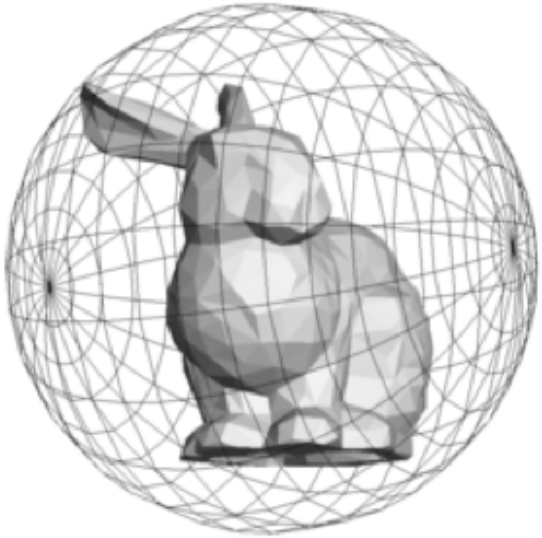
Minimum Bounding Box

- El BB tiene volumen mínimo
- Ventajas:
  - Se minimiza el espacio vacío, por lo que los cálculos son mucho más precisos
- Inconvenientes:
  - Más difícil de calcular: algoritmo de Joseph O'Rourke o aproximaciones para orientar el objeto paralelo a los ejes
  - El test de interioridad es más complejo



Bounding spheres

- También son muy utilizados, sobre todo por lo sencillo que es calcular la intersección entre una esfera y un rayo (una recta)
- Basta con almacenar el centro de la esfera y el radio

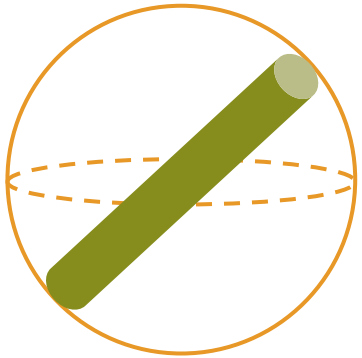


Bounding Spheres

- Algoritmo simple aproximado: no se obtiene necesariamente la esfera envolvente mínima
  - Centro = Σ vértices / nºvértices (centro de masas de los vértices del objeto)
  - Radio = máxima distancia del centro a los vértices
- Algoritmo de Ritter: obtiene la esfera envolvente mínima

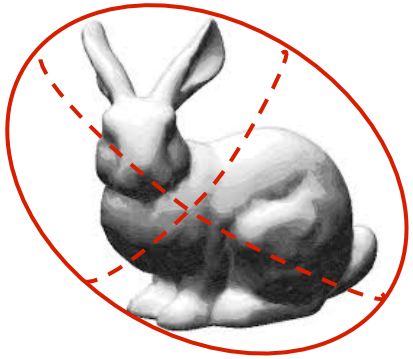
Bounding Spheres

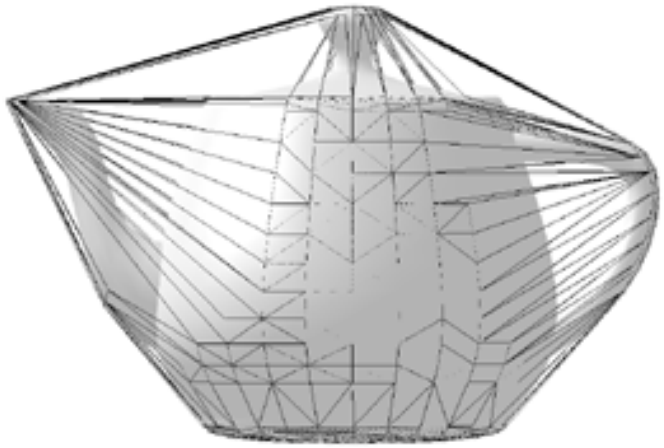
- Ventajas:
  - Fácil de calcular
  - Test de interioridad muy sencillo: un punto es interior a la BS si al sustituir sus coordenadas en la ecuación de la esfera, el resultado es <= 0
- Inconvenientes:
  - Pueden dejar mucho espacio vacío, por lo que los cálculos son poco precisos



Otros *Bounding Volumes*

- Se pueden utilizar otros volúmenes envolventes para mejorar la aproximación al objeto
- BV con geometría simple:
  - Cilindros
  - Elipsoides
- BV para minimizar el espacio vacío:
  - Poliedros: aproximaciones a la envolvente convexa (*Convex Hull*)





BVs en el motor

¿Dónde colocamos los BVs en el motor?

