7 Efectos avanzados

Múltiples pasos de dibujo

Algoritmos de varias pasadas

Texturas múltiples

Ejemplos de texturas múltiples

Múltiples pasos de dibujo

- Dibujar en múltiples pasos (*multi-pass rendering*)
  - Técnica que consiste en construir la imagen final a partir de la combinación de un conjunto de pasos de *rendering*
  - La tubería gráfica dibuja un triángulo cada vez, de manera que sólo se dispones de información local
  - Muy popular en los juegos debido la rapidez de aplicación utilizando el hardware gráfico

- Funcionamiento:
  - Utilizar buffer auxiliares para guardar información
  - Aplicar operaciones lógicas con los valores de los *buffers*
  - Ejemplo: espejos, volúmenes de sombreado, antialiasing, composición

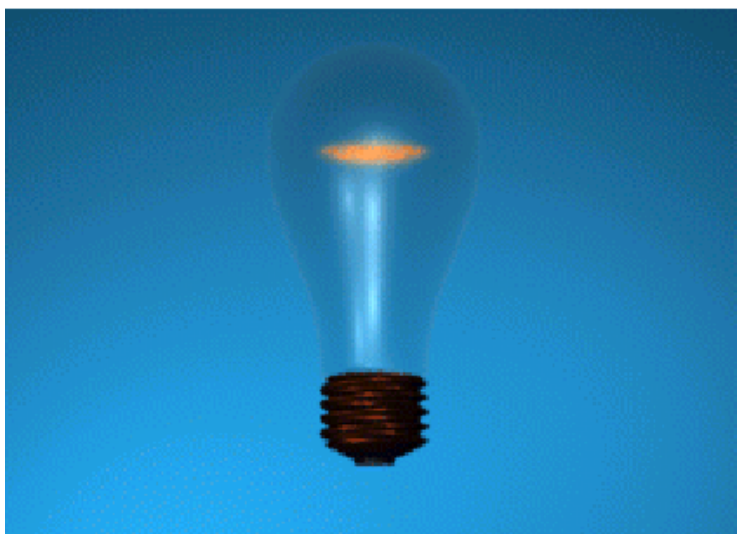
Múltiples pasos de dibujo

- Los *buffers*
  - Permite almacenar información global de la escena a visualizar
  - Son como un espacio de trabajo aparte o zonas de memoria extra
  - Se definen por:
    - Los tipos de valores que almacenan
    - Las operaciones que permiten
    - La manera en que se acceden (lectura y escritura)

- *Buffers* en OpenGL
  - **Color (*color buffer*).** Almacenan el color RGBA para cada píxel:
    - *front/back* (doble *buffer*) , *left/right* (estéreo) y otros auxiliares)
  - **Profundidad (*depth buffer*).** Almacena la información de profundidad para cada píxel
  - **Acumulación (*Accumulation buffer*).** Como el *buffer* de color, pero con más resolución y diferentes operaciones
  - **Plantilla (*stencil buffer*).** Almacena un número de bits para cada píxel

Múltiples pasos de dibujo

- Los fragmentos
  - Un fragmento es un trozo de polígono que ocupa un píxel, con color e información de profundidad, después de iluminado (píxel shader) y/o texturado
  - Las operaciones realizadas con fragmentos sobre el buffer de color son esenciales para comprender las técnicas de múltiples pasadas:
    - *alpha test, blending, depth test* y *stencil test*



