



UNIVERSITAT POLITÈCNICA DE CATALUNYA  
BARCELONATECH

Departament d'Arquitectura de Computadors

# Tarjetas Gráficas y Aceleradores

## Texturas y Antialiasing

### Agustín Fernández

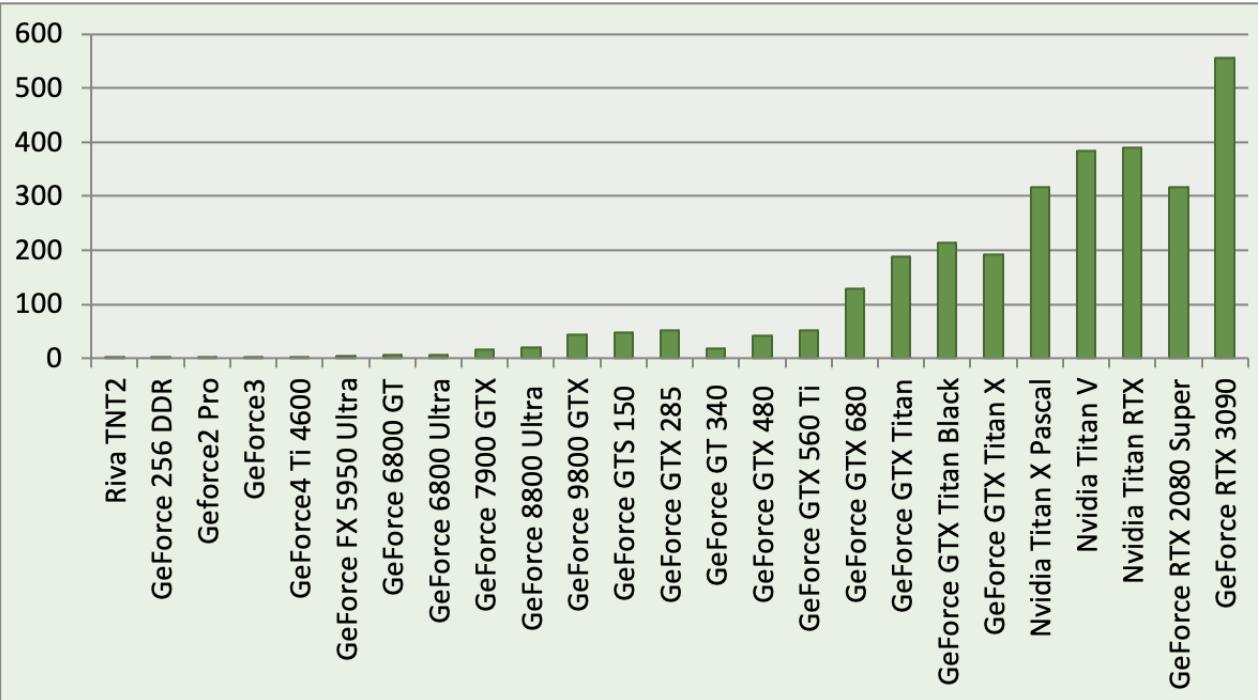
Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya



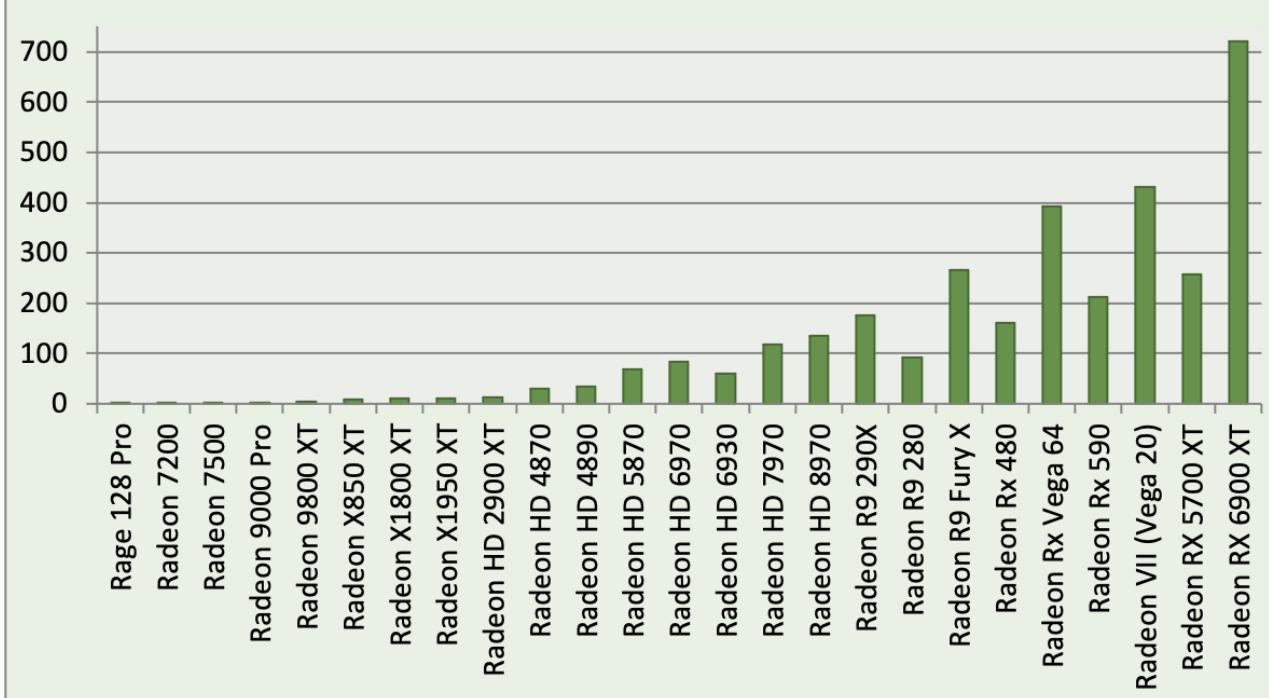
# nVIDIA: Gtex/s



- Texel ≈ Textura
- Pixel ≈ Frame Buffer
- GTex/s: Giga accesos a texturas por segundo

| NVIDIA                  | Año  | TU  | Mtex/s  |
|-------------------------|------|-----|---------|
| Riva TNT2               | 1999 | 2   | 250     |
| GeForce 256 DDR         | 2000 | 4   | 480     |
| Geforce2 Pro            | 2000 | 8   | 1.600   |
| GeForce3                | 2001 | 8   | 1.600   |
| GeForce4 Ti 4600        | 2002 | 8   | 2.400   |
| GeForce FX 5950 Ultra   | 2003 | 8   | 3.800   |
| GeForce 6800 GT         | 2004 | 16  | 5.600   |
| GeForce 6800 Ultra      | 2005 | 16  | 6.400   |
| GeForce 7900 GTX        | 2006 | 24  | 15.600  |
| GeForce 8800 Ultra      | 2007 | 32  | 19.584  |
| GeForce 9800 GTX        | 2008 | 64  | 43.200  |
| GeForce GTS 150         | 2009 | 64  | 47.232  |
| GeForce GTX 285         | 2009 | 80  | 51.840  |
| GeForce GT 340          | 2010 | 32  | 17.600  |
| GeForce GTX 480         | 2010 | 60  | 42.000  |
| GeForce GTX 560 Ti      | 2011 | 60  | 52.610  |
| GeForce GTX 680         | 2012 | 128 | 128.800 |
| GeForce GTX Titan       | 2013 | 224 | 187.500 |
| GeForce GTX Titan Black | 2014 | 240 | 213.400 |
| GeForce GTX Titan X     | 2015 | 192 | 192.000 |
| Nvidia Titan X Pascal   | 2016 | 224 | 317.400 |
| Nvidia Titan V          | 2017 | 320 | 384.000 |
| Nvidia Titan RTX        | 2018 | 288 | 388.800 |
| GeForce RTX 2080 Super  | 2019 | 192 | 316.800 |
| GeForce RTX 3090        | 2020 | 328 | 556.000 |