¿Qué entidades pueden tener BVs?

BV en el motor

- Proponemos BVs asociados a las mallas, aunque se podrían asociar también a cámaras o luces
- En una versión simple, almacenamos junto a la malla su BV:
  - Si es un BB: dos vértices, para almacenar los valores mínimos y máximos
  - Si es una BS: un vértice, para almacenar el centro, y el radio

Partición del espacio

- Una de las formas más eficientes de optimizar en gráficos es reducir al mínimo el flujo de datos geométricos, mediante técnicas de *clipping* (recortado) y *culling* (eliminación de oclusiones)
- Estas técnicas suelen necesitar una partición del espacio
- Particionar el espacio (*Space Partitioning*) es dividir el espacio en regiones disjuntas (no solapadas)
  - Partición jerárquica: Arboles octales (*Octrees*), Arboles de partición binaria (*BSP*)
  - Partición lineal: *Tiles*

Partición del espacio mediante *Tiles*

- Los juegos basados en Tiles son muy frecuentes
- Es la partición más sencilla: el espacio se representa como una matriz.
- Cada una de las celdas de la matriz representa una pequeña región cuadrada (en 2D) o cúbica (en 3D)
- El espacio queda así particionado en forma de *tiles* o teselas

Partición de espacio mediante *Tiles*

- La partición en *tiles* facilita el recorte (*culling*)
- Dadas las coordenadas de un *tile* es muy fácil saber si se dibuja, viendo si es interior al volumen de visualización (*frustum*)

Tiles en el motor

¿Dónde colocamos los *tiles* en el motor?

¿Cómo los gestionamos?

Tiles en el motor

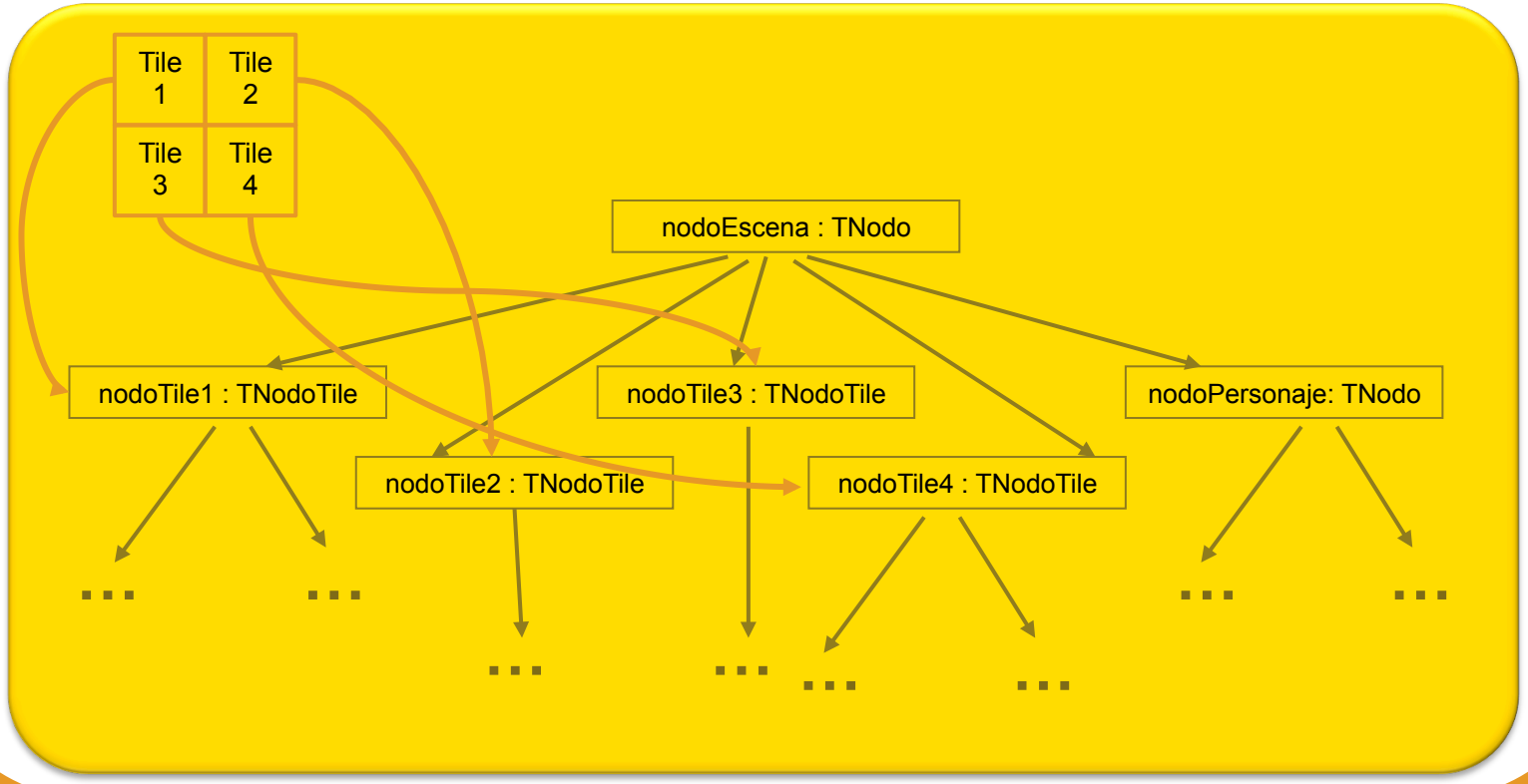
- Con pocas modificaciones podemos hacer que nuestro motor incorpore la partición mediante *tiles*.
- Los nodos ahora pueden ser de tipo *tile*, y en ese caso, deben contener la posición (x,y) del *tile* en la matriz de *tiles*
- Alternativas:
  - Nuevos atributos para la clase **TNodo**, que indiquen si es un *tile* y su posición
  - Nueva clase **TNodoTile** derivada de **TNodo** que incorpora la posición del *tile*
- Para el acceso rápido a los *tiles*, podemos llevar un registro, parecido al de cámaras y luces

