Informática Gráfica II

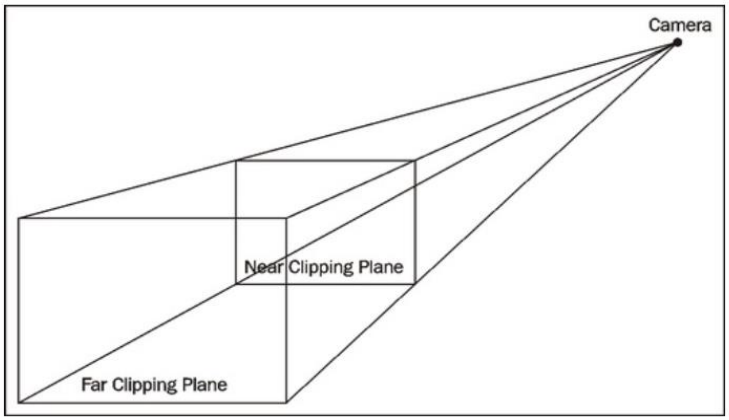
Object-oriented Graphics Rendering Engine

Estructuras de Datos espaciales

Curso 24/25: Alberto Núñez
Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

Frustum

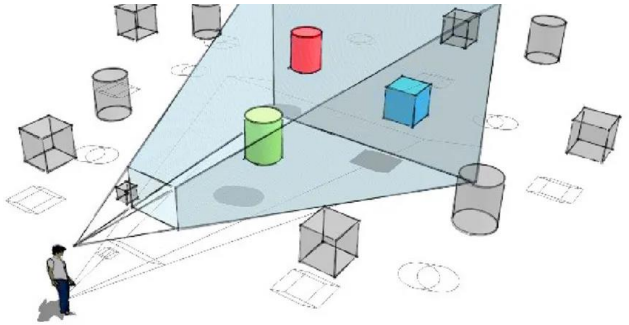
- El frustum es la figura sólida que contiene "todo lo que se puede ver" desde la cámara
- Pirámide truncada
  - Intersección de dos planos:
    - *Near clipping:* : La distancia más cercana a la cámara a la que se renderizan los objetos.
      - Todo lo que esté más cerca de este plano no es visible.
    - *Far clipping:* La distancia más alejada de la cámara a la que se renderizan los objetos.
      - Todo lo que esté más allá de este plano no será visible.



Fuente: Ogre 3D 1.7 Beginner's Guide. Felix Kerger

Culling

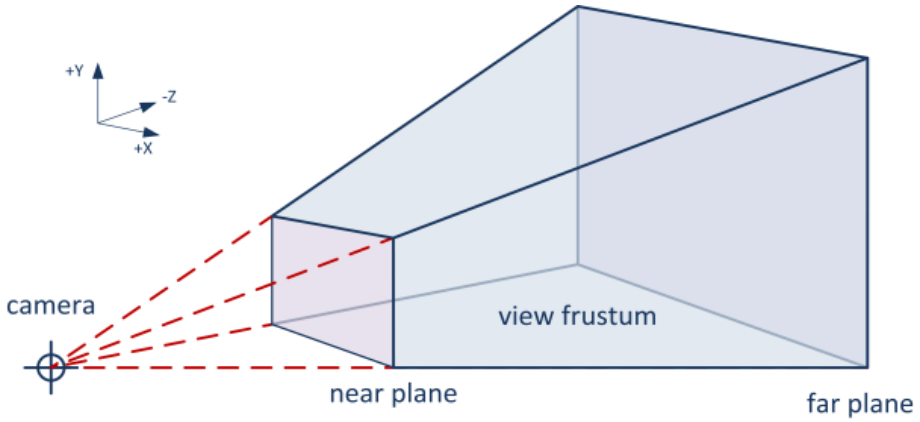
- En castellano se denomina "eliminación selectiva"
- Técnica que consiste en descartar objetos - o partes de objetos - que no es necesario renderizar.
- Ayuda a mejorar el rendimiento del renderizado reduciendo el número de polígonos procesados.
- Existen varios tipos de culling:
  - Backface culling: Eliminación de caras traseras
  - Frustum culling: Eliminación de las caras de una malla que están fuera del frustum
  - Occlusion culling: Eliminación de objetos que están ocultos por otros elementos de la escena
- Orden de complejidad de los tipos de culling:
  - Backface culling opera a nivel de una sola cara
  - Frustum culling opera a nivel de conjuntos de caras
  - Occlusion culling opera a nivel de objetos.
  - En consecuencia:
    - BC < FC < OC
    - donde < representa "es más sencillo que"