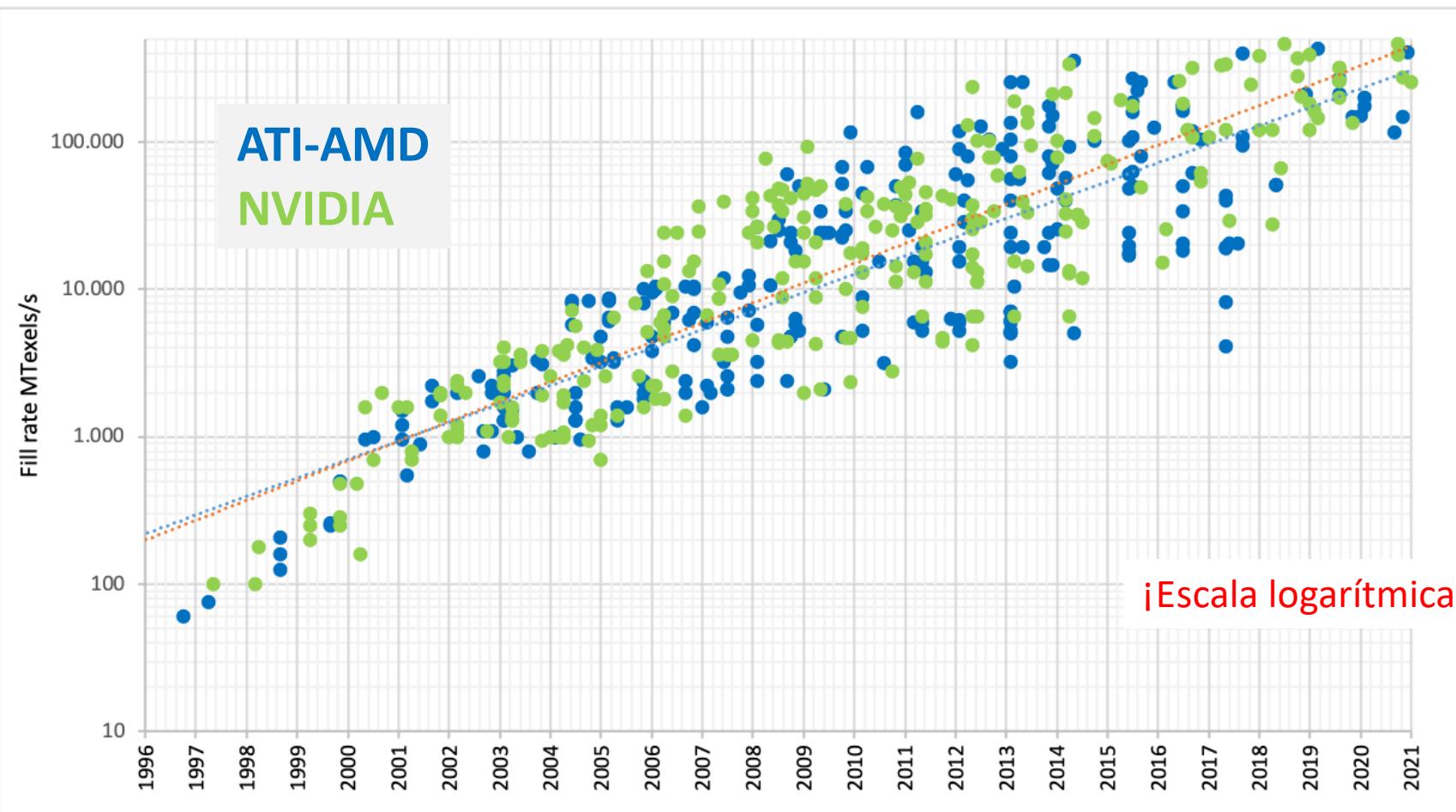
# AMD: Gtex/s



- Texel ≈ Textura
- Pixel ≈ Frame Buffer
- GTex/s: Giga accesos a texturas por segundo

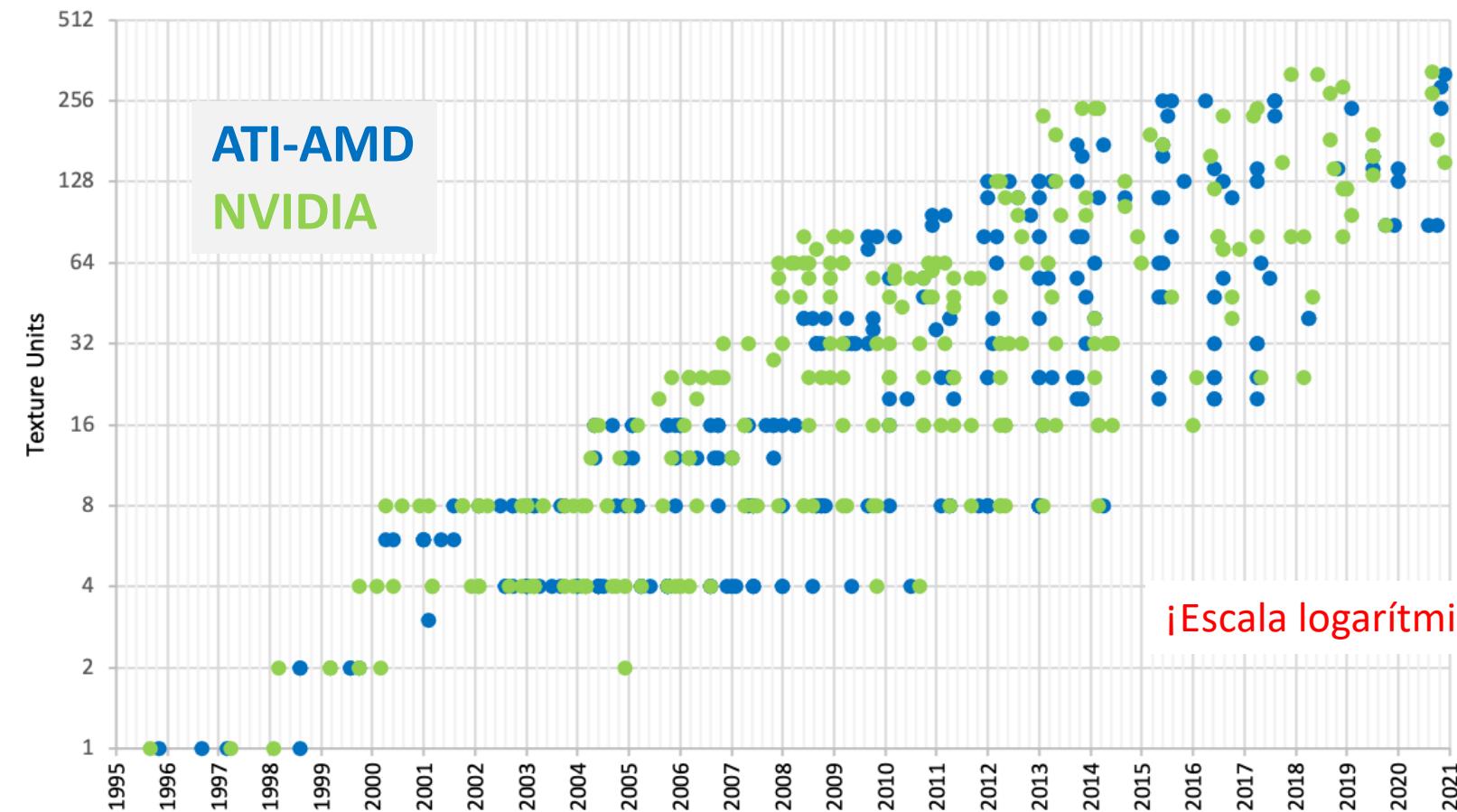
| AMD                  | Año  | TU  | Mtex/s  |
|----------------------|------|-----|---------|
| Rage 128 Pro         | 1999 | 2   | 250     |
| Radeon 7200          | 2000 | 6   | 1098    |
| Radeon 7500          | 2001 | 6   | 1740    |
| Radeon 9000 Pro      | 2002 | 4   | 1.000   |
| Radeon 9800 XT       | 2003 | 8   | 3.296   |
| Radeon X850 XT       | 2004 | 16  | 8.320   |
| Radeon X1800 XT      | 2005 | 16  | 10.000  |
| Radeon X1950 XT      | 2006 | 16  | 10.000  |
| Radeon HD 2900 XT    | 2007 | 16  | 11.900  |
| Radeon HD 4870       | 2008 | 40  | 30.000  |
| Radeon HD 4890       | 2009 | 40  | 34.000  |
| Radeon HD 5870       | 2009 | 80  | 68.000  |
| Radeon HD 6970       | 2010 | 96  | 84.500  |
| Radeon HD 6930       | 2011 | 80  | 60.000  |
| Radeon HD 7970       | 2012 | 128 | 118.400 |
| Radeon HD 8970       | 2013 | 128 | 134.400 |
| Radeon R9 290X       | 2013 | 176 | 176.000 |
| Radeon R9 280        | 2014 | 112 | 92.600  |
| Radeon R9 Fury X     | 2015 | 256 | 266.800 |
| Radeon RX 480        | 2016 | 144 | 161.300 |
| Radeon RX Vega 64    | 2017 | 256 | 393.200 |
| Radeon RX 590        | 2018 | 144 | 211.500 |
| Radeon VII (Vega 20) | 2019 | 240 | 432.000 |
| Radeon RX 5700 XT    | 2019 | 160 | 256.800 |
| Radeon RX 6900 XT    | 2020 | 320 | 7200000 |

