Optimización de la eficiencia

Visualización II

Real time?

GPUs más potentes

	GTX 295	GF100	HD5870	HD5970
Processing units	480 scalar	512 scalar	320 vec5	640vec5
Texturing units	160 vec4	64 vec4	80 vec4	160 vec4
GPU clock (MHZ)	576	725	850	725
Triangles rate (Mtriangles/s)	1152	2800	850	725
Memmory bandwidth (GB/s)	208.6	214.7	143.1	238.4

Uso de la GPU real

- Múltiples iteraciones pre-render/post-render
  - Frame rate real>>20 fps
- Uso de GPU para otros cálculos (por ej. Motor físico)
- +Efectos +Complejidad de mallas
- **Mucho procesamiento en 2D**

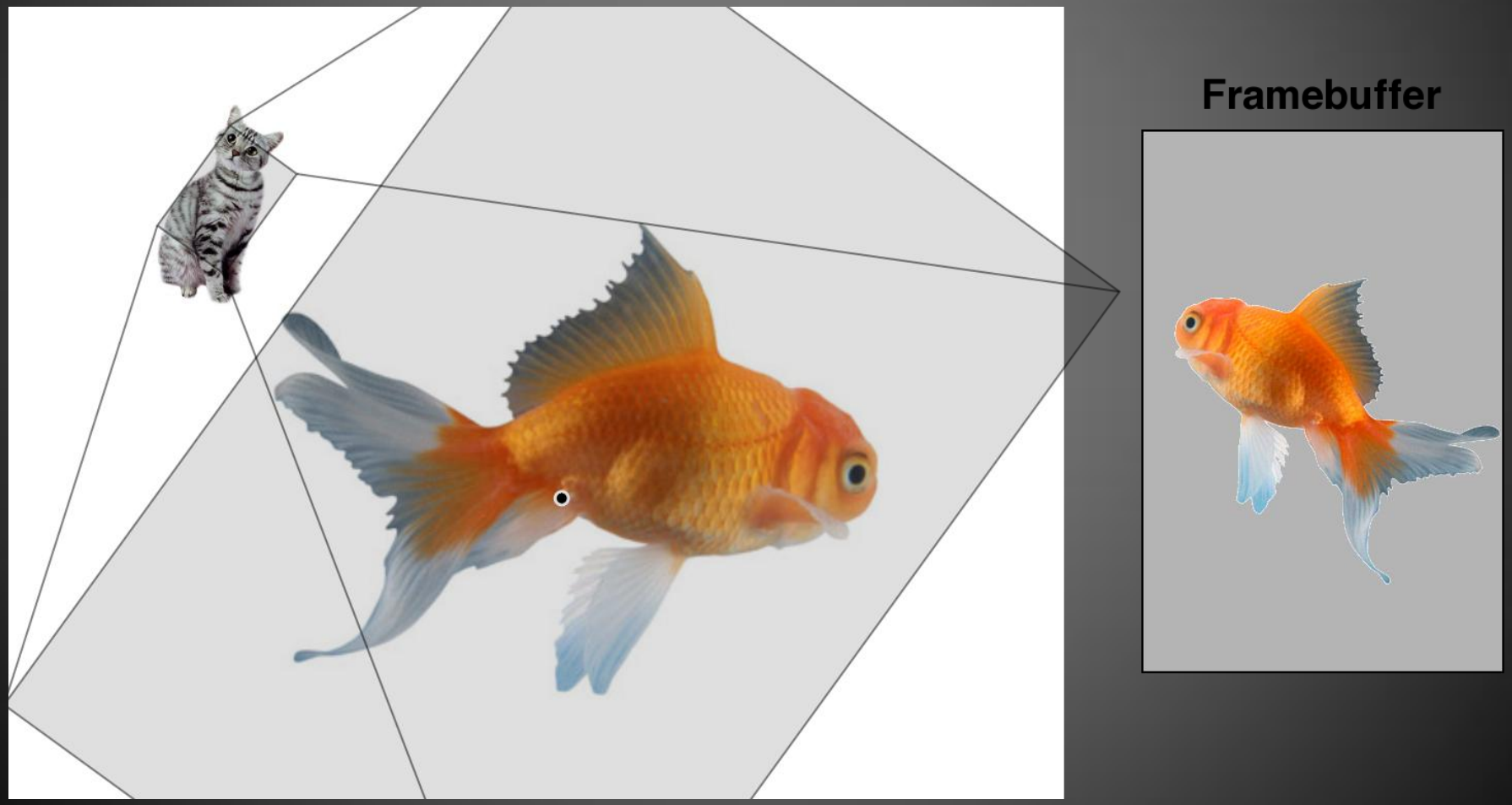
Contenido

- CPU
  - Selección de objetos visibles(Frustum)
  - División del espacio
    - N-Trees, Grillas, Planos
    - Bounding-Volumes
  - LODS
  - Otros métodos
  - Clonado
  - Reemplazo por textura
- GPU
  - Estructuras y funciones de OpenGL
  - Administración de la GPU