Fuente: https://aayushsoni92.medium.com/culling-in-game-development-710e12c347d4

Frustum culling

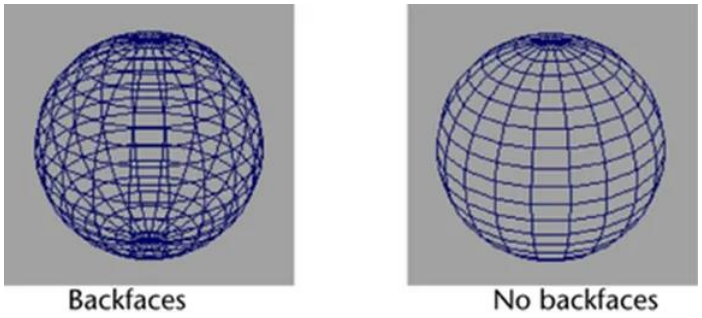
- Sólo se tienen en cuenta para el renderizado las caras "dentro del frustum"
- Se ahorra en potencia de cómputo
- Ley de la informática gráfica: "No pintes lo que no se ve"
- Objetivo: Identificar aquellos objetos ubicados fuera del frustum
  - Así no se tienen en cuenta para el renderizado
- Filtrar aquellos que no intersecan
- Uso de estructuras de datos que dividen el espacio para optimizar el proceso



Fuente: https://learnopengl.com/Guest-Articles/2021/Scene/Frustum-Culling

Backface culling

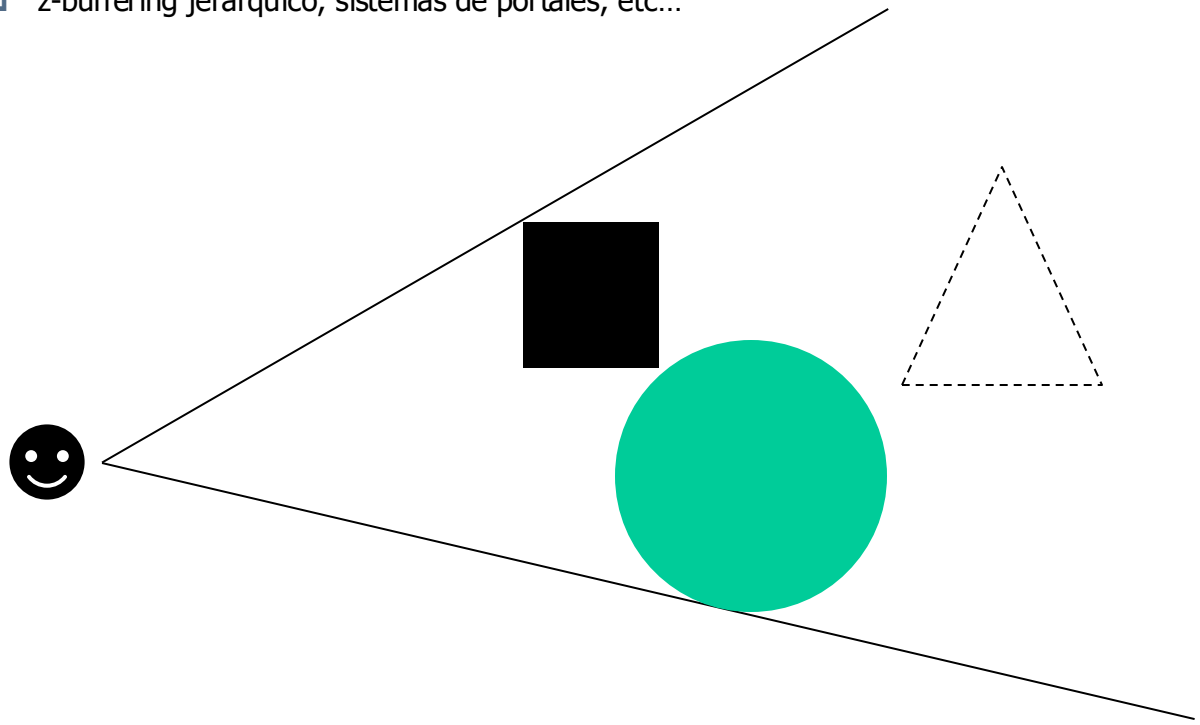
- Descarta las caras de los objetos que están "de espaldas" a la cámara
- Las caras – de un objeto 3D - que están orientadas en sentido contrario a la cámara no son visibles.
  - Al no renderizar estas caras, se reduce el número de polígonos procesados.