Tiles en el motor

- Los nodos *tile* siempre ocupan una posición intermedia en el árbol, normalmente entre la raíz y la primera transformación de la rama
- De los nodos *tile* cuelgan todos los elementos del escenario
- El personaje, colgará de otra rama de tipo distinto a *tile*
- La función de dibujado del nodo cambia:
  - Si el nodo es de tipo *tile* y ese *tile* no es visible (se sabe gracias a la posición) no avanzar por esa rama
  - En caso contrario, dibujar normalmente

Tiles en el motor

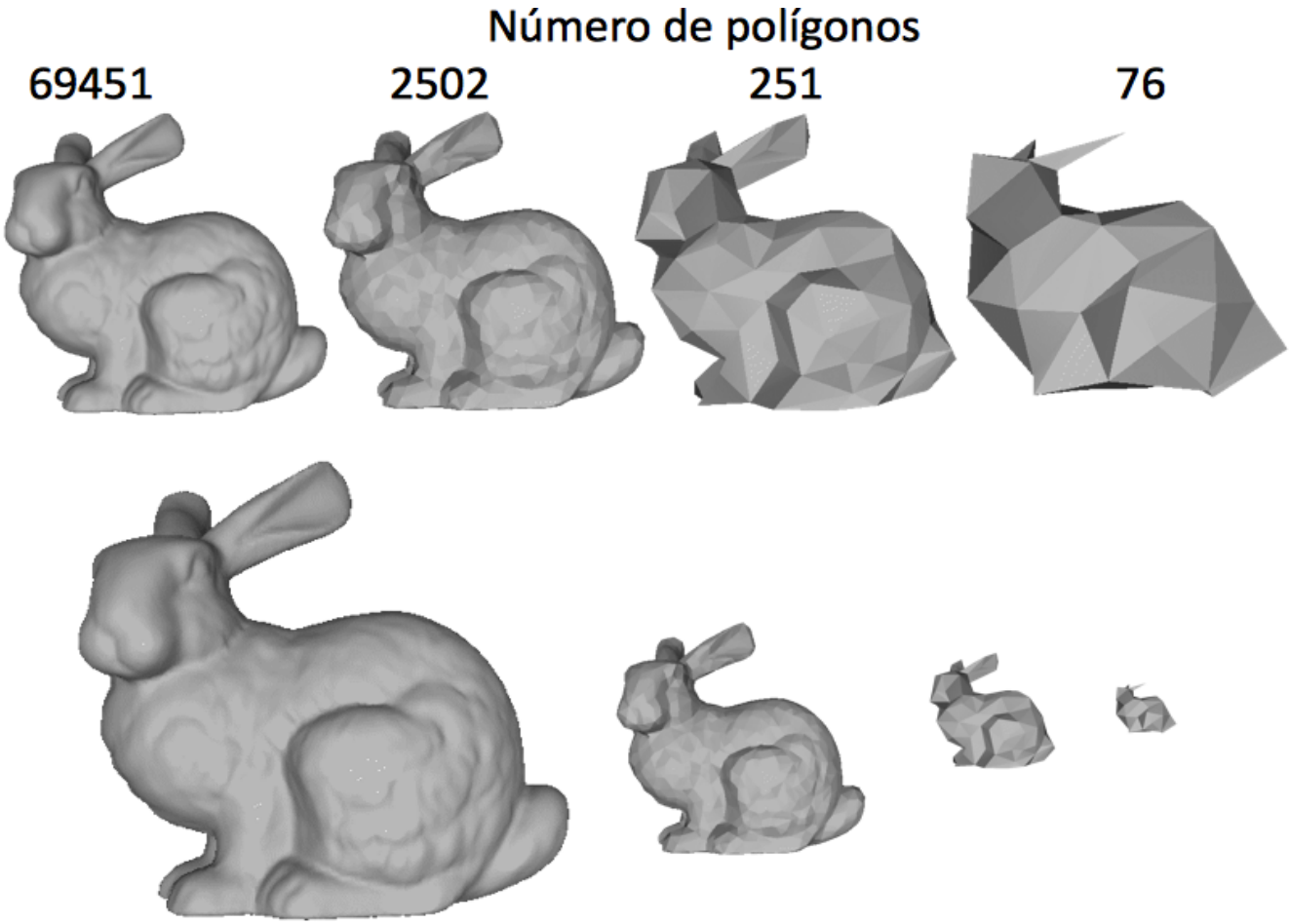
- Clase derivada y registro de *tiles*



Niveles de detalle

- LODs = Levels of Detail = Niveles de detalle
- Se refiere a posibilidad de trabajar con varios niveles de resolución según convenga en cada caso
- El nivel de detalle utilizado dependerá de:
  - Distancia al observador
  - Importancia del objeto
  - Velocidad de movimiento del objeto o de la cámara
  - Posición del objeto en la escena
- Los niveles de detalle pueden afectar a:
  - Número de polígonos que forman la malla
  - Resolución de las texturas a aplicar (mipmapping)
  - Complejidad de los shaders a aplicar

Niveles de detalle



Niveles de detalle

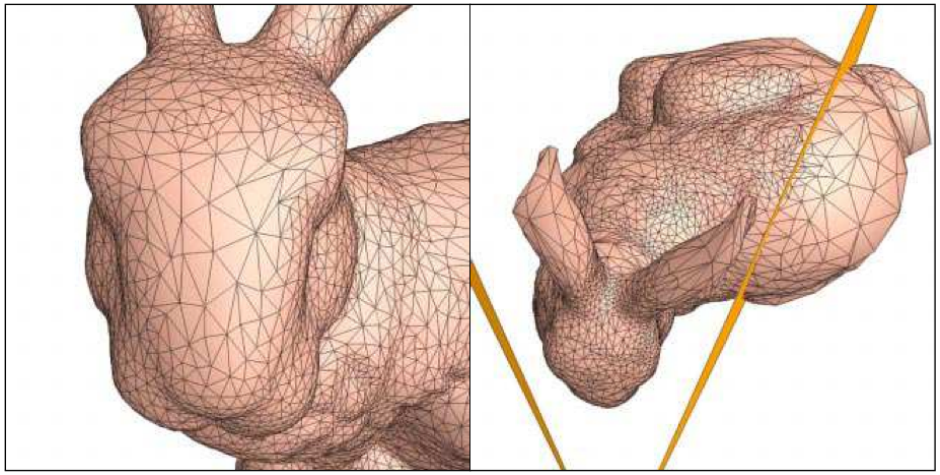
- ¿Cuándo utilizar LODs?
  - Cuando no tengamos suficiente potencia de cálculo para dibujar a la velocidad necesaria → aplicaciones en tiempo real → videojuegos
  - Cuando los objetos tienen detalles que incrementan el número de polígonos pero que no se ven a cierta distancia
  - Cuando tenga objetos redondeados u orgánicos que utilizan mallas de polígonos muy tupidas para eliminar aristas