# Fill Rate (MTexels/s)



Fill Rate (MTexels/s): número de texels (unidad mínima de textura) que una GPU puede procesar por segundo.

# Texture Units



¡Escala logarítmica!

Texture Units: elementos encargados de leer y filtrar texturas. Se usan exclusivamente para aplicaciones gráficas

# Introducción

## Algoritmo Básico de Renderización

- Iluminación más realista a mayor densidad de vértices.
- Necesitamos más vértices para mejorar el detalle de las superficies.
- **Consecuencia:**  
 $\uparrow$ realismo  $\Rightarrow$   $\uparrow$  vértices  $\Rightarrow$   $\uparrow$ cálculo  $\Rightarrow$   $\downarrow$ fps
- **Solución:**
  - Definir las superficies con  $\downarrow$ vértices y
  - Especificar los detalles superficiales con

# TEXTURAS

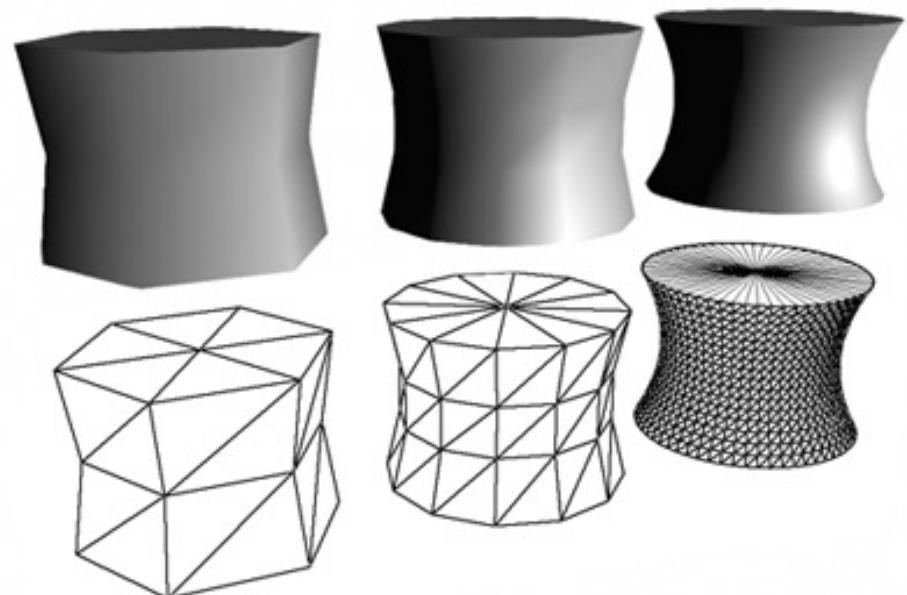


Imagen tomada de «The Cg Tutorial – The definitive Guide to Programmable Real-Time Graphics» disponible online en developer.nvidia.com



# Aumentando la calidad de la imagen: Texturas



Bethesda Softworks's Oblivion

Múltiples efectos se consiguen utilizando texturas:

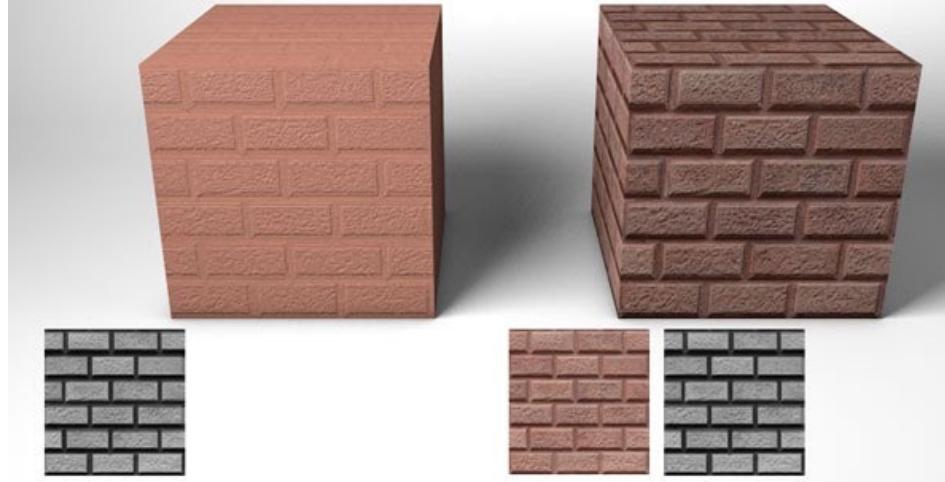
- Detalles superficie
- Reflejos
- Relieves
- ...



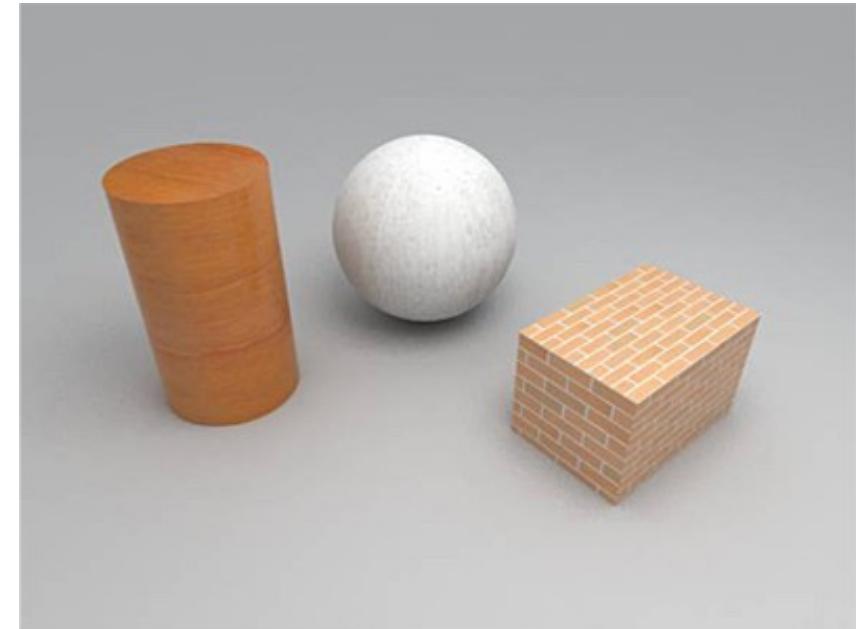
# Aumentando la calidad de la imagen: Texturas



Bump Map



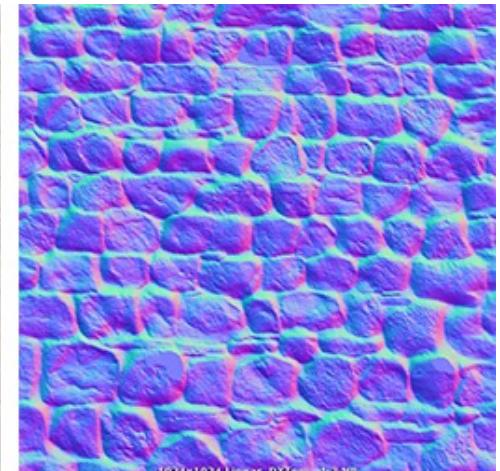
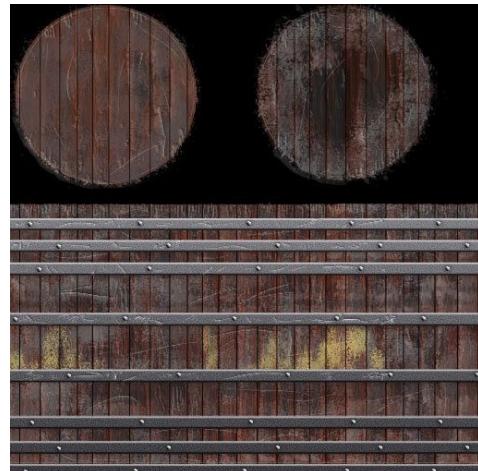
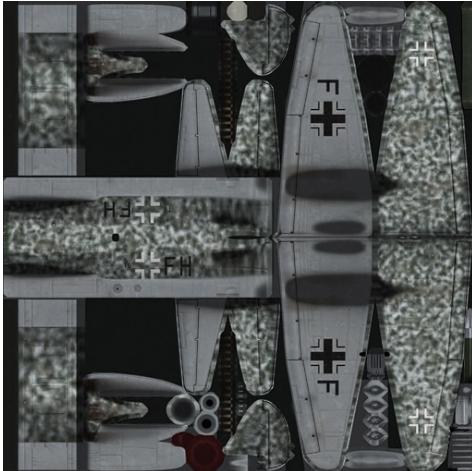
Texture Map + Bump Map



Las texturas permiten aumentar la calidad de la imagen a un coste razonable.

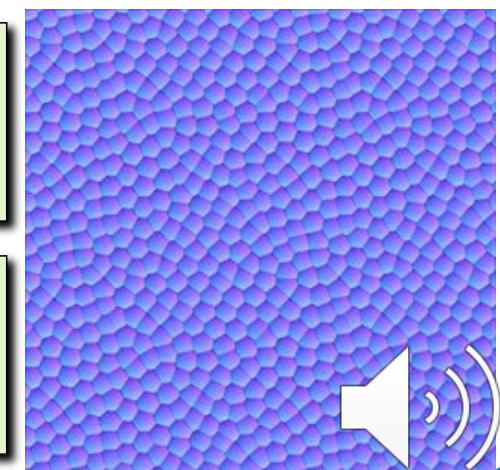


# Aumentando la calidad de la imagen: Texturas



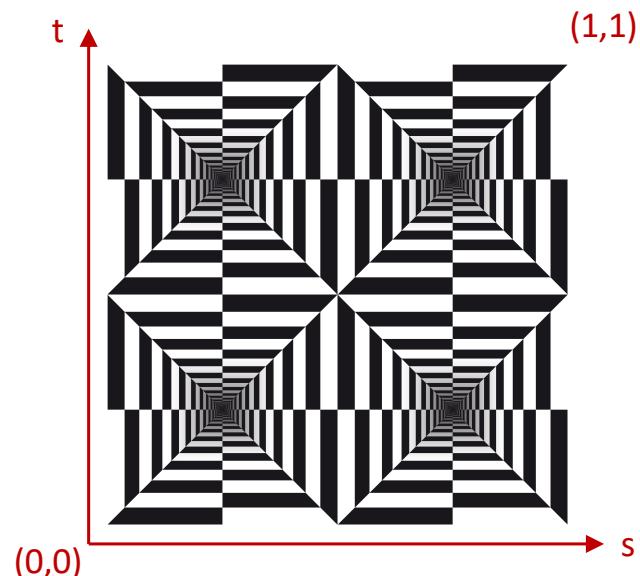
Una textura es una imagen almacenada en un fichero en un formato estándar.  
También se puede calcular por programa.

No tienen porqué ser cuadradas, pero en general, las dimensiones son siempre potencia de 2 ( $2^m \times 2^n$ ).



# Acceso: coordenadas de Texturas

- Las coordenadas de texturas son un atributo más de los vértices.
- Y posteriormente de los fragmentos.
- El rango de las coordenadas de textura ( $s, t$ ) está entre 0 y 1, no importa el tamaño de la textura en píxeles.
- Si las dimensiones de la textura son potencia de 2 ( $2^m \times 2^n$ ), calcular la dirección para acceder a la textura a partir de ( $s, t$ ) es trivial.



ssssssssssssssss

tttttttttttttt

Coordenadas de textura ( $s, t$ )

Textura de dimensiones  
 $1024 \times 1024 (2^{10} \times 2^{10})$ .

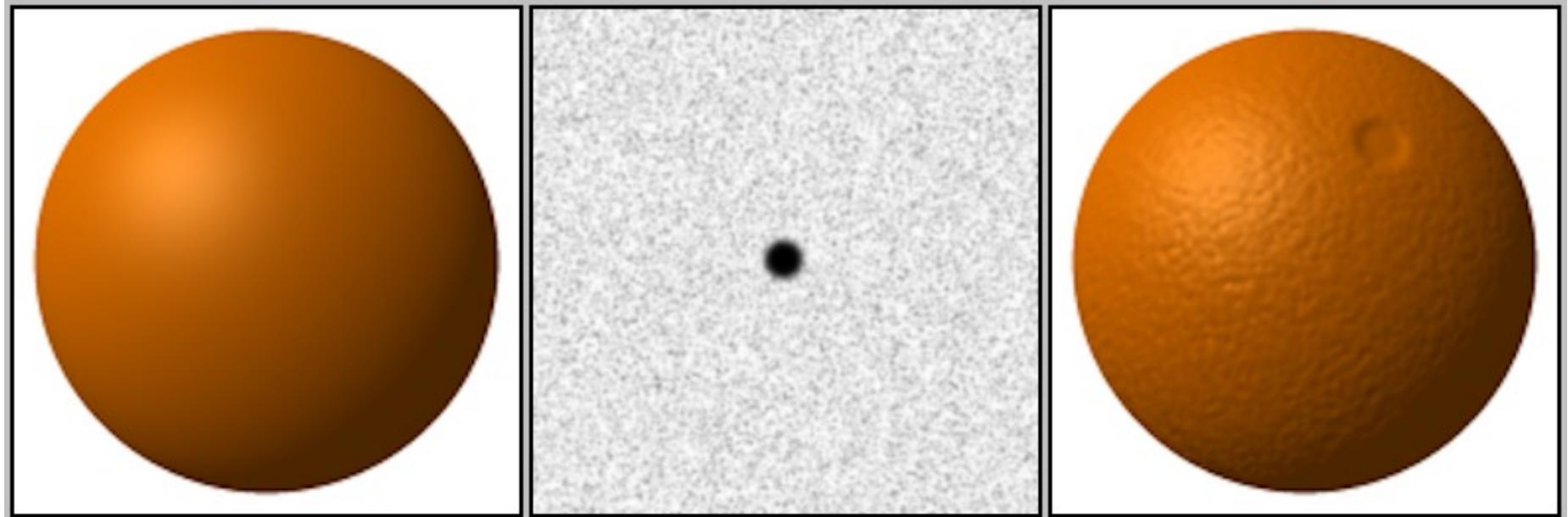
ssssssssss

tttttttttt

Índices para acceder a memoria

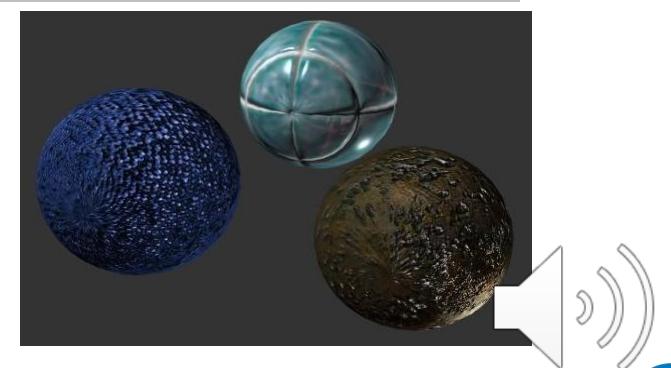


# Bump Mapping



- La textura contiene los vectores de la normal en cada punto.
- Usados para modificar la normal de cada fragmento respecto a las fuentes de iluminación.
- Efecto 3D
- Técnica introducida por James F. Blinn en 1978 (1).

(1) James F. Blinn. "Simulation of Wrinkled Surfaces". Computer Graphics, Vol. 12(3), pp286-292 SIGGRAPH-ACM (Aug 1978)



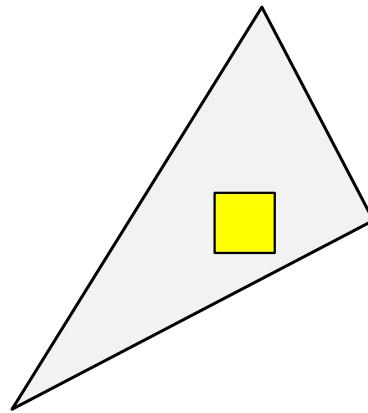
# Environmental Mapping



- Se basa en la generación de un cube map.
- 6 imágenes (texturas), se calculan como una imagen cualquiera, variando la posición del observador.
- A continuación se aplica la textura.
- El mapping también se podría hacer sobre una esfera



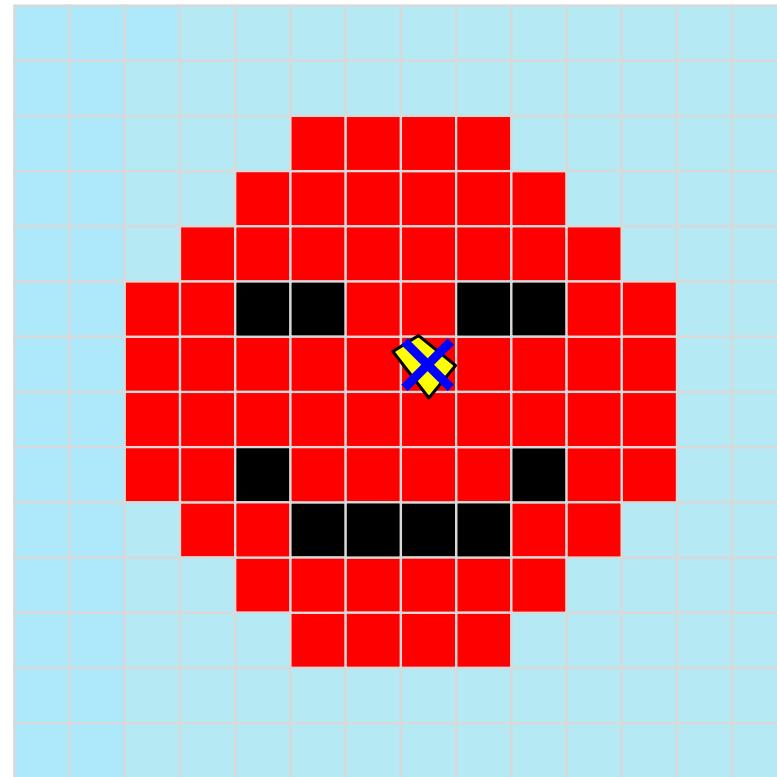
# Aplicación de Texturas: Filtro Lineal



A partir del fragmento se calculan las coordenadas de textura ( $s, t$ ).  
Se utiliza el texel más cercano.

**Coste:**

- 1 acceso Memoria

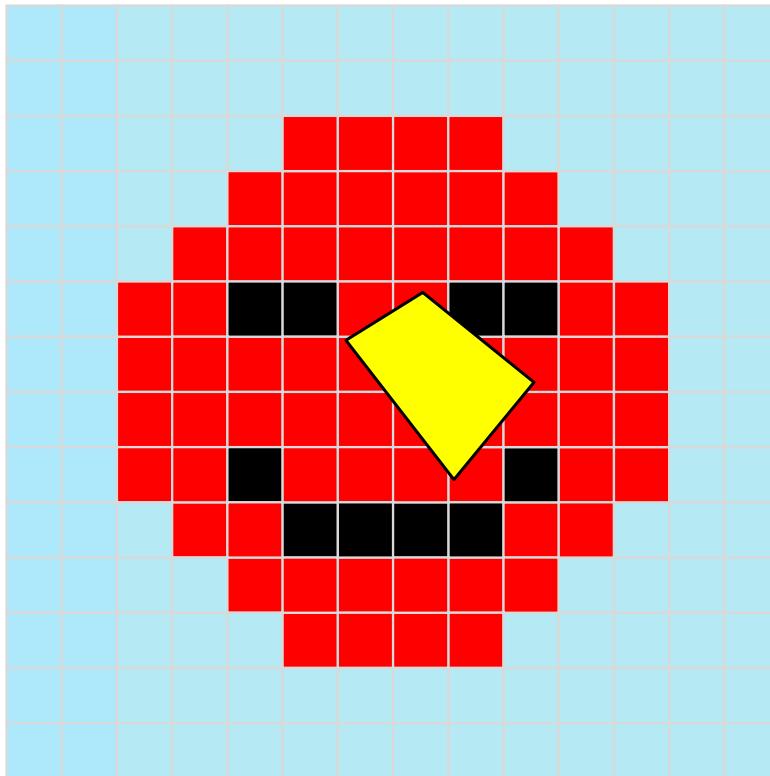


Resultado **ACEPTABLE** si:

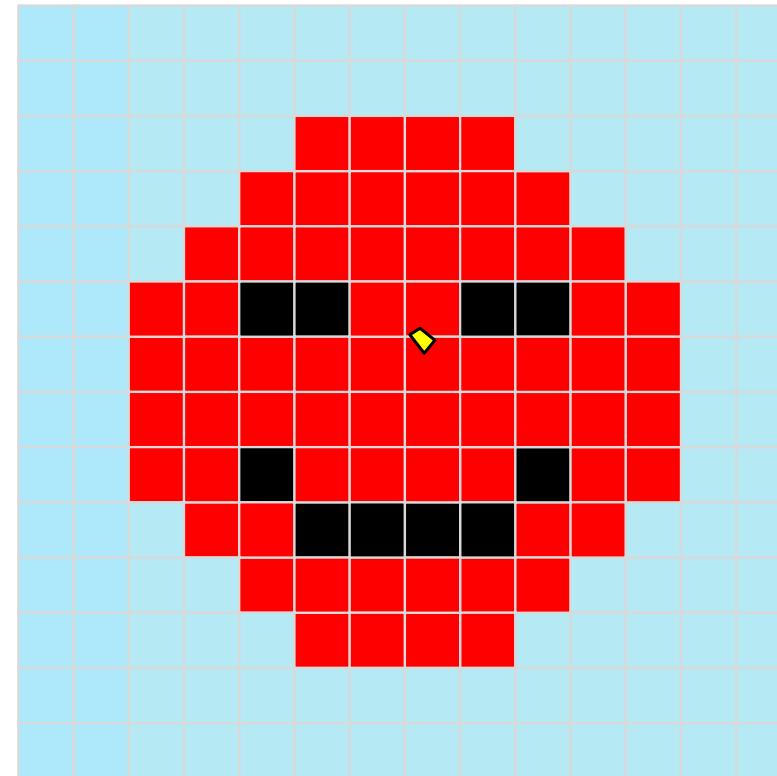
- Tamaño Pixel  $\approx$  Tamaño Texel



# Aplicación de Texturas: Filtro Lineal



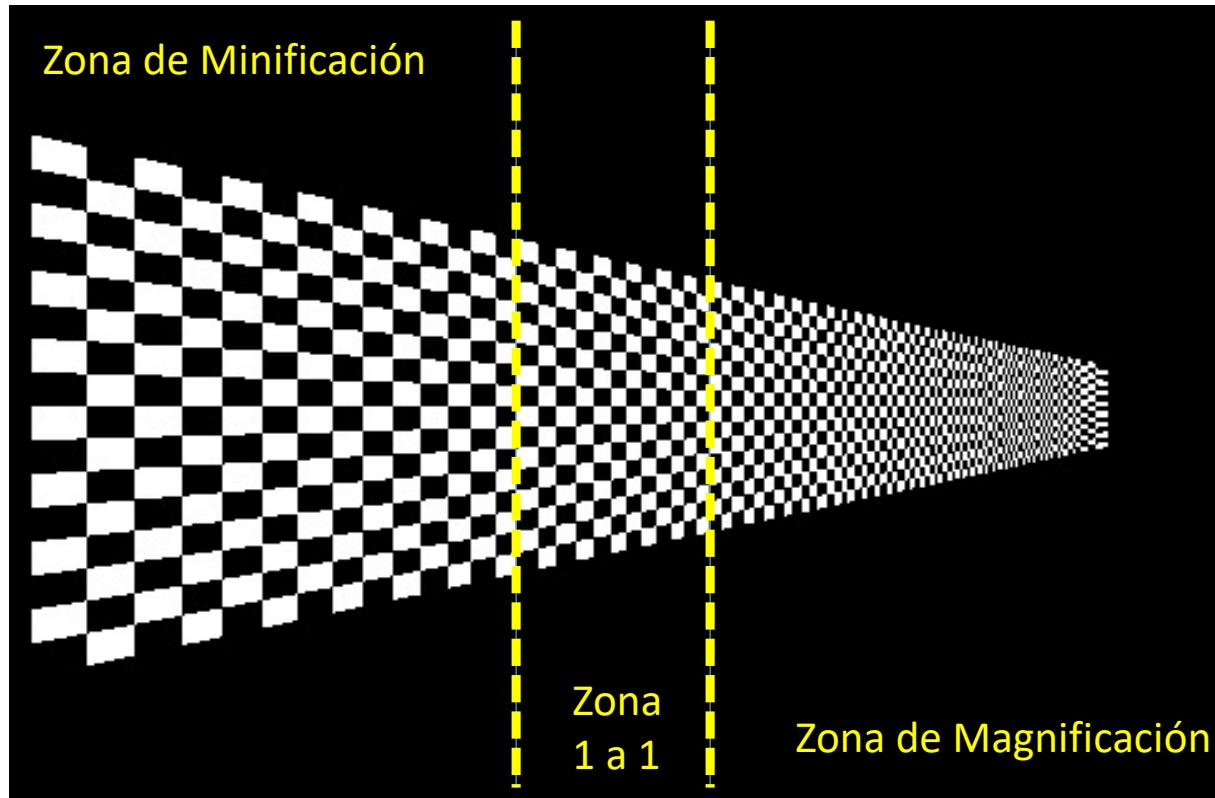
- Tamaño Pixel > Tamaño Texel  
**MINIFICATION**



- Tamaño Pixel < Tamaño Texel  
**MAGNIFICATION**



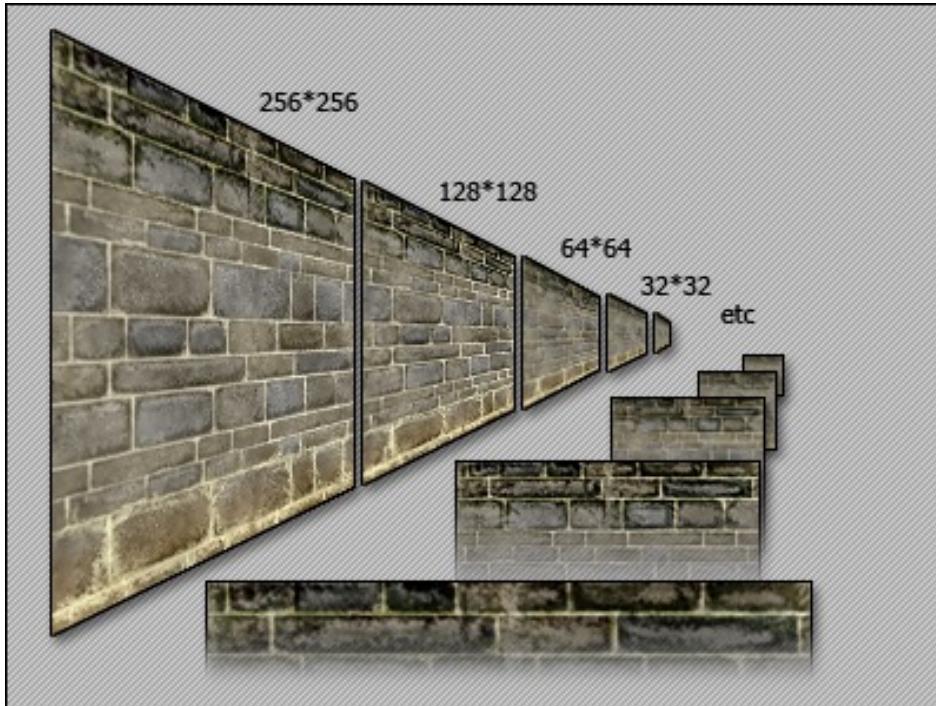
# Aplicación de Texturas: Filtro Lineal



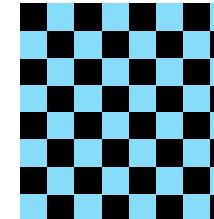
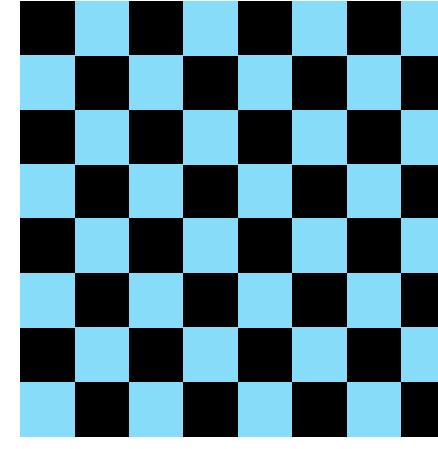
Problemas de aliasing evidentes



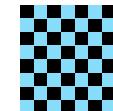
# MipMapping



Concepto introducido por Lance Williams en: “Pyramidal Parametrics”, SIGGRAPH 1983.



**¡No confundir con la compresión de texturas!**

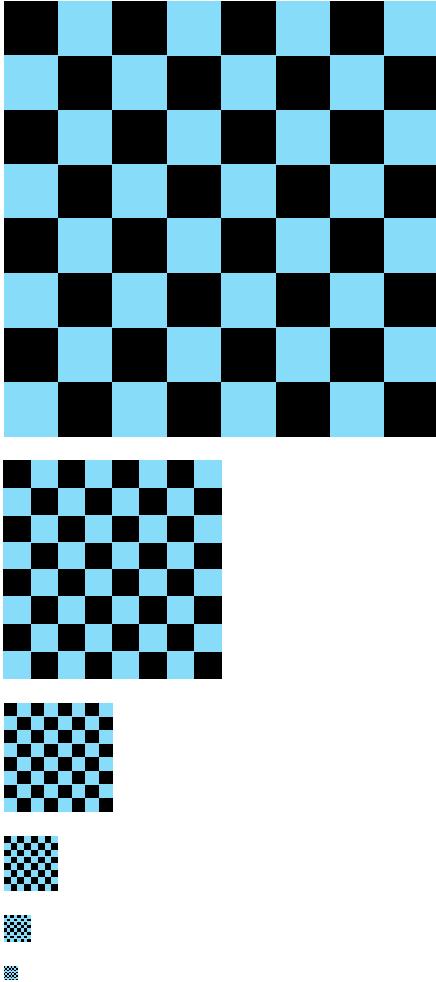


**mip**

“multum in parvo”  
mucho en poco espacio



# MipMapping



- Construir un mipmap de dimensiones ( $2^{m-1} \times 2^{n-1}$ ) a partir de otro mipmap de dimensiones ( $2^m \times 2^n$ ) es trivial.
- Cada grupo de 4 texels ( $2 \times 2$ ) se convierte en 1 texel.
- OpenGL tiene una función para generar los mipmaps de forma automática:

```
void glGenerateTextureMipmap(GLuint texture);
```

- Almacenamiento necesario:  $1 + 1/4 + 1/16 + \dots \approx 1+1/3$



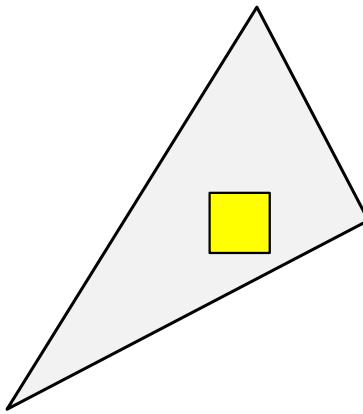
# MipMapping



Assassin's creed 3



# Aplicación de Texturas: Filtro BiLineal

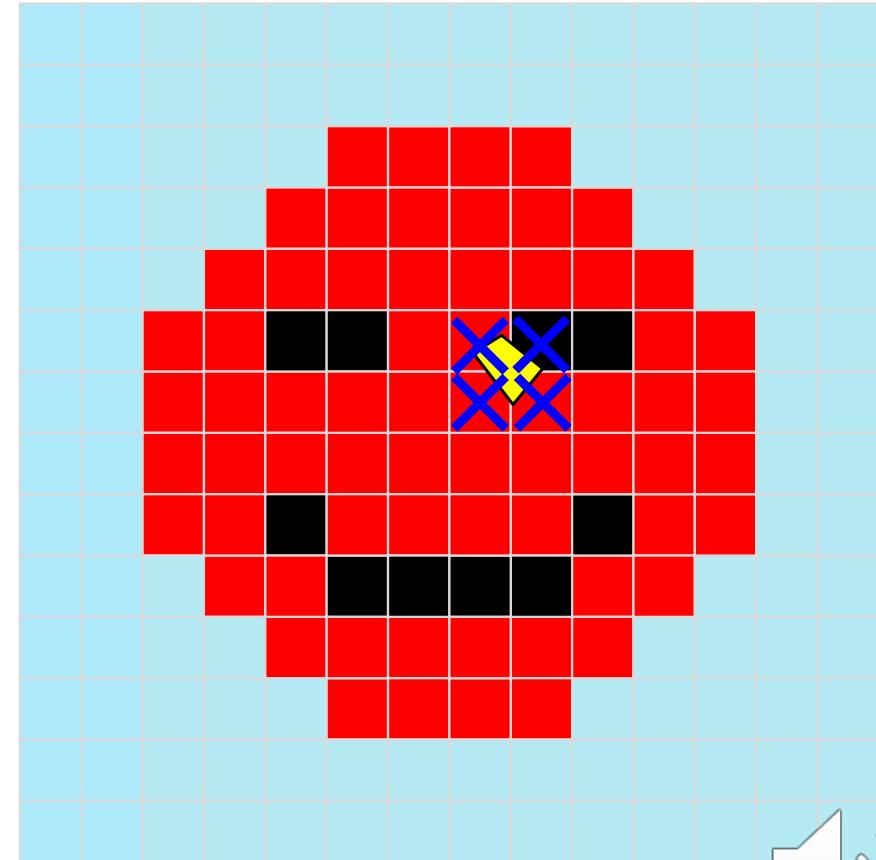


Se utiliza el mipmap que mejor se ajuste al tamaño del pixel.

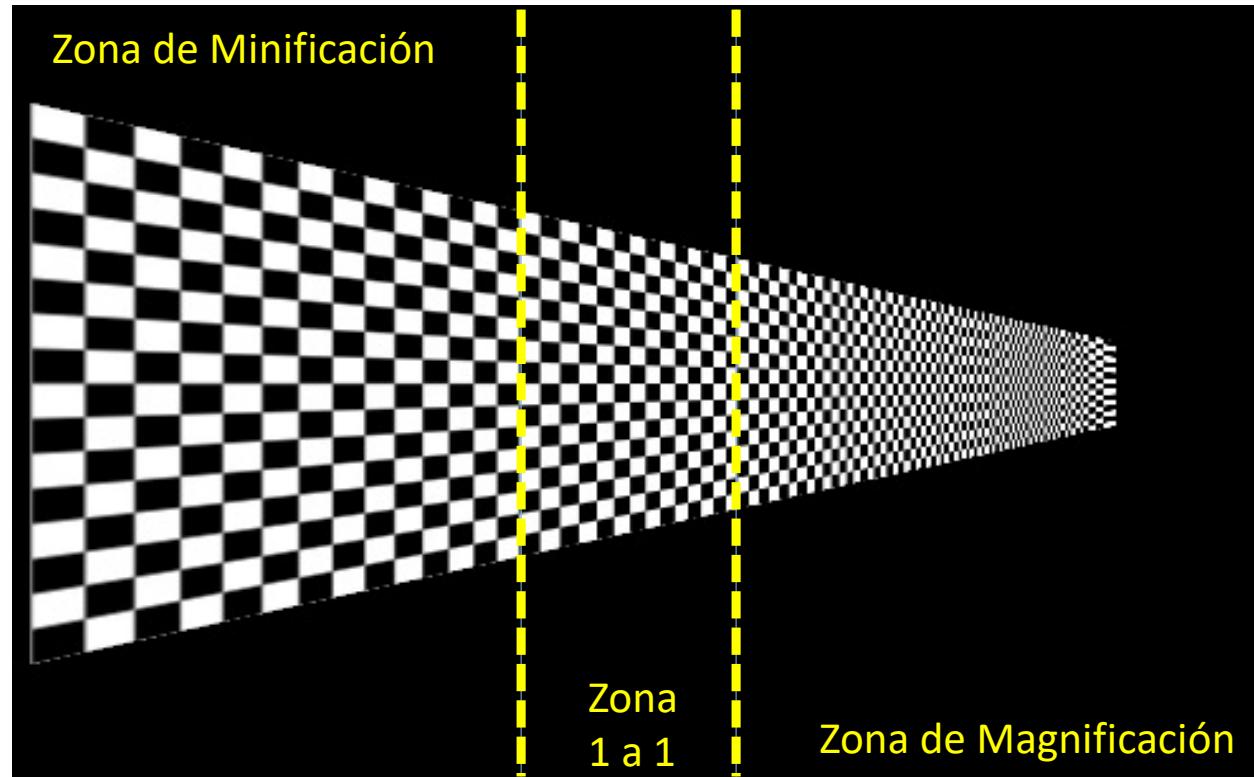
Se utiliza el valor promedio de los 4 texeles más cercanos.

## Coste:

- 4 accesos a Memoria
- ReSampling (3 MUL y 6 ADD)



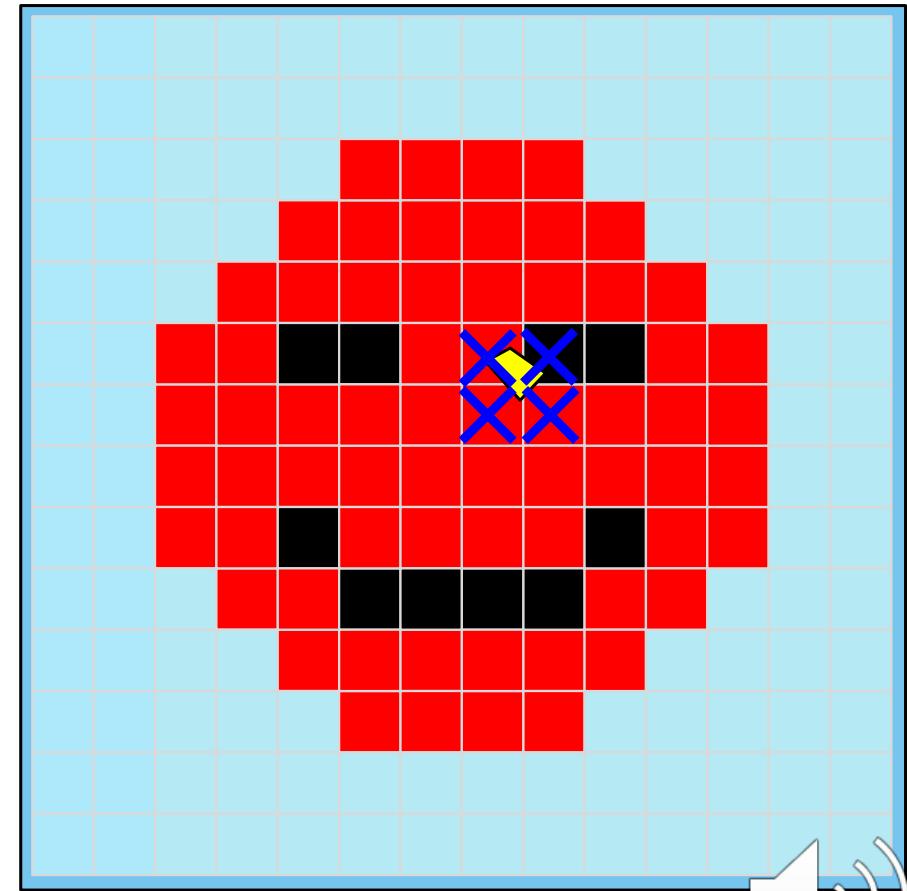
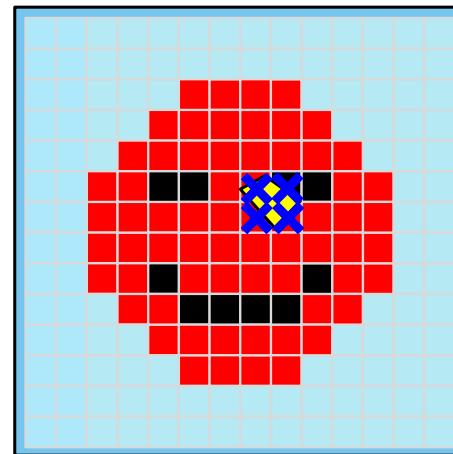
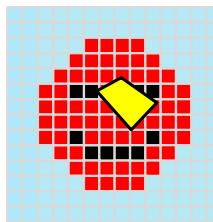
# Aplicación de Texturas: Filtro BiLineal



Problemas de aliasing en la zona de magnificación



# Aplicación de Texturas: Filtro TriLineal



Se utilizan los 2 mipmap que mejor se ajusten al tamaño del pixel.