Principio general de Simplificación del Dominio

- Conjuntos de datos pueden ser muy complejos para representar en tiempo real
- Se necesita reducir la cantidad de triángulos a renderizar en la escena
- **Soluciones**
  - **Seleccionar elementos visibles**
  - **Simplificar los objetos**

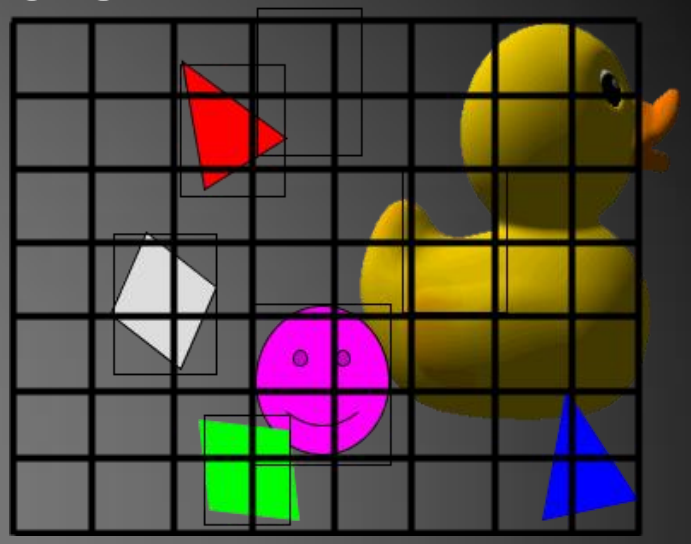
FRUSTUM de Vista



Por FRUSTUM de Vista

- Los elementos que nos interesan se deben encontrar dentro de este volumen
- Los chequeos de inclusión son productos escalares
  - // El *frustum* se forma por 6 planos (es un prisma deformado)
  - // *P* es el centro del objeto , *radius* es el eje más largo
  - For j =0; j<6
  - if dot(P , Plane[i])>radius//se encuentra fuera del Frustum
    - Return false
- Los planos deben generarse correctamente

Métodos de clasificación



- Celdas regulares
  - Consulta = O(n/c)
  - Construcción= O(n)
- Quadtree / Octree / Bsp
  - Asignan *triángulos* u *objetos* de toda la escena en cada celda
  - Consulta = O(log n)
  - Construcción= O(n log n)