Fuente: https://aayushsoni92.medium.com/culling-in-game-development-710e12c347d4

Occlusion culling

- No renderizar elementos ocultos por otros elementos de la escena
- Si no se ven, ahorramos tiempo de cómputo en el renderizado
- En este caso el triángulo no se ve, lo impiden el cuadrado y el círculo
- Es más complejo que los anteriores
  - z-buffering jerárquico, sistemas de portales, etc…



EEDD para optimizar estas técnicas

- Dividir el espacio en celdas que contienen objetos y hacer las siguientes hipótesis:
  - La división y asignación solo se hace una vez
  - La división es jerárquica, de forma que si una celda no interseca, sus hijos tampoco (poda).
  - El proceso de división de una celda no es costoso.
- La mayoría tiene carácter jerárquico ⟶ árboles
  - Un nodo interno tiene asociado un volumen que recubre las primitivas que incluyen sus hijos.
  - Una hoja tiene una colección de (referencias/índices) a primitivas.
- Exploración recursiva
  - Si el volumen de un nodo interno no es de nuestro interés, todo ese subárbol no se explora (poda).
- Construcción
  - Mediante inserciones sucesivas de distintas primitivas
  - Algoritmo top-down
  - Criterios de terminación:
    - Nunca superar una profundidad determinada.
    - Hojas de tamaño limitado.

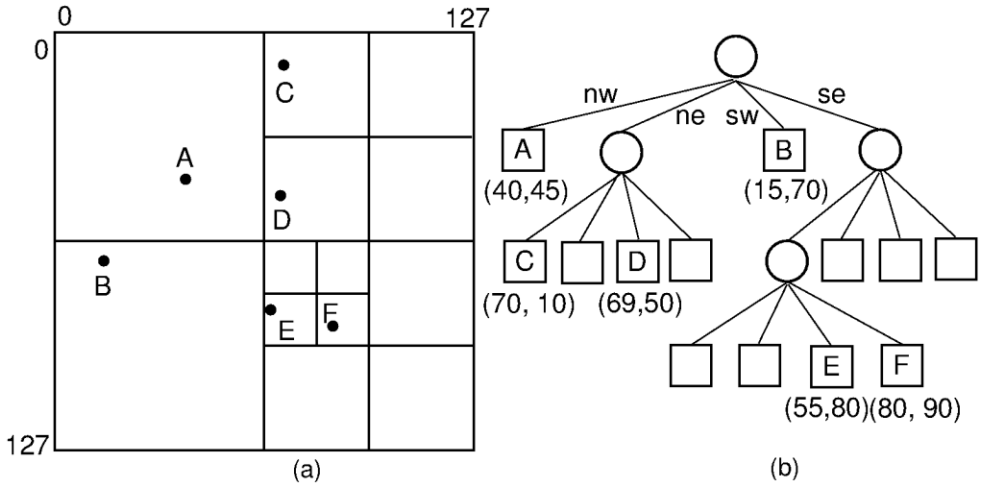
EEDD para optimizar estas técnicas

- Eficiencia
  - Determinar si el volumen asociado a un nodo interno es fácil.
  - Árboles equilibrados:
    - Algoritmos de construcción sofisticados
    - Heurísiticas
  - Escenas estáticas versus dinámicas:
    - Reconstrucción
    - Eliminación + inserción
- Aplicaciones
  - Ray tracing
  - Frustum culling
  - Selección
  - Colisiones

Quadtrees y Octrees

- Estructuras de datos jerárquicas útiles para dividir el espacio y optimizar operaciones
  - Detección de colisiones, determinación de la visibilidad y el renderizado.

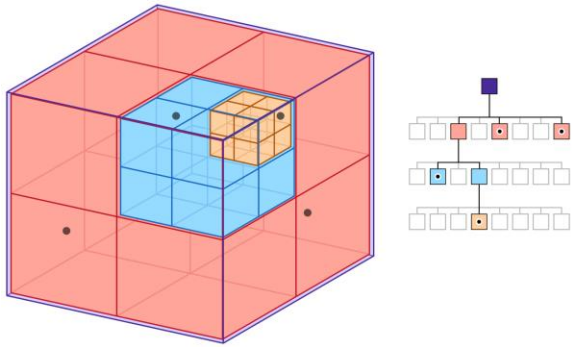
- Un **quadtree** es una estructura de datos en forma de árbol en la que cada nodo interno tiene exactamente **cuatro** hijos.
  - La idea es particionar un espacio bidimensional
    - Subdividiéndolo recursivamente en **cuatro** cuadrantes o regiones.



Fuente: https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/PRquadtree.html

Octree

- Un **octree** es una estructura de datos en forma de árbol
  - Cada nodo interno tiene exactamente **ocho** hijos.
  - Se utiliza para particionar un espacio tridimensional
    - Subdividiéndolo recursivamente en **ocho** octantes.
- Consulta eficiente de datos espaciales en el espacio 3D
  - Búsqueda de todos los objetos dentro de un determinado volumen.