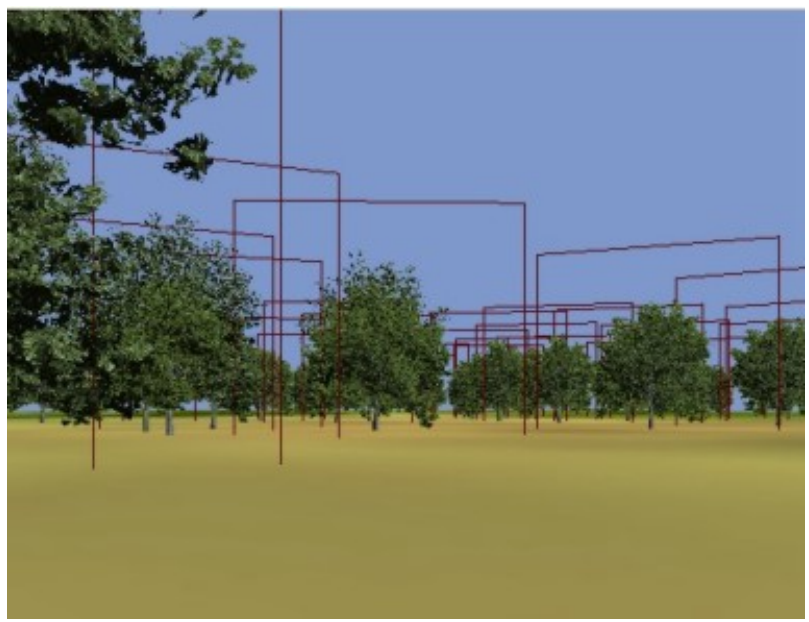
Múltiples pasos de dibujo

- Transparencias
  - El *alpha test* permite a un fragmento pasar o no dependiendo de:

si ($\alpha_{fragmento}$ op $\alpha_{referencia}$)
pasa el fragmento

  - Donde, $\alpha_{fragmento}$ es el valor de alfa de fragmento y $\alpha_{referencia}$ es el valor especificado
  - op puede ser: <, <=, =, !=, >, >=, always o never

  - Ej. Billboards
    - Polígonos texturados que suelen representar árboles
    - El valor de transparencia vale 1 donde esta el árbol y 0 donde no esta



Múltiples pasos de dibujo

- Combinación de imágenes (*blending*)
  - Permite mezclar la imagen actual con la del *frame buffer*
  - *Ecuación:*

$$(R,G,B,\alpha)_{new} = (R,G,B,\alpha)_{source} * blend_{source} + (R,G,B,\alpha)_{dest} * blend_{dest}$$

  - Ejemplos de factores de combinación en juegos

Aplicación	**$blend_{source}$**	**$blend_{dest}$**
Transparencias	$(\alpha,\alpha,\alpha,\alpha)_{source}$	$(1-\alpha,1-\alpha,1-\alpha,1-\alpha)_{source}$
Halos y explosiones	(1,1,1,1)	(1,1,1,1)
Sombras y marcas	(0,0,0,0)	$(1-\alpha,1-\alpha,1-\alpha,1-\alpha)_{source}$
Mapas de iluminación	(0,0,0,0)	$(\alpha,\alpha,\alpha,\alpha)_{source}$
Mapas de niebla	$(1-\alpha,1-\alpha,1-\alpha,1-\alpha)_{source}$	$(1-\alpha,1-\alpha,1-\alpha,1-\alpha)_{source}$

Múltiples pasos de dibujo

- Z-buffer
  - El test de profundidad compara la profundidad del fragmento actual con la del Z-buffer
  - Posibles comparaciones: Always, Never, <, <=, =, !=, >, >=
- El buffer de acumulación
  - Es un lugar para almacenar y realizar cálculos con los píxeles (RGBA)
  - No esta disponible para escribir directamente
  - Operaciones
    - Cargar el buffer de color en el buffer de acumulación
    - Acumular el contenido del buffer de color (multiplica los valores de un buffer por un valor y sumar el resultado a los valores del buffer de acumulación
    - Sumar o multiplicar todos los píxel por una constante
    - Devolver los valores al buffer de color (escalados por una valor)
  - Lento para aplicaciones interactivas
  - Ej: antialiasing, motion blur, depth of field