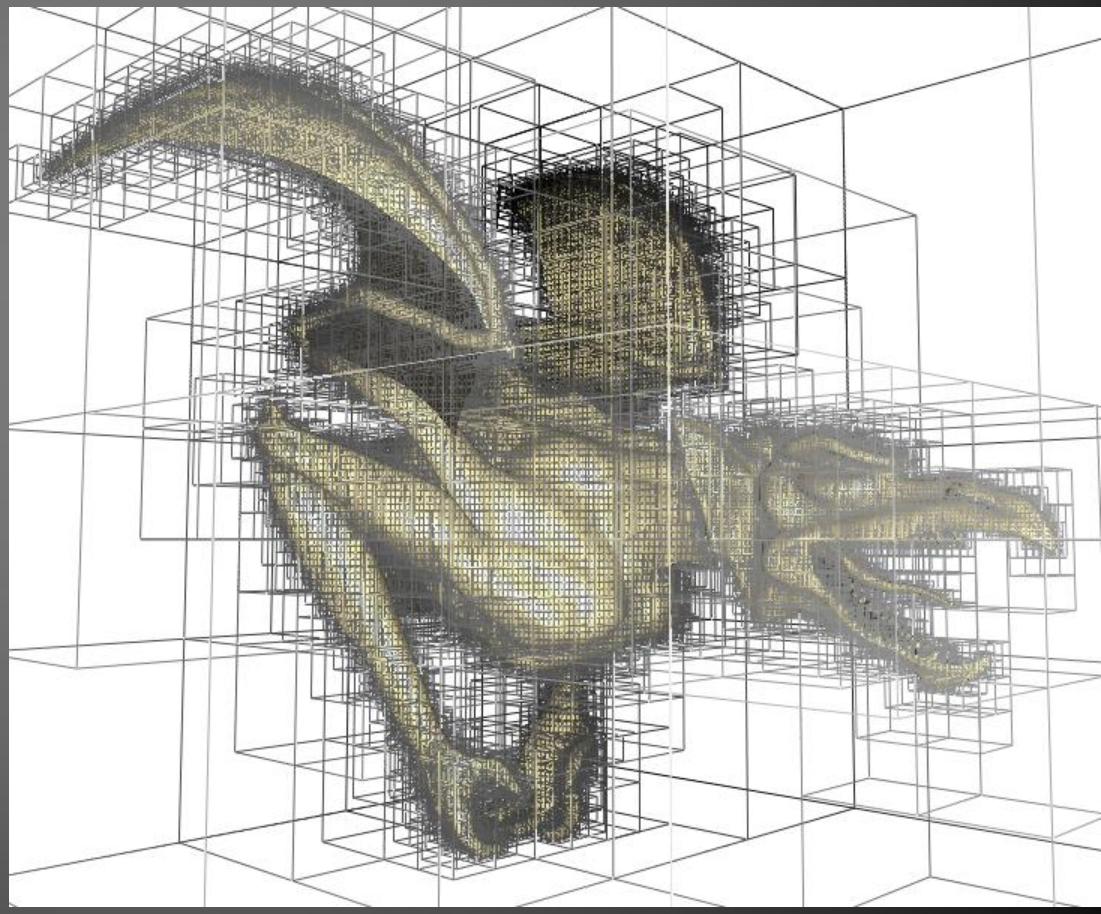
Métodos de clasificación

+render eficiente

+colisiones

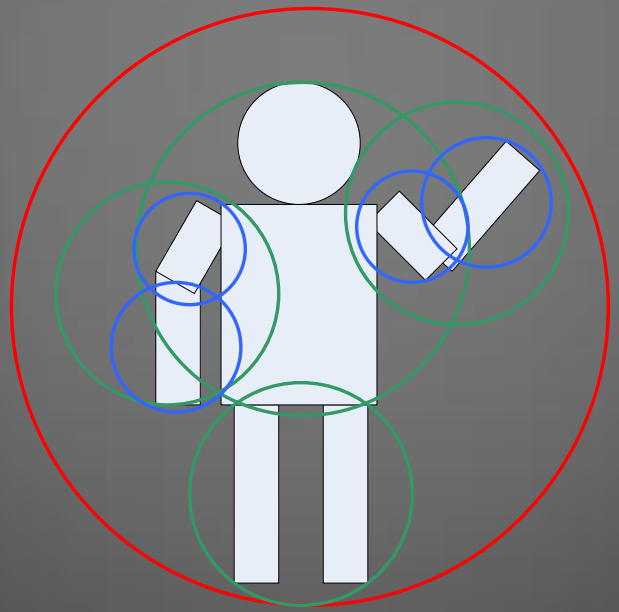
-complejidad de rendering

-objetos animados



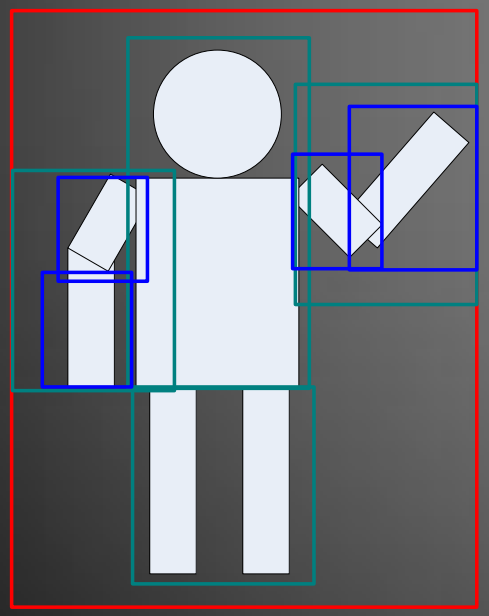
Bounding Volumes

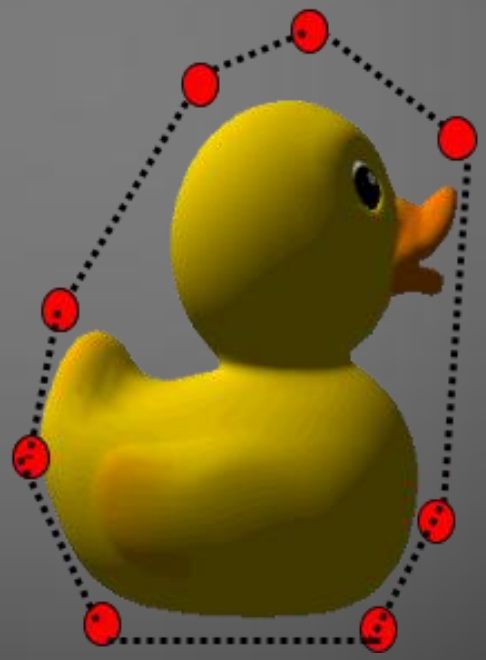
- Se usan representaciones matemáticas simples(Esferas, cubos, prismas (boxes))
- Requiere pocos datos :
  - Posiciones con respecto al centro del objeto y dimensiones de los volúmenes



BV– Boxes / ConvexHull

- *Bounding boxes* orientados (OBB) son más adecuados al objeto pero más costosos para hacer los cálculos:
- *Convex Hull* > Precisión > Primitivas



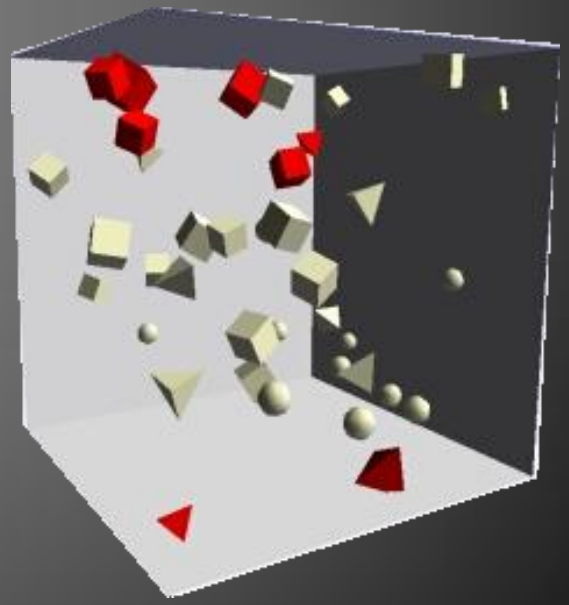


Bounding Objects

- Ventajas:
  - Cálculos simples
  - Requieren poco espacio en memoria (para la esfera es una posicion y un radio)
- Desventajas
  - No tienen buena precisión
  - No reducen la cantidad de polígonos a representar

Resolver colisiones x Objeto

- Se distribuyen los objetos utilizando grillas regulares.
- Se comparan los objetos con no más de una celda de distancia:
  - for all objects O
    - for y = O.grid_y-1 to O.grid_y+1
    - for x = O.grid_x-1 to O.grid_x+1
    - for all objects O’ in grid(x,y)
      - check collision O <-> O’



La idea

- El problema:
  - **Conjuntos de datos pueden ser muy complejos para representar en tiempo real**
- Una solución:
  - **Usar geometría mas simples de modelos pequeños o distantes.**
  - **Eso es lo conocido como *Level of Detail* or *LOD***

La idea

- Crear *levels of detail* (*LODs*) de los objetos:

LOD Discreto

- Los LODs los crea un diseñador 3D
- En tiempo de ejecución se elije el LOD adecuado de acuerdo a la distancia (o por otro criterio)

Pros

- Ventajas
  - Granularidad adecuada:
    - Utiliza los polígonos más representativos
    - La fidelidad es la más elevada
  - La mejor aproximación para objetos muy detallados
    - Permite simplificarlos drásticamente
- Desventajas
  - Es costoso en tiempo real

Usar LODs (o no)

- Combinar con otras técnicas (BSP, Octree)
- Cuando usar esta técnica en la escena, y con qué objetos?
  - LOD discretos : objetos muy replicados y con un número razonable de polígonos
  - LOD continuo : Mapas, grillas de agua (espacios navegables y uniformes)

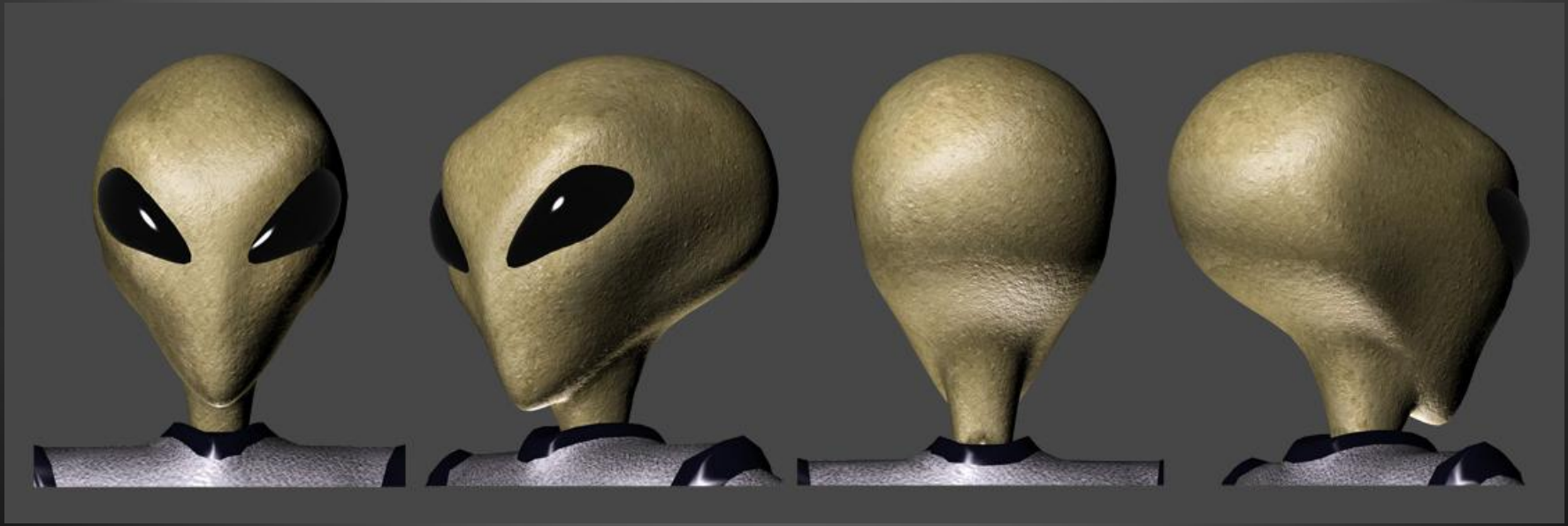
Donde mas se usa LOD?

Resumen

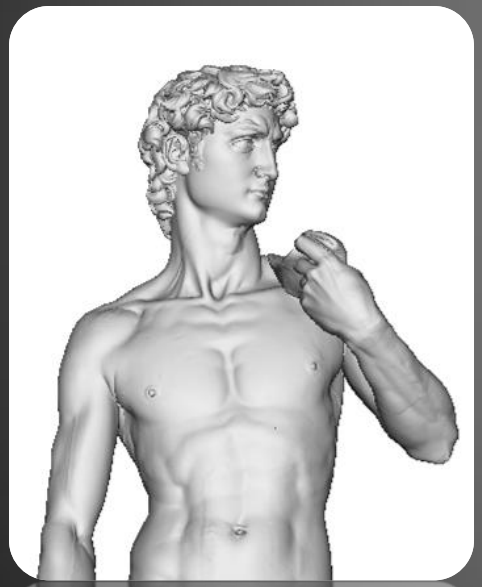
	Técnicas	Tiempo	Pros	Cons
LOD	▪**Discreto** ▪**Continuo** ▪**Jerárquico**	▪Generación ▪***O(n2)*** ▪Consulta ▪***O(1)***	▪**En general, la visualización no depende de la técnica**	▪**No existe una técnica general** ▪**No es viable cuando los objetos se deforman**
Space Partition	▪**Grillas** ▪**Quadtree/Octree** ▪**BSP**	▪Generación ▪***O(n)*** ▪Consulta ▪***O(log n)***	▪**Genera una buena distribución de los elementos** ▪**El costo de búsquedas es de *ln #Elementos***	▪**Mayor complejidad para visualizar** ▪**Las modificaciones tienen un costo elevado**
Bound	▪**Esferas** ▪**AABB** ▪**OBB**	▪Generación ▪***O(n)*** ▪Consulta ▪***O(1)***	▪**Sencilla** ▪**Conveniente para objetos deformables** ▪**Eficiente**	▪**No son dependientes de la visualización** ▪**Son malas aproximaciones**

Simplificación por apariencia

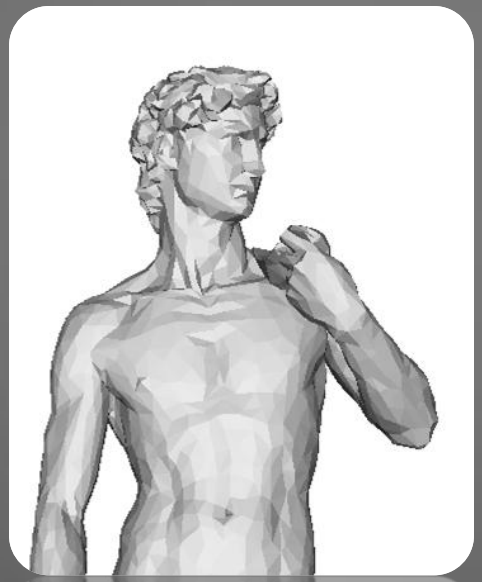
- Utilizar texturas en lugar de triángulos
- Se aplican mapas de normales (bumpmaps)



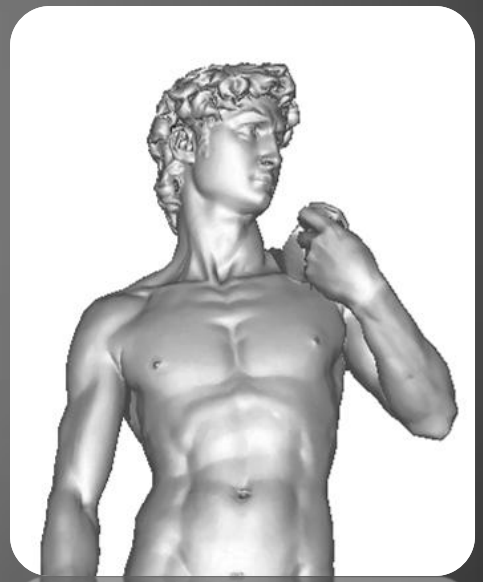
Simplificación por apariencia



1683 k Triangulos



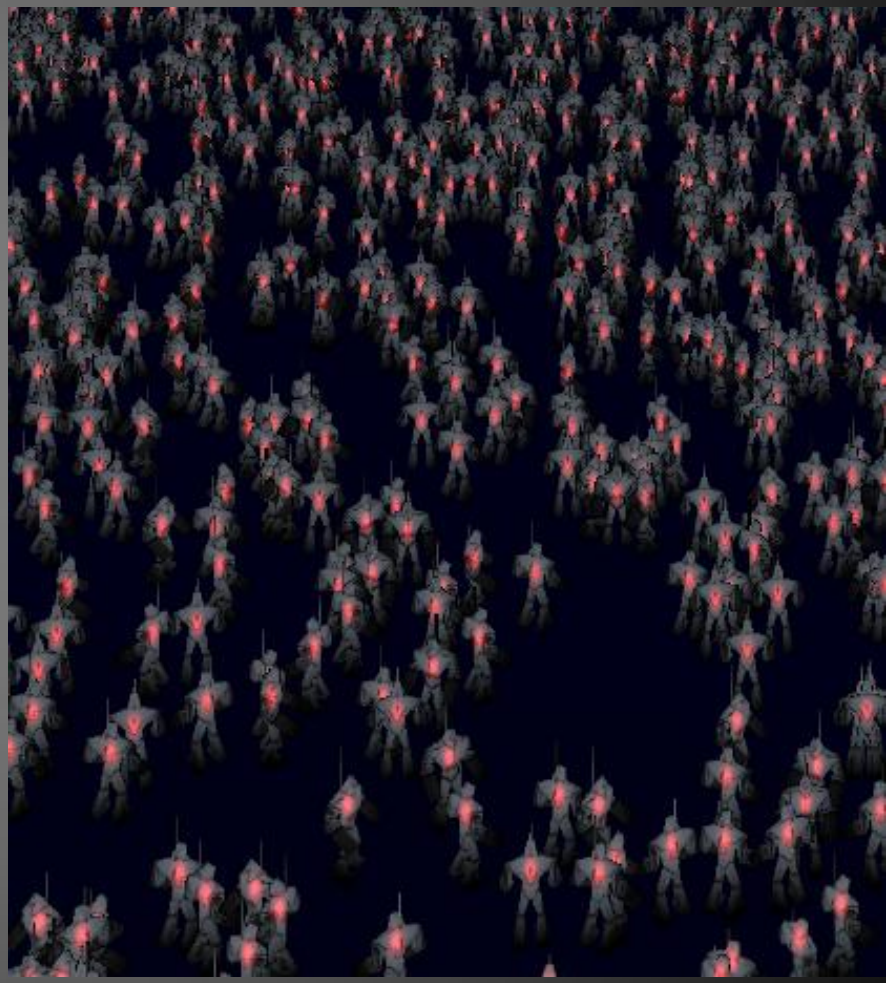
10 k Triangulos



10 k Triangulos + Shading

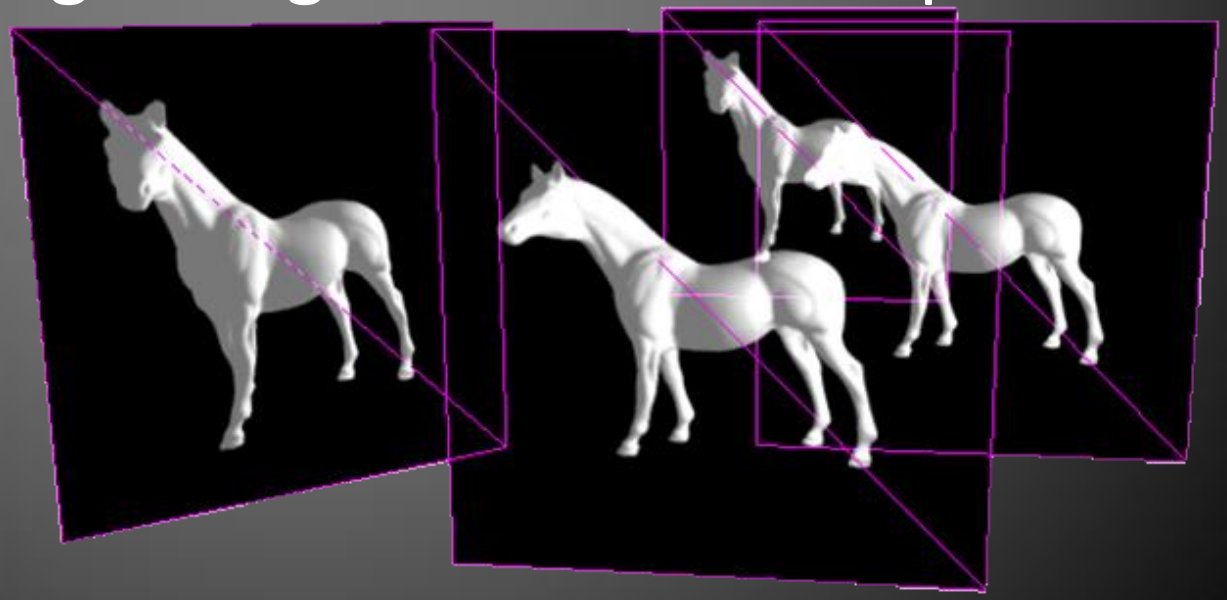
Clonado de objetos

- Varias instancias del mismo objeto
  - cambian las transformaciones o el color
- Por cada nuevo objeto
  - Se hace referencia a la misma estructura de la geometría (vértices, triángulos, texturas)
  - Se crean las transformaciones propias



Pre-rendered on texture

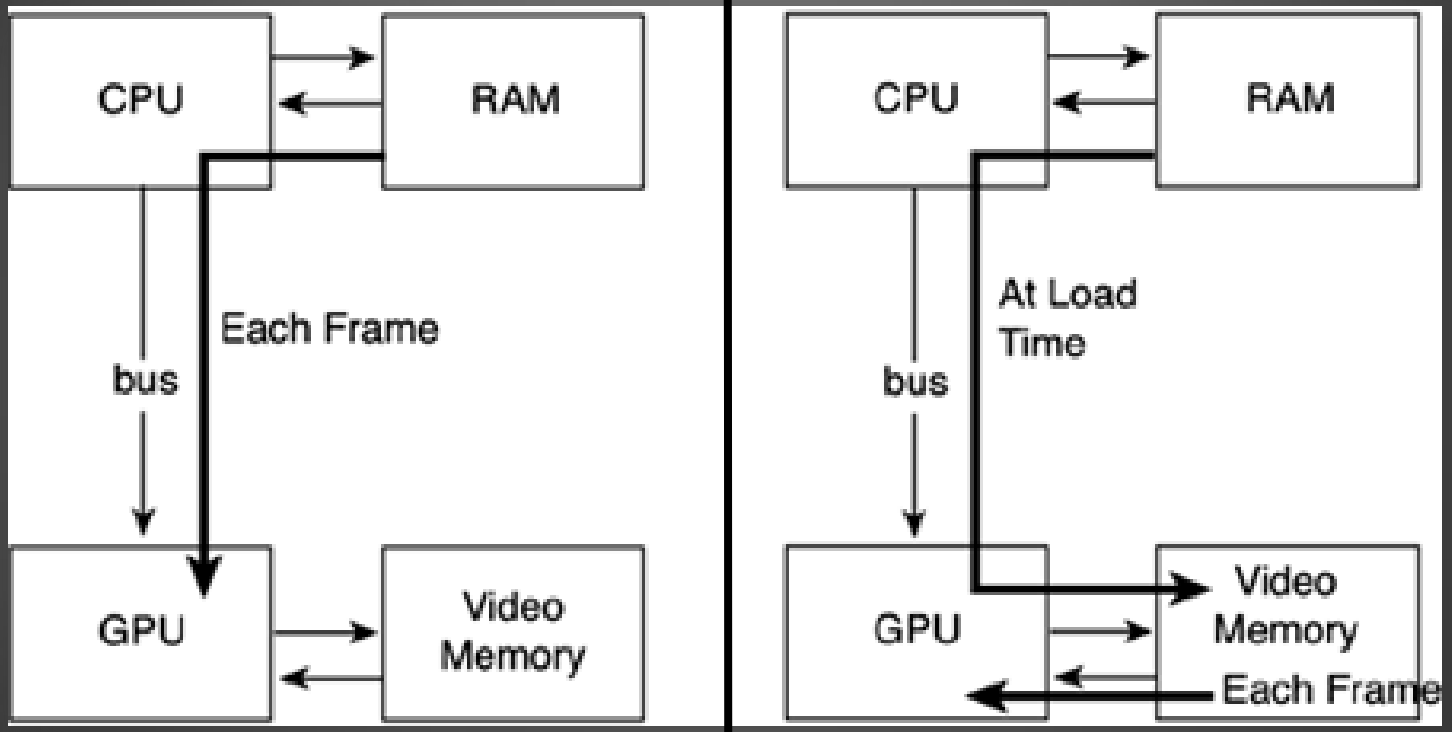
- Renderizar (offline/offscreen) el objeto a textura
- Útil para objetos estáticos (*impostors*)
- Costo Rendering->Polígono + textura mapeada



Que provee OpenGL ?

- OpenGL provee estructuras de datos eficientes, métodos de control y manejo de búffer
  - Vertex Buffer Objects
  - Frame Buffer en GPU
  - Culling Queries
  - Renderizar los buffer a texturas

Estructuras GPU



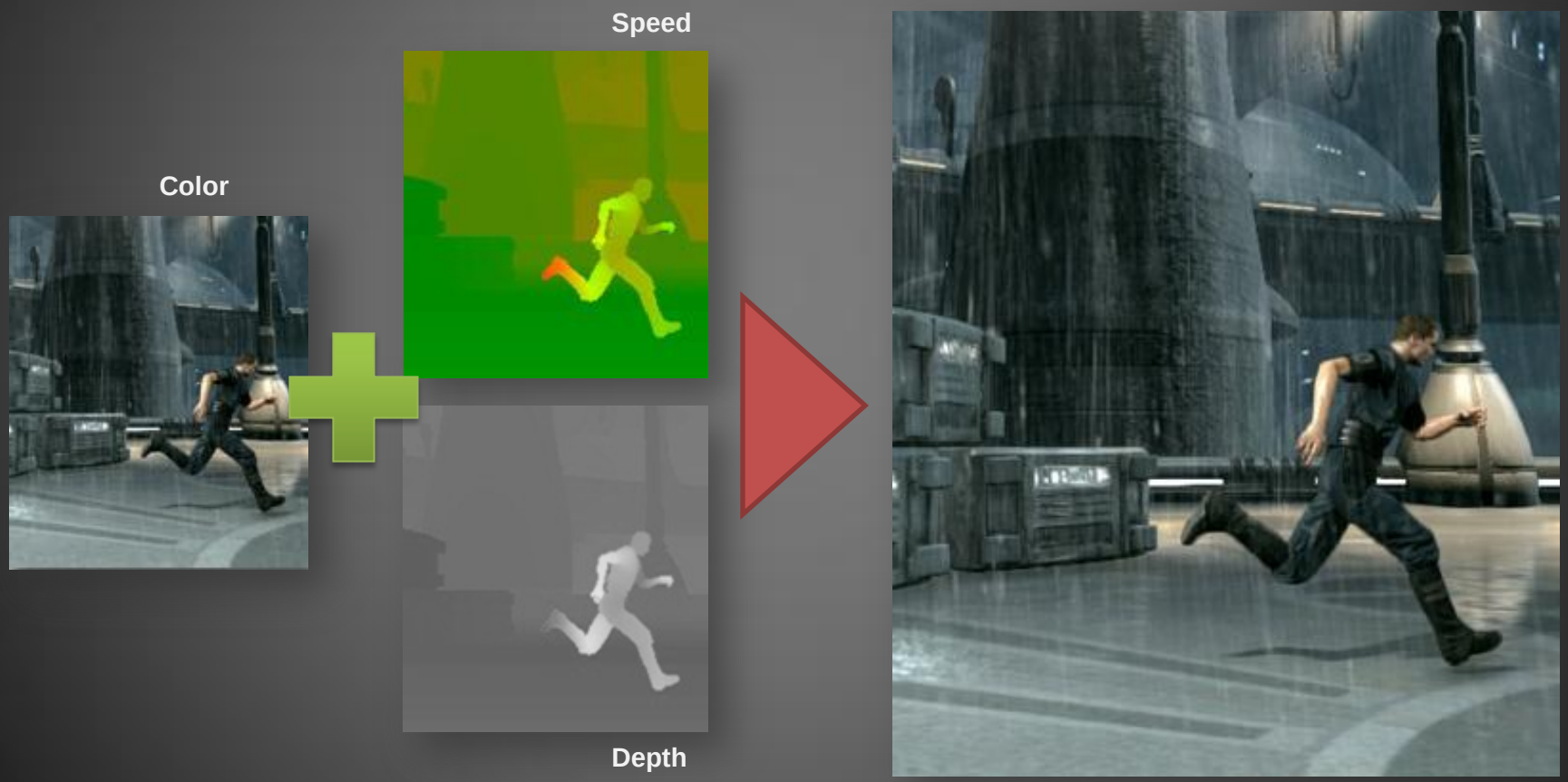
Vertex arrays(CPU)

VBO(GPU)

Culling Queries

- Verificar que un objeto *O* será dibujado en pantalla
  - Se dibuja una versión simple del objeto (por ej.bound, convex hull)
  - Se leen cuantos píxeles afectaron del Buffer
    - Si la cantidad de pixeles > cota
    - IncludeToRender(O)
  - Se renderiza la escena final

Render->Buffer



Efectos ?

Efectos en Pantalla

Efectos en Pantalla

Efectos x Objeto

Efectos x Objeto

Administrar la GPU

- Realizar efectos en GPU requiere *múltiples pasadas* renderizando a distintos *buffers*
  - *depth_map*(Blur-motion / Ambient Occlusion / Depth of Field / Shadow Map)
  - *Frame buffer*(Edge detection / HDR / Reflections / Cubemap Reflections)

Administrar la GPU

- Administrar los efectos visuales.
  - No aplicar a objetos *lejanos*
  - Filtrar objetos por *Frustum* (un *frustum* x cámara)
- Calcular depthBuffer
  - Desactivar el *color_buffer* y renderizar solo la malla para obtener el *depth_buffer*
- Sombras :
  - Aplicar siempre LOD

Administrar la GPU

- Resolución de los búffer y texturas

  > Resolución >Demora

  Se utiliza generalmente

  - **Sombras = 2048x2048**
  - **Depth Buffer = 1024 x 1024**
  - **Color Buffer = 1024 x 1024**
  - **Cubemaps = 512 x 512**
  - **Texturas = 256 x 256**

Links

- Tutorial Frustum
  - http://www.lighthouse3d.com/opengl/tutorials/
- http://graphics.cs.cmu.edu/projects
- http://developer.download.nvidia.com/whitepapers/2007/SDK10/RainSDKWhitePaper.pdf/