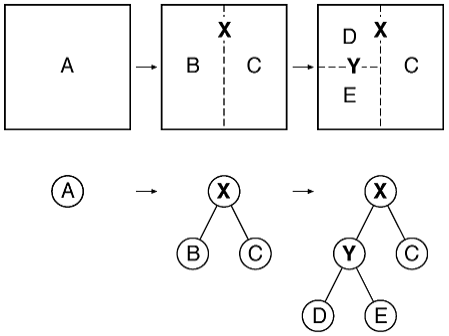


Fuente: https://learnopengl.com/Guest-Articles/2021/Scene/Frustum-Culling

- Optimiza el renderizado
  - Determinando – con poco coste - qué objetos se encuentran dentro del marco de visión.
- Determina rápidamente posibles colisiones en el espacio 3D
  - Comprobando los objetos que se encuentran en la misma región o en regiones vecinas.

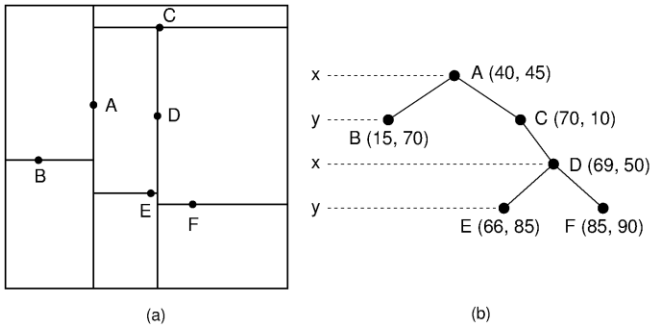
Binary Space Partitioning (BSP)

- Un árbol BSP (Binary Space Partitioning) es una estructura de datos que representa una subdivisión jerárquica recursiva del espacio n-dimensional en subespacios convexos.
  - Utilizado para ordenar y buscar estructuras en un espacio n-dimensional.
    - Aunque generamente se utilizan en las 2D y 3D
  - En su conjunto representa todo el espacio, y cada nodo del árbol representa un subespacio convexo.
  - Cada nodo almacena un *hiperplano* que divide el espacio que representa en dos mitades, y referencias a dos nodos que representan cada mitad.



https://people.eecs.berkeley.edu/~jrs/274s19/bsptreefaq.html#6.txt

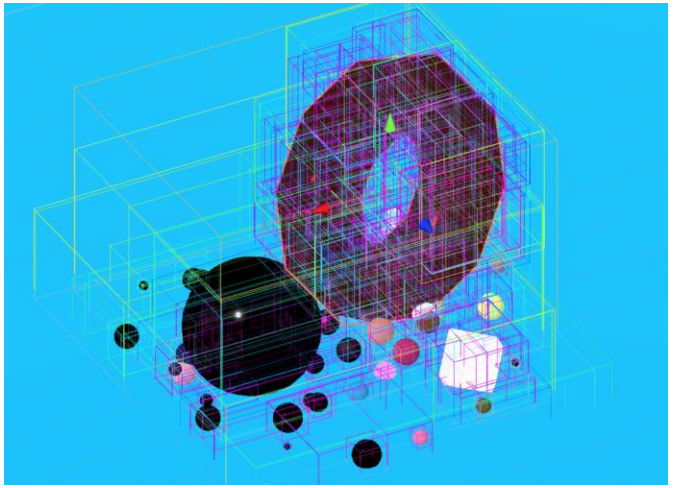
- Un **kD-Tree** es una EEDD binaria arbórea
  - Se utiliza para organizar puntos en un espacio k-dimensional.
  - Útil para la búsqueda del vecino más próximo, la búsqueda de rangos y otras consultas de búsqueda multidimensional.
  - Modificación del árbol BST (Binary Search Tree)
    - Permite procesar eficazmente claves de búsqueda multidimensionales.
    - Difiere del BST en que cada nivel del árbol kd toma decisiones de ramificación basadas en una clave de búsqueda concreta asociada a ese nivel, denominada **discriminador**.
  - El ejemplo muestra dos discriminadores
    - x para la separación vertical
    - y para la separación horizontal



https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/KDtree.html

Bounding Volume Hierarchies (BVH)

- EEDD arbórea utilizada para acelerar el proceso de renderizado y detección de colisiones.
- Cada nodo del árbol BVH representa un volumen delimitador
  - Engloba un conjunto de primitivas geométricas.
- El árbol es jerárquico
  - Cada nodo puede tener nodos hijos
  - El volumen delimitador de un nodo *padre* contiene los volúmenes delimitadores de sus hijos.
- La idea es eliminar rápidamente grandes porciones de la escena
  - No necesitan ser comprobadas para detector colisiones o intersecciones de rayos
  - Mejora el rendimiento.
  - Muy útil en escenas dinámicas



Fuente: https://smashriot.com/path-tracing/