Múltiples pasos de dibujo

- *Stencil Buffer*
  - Zona de memoria del mismo tamaño que la imagen, se utiliza como una plantilla que deja pasar sólo algunos fragmentos (extensión del Z-buffer)
  - Almacena varios bits para cada píxel (se tiene control sobre el número de bits)
  - Es necesario especificar
    - **El test** para decidir que fragmentos pasan
    - **La acción** a realizar sobre el buffer según el resultado del test
  - Todos los tests y operaciones se realizan sobre el valor del *stencil* que se corresponde con la localización del píxel del fragmento
  - Ej: en la primera pasada se fijan valores en el *stencil* que indican que zonas de la imagen van a dibujarse en la segunda pasada

Múltiples pasos de dibujo

- *Stencil test*
  - Antes del test de profundidad
  - Resultados posibles
    - El test del *stencil* falla
    - El test del *stencil* pasa y el test de profundidad falla
    - Ambos tests pasan
  - Condiciones del test
    - *Always*. El frag. siempre pasa
    - *Never*. El frag. nunca pasa
    - Pasa si (ref. & masc.) op (stencil & masc.)
      - **op** es: <, <=, =, !=, >, >=
      - **ref.** valor para el test
      - **masc.** selecciona los planos de bits
- Acción sobre el *stencil*
  - Mantener el valor actual
  - Actualizar con el valor = 0
  - Reemplazar el stencil con el valor de referencia
  - Incrementar el stencil
  - Decrementar el stencil
  - Invertir el stencil

Algoritmos de varias pasadas

- Transparencia
  - Es necesario ordenar los polígonos transparentes en orden (de atrás hacia delante)
  - Operación de *blending*

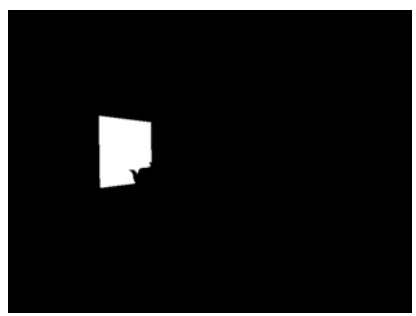
$$C = t\ C_f + (1-t)\ C_b$$

    - Donde t es el factor de transparencia y $Z_f < Z_b$
  - Algoritmo
    1) Visualizar los objetos opacos normalmente (activando Z-buffer)
    2) Ordenar los polígonos transparentes
    3) Visualizar los polígonos transparentes en orden (con *blending* y Z-buffer)
  - Existe otra solución sin necesidad de ordenar los polígonos que utiliza dos Z-buffers

Algoritmos de varias pasadas

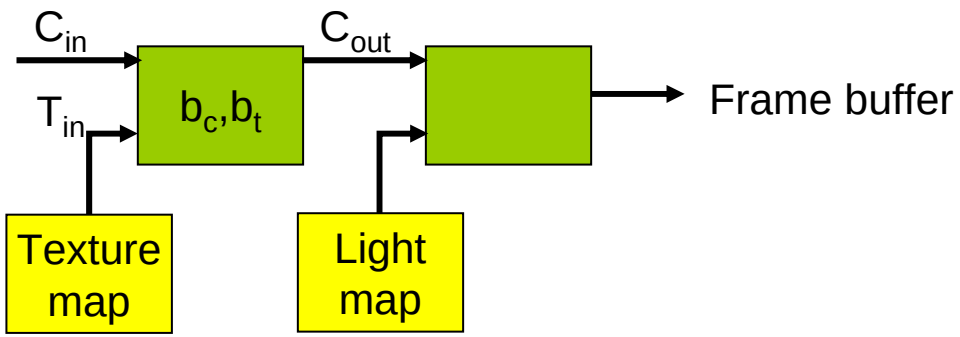
- Reflexiones
  - Dos pasadas: una para la escena y otra para el reflejo
  - La escena reflejada se evalúa:
    - Reflejando los objetos sobre el plano del espejo
    - Reflejando la cámara

  - Algoritmo
    1) Dibujar la escena excluyendo el espejo (*stencil* test des-habilitado, Z-buffer habilitado)
    2) Inicializar *stencil* y Z-buffer. Dibujar el polígono espejo en el Z-buffer y (cuando pasa el Z-buffer) en el *stencil*
    3) Dibujar la espejo en el Z-buffer utilizando el stencil para inicializar a la máxima profundidad
    4) Dibujar la escena reflejada utilizando el stencil test.



Texturas múltiples

- Características
  - Combinación de texturas utilizando el Hardware Gráfico
  - Aunque se pueden obtener mediante algoritmos de varias pasadas
  - Ejemplo:



  - La ecuación general de *blending* para múltiples texturas

$$C_{out} = b_c\ C_{in} + b_t\ T_{in}$$

    - Donde b_c y b_t son los factores de blending

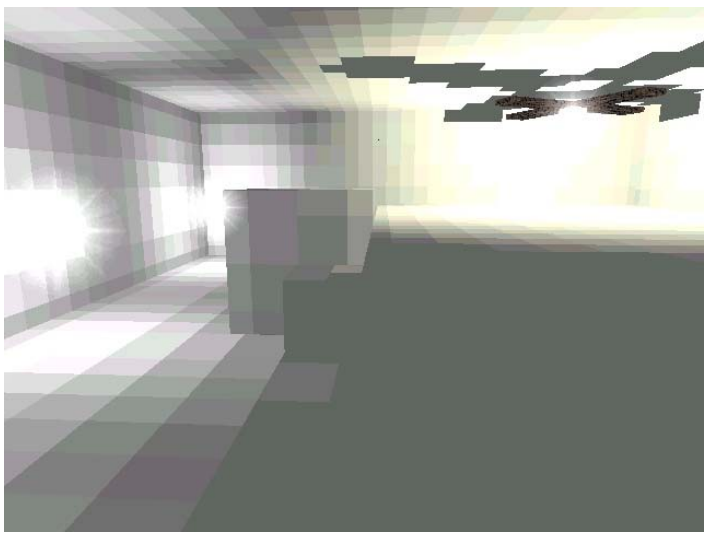
Texturas múltiples

- Comparación de técnicas

Múltiples pasadas	Múltiples texturas
Seleccionar (textura) Dibujar (objeto) Seleccionar (Light Map) Dibujar objeto	Seleccionar (textura, lightmap) Dibujar (objeto)

Ejemplos de texturas múltiples

- Iluminación en juegos
  - La iluminación estática puede precalcularse en una textura
  - El hardware mezcla las texturas: ligth map, texture map, fog map, etc.



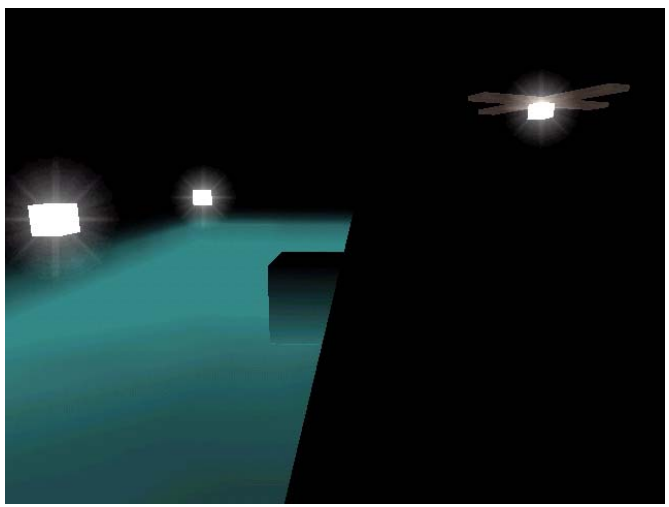
Light map

Light map + filtro HW

Ejemplos de texturas múltiples

Light map

Texture map



Fog map

Final

Ejemplos de texturas múltiples

- Construcción del Light Map
  - Encontrar conjuntos de polígonos coplanares e interconectados
  - Encontrar AABB para ese conjunto
  - Encontrar la cara con mayor extensión y utilizarla como light map del conjunto
  - Marcar cada polígono con un índice de light map y continuar hasta que todas las caras tengan light map