Niveles de detalle

- ¿Cuándo no vale la pena utilizar LODs?
  - Cuando los modelos no tienen una gran cantidad de polígonos
  - Cuando no podamos reducir el número de polígonos en al menos un 30%
  - Cuando no tengamos necesidades de tiempo real

Tipos de LODs

- LODs discretos: En tiempo de diseño, se modelan varias mallas, una para cada resolución
- LODs contínuos: Sólo se modela una malla, que se simplifica en tiempo de ejecución
- LODs dependientes de la vista: Sólo una malla, que se simplifica en tiempo de ejecución y que es más o menos densa en cada parte, dependiendo de la distancia al observador



LODs en el motor

¿Dónde colocamos los LODs en el motor?

¿Cómo los gestionamos?

LODs en el motor

- Utilizaremos LODs discretos. Consejos:
  - Utilizar las herramientas automáticas que nos proporcionan los programas de modelado cuando sea posible. Es probable que hay que retocar la malla
  - Los modelos a baja resolución serán vistos desde lejos → ocultar todos los detalles que no se vayan a ver
  - No eliminar partes que supongan variar el perfil del objeto
  - Eliminar todo lo que quede cubierto. Por ejemplo, en un coche podemos eliminar los asientos, pero no las ruedas
  - No alterar el mapeado de texturas, es lo que más fácilmente detecta el ojo humano

LODs en el motor

- Con pocas modificaciones podemos hacer que nuestro motor incorpore LODs.
- Nueva entidad **TLod** (puede derivar de **TEntidad** directamente o de **TMalla**):
  - En vez de almacenar un recurso TRecursoMalla, almacena un *array* de mallas, una para cada nivel de detalle.
  - El dibujado necesita un parámetro nuevo que permita elegir una malla u otra en función del aspecto que consideremos (distancia al observador, importancia …)

Animaciones

- Animación: Cualquier cambio en la imagen, en función del tiempo:
  - Cambios de posición: traslación, rotación, ...
  - Cambios de tamaño: escalado
  - Cambios en las formas
  - Cambios en los colores y las texturas
  - Cambios en la iluminación
  - Cambios en los parámetros de la cámara

Animaciones

- Tipos de animación
  - Animación cuadro a cuadro
  - Animación de cuadros clave (keyframing)
  - Morphing
  - Animación procedimental: objetos blandos, estructuras articuladas, sistemas de partículas …
- La forma más sencilla de animación es cuadro a cuadro (la que incorporaremos al motor)

Animación cuadro a cuadro

- El diseñador debe modelar cada una de las poses correspondiente a los fotogramas o cuadros
- Posibilidades para modelar un movimiento:
  - Animación a través de una herramienta de modelado, capturando la malla en cada pose del movimiento, correspondiente a un fotograma
  - Captura del movimiento de un actor real, obtención de la secuencia sobre el modelo virtual y transformación en una secuencia de poses

Animaciones en el motor

¿Dónde colocamos las animaciones en el motor?

¿Cómo las gestionamos?

Animaciones en el motor

- Con pocas modificaciones podemos hacer que nuestro motor incorpore animaciones cuadro a cuadro.
- Nueva entidad **TAnimacion** (puede derivar de **TEntidad** directamente o de **TMalla**):
  - En vez de almacenar un recurso TRecursoMalla, almacena un *array* de mallas, una para cada pose de la animación
  - El dibujado necesita un parámetro nuevo que permita elegir una malla u otra en función del frame en el que nos encontremos
  - La interpolación entre poses es más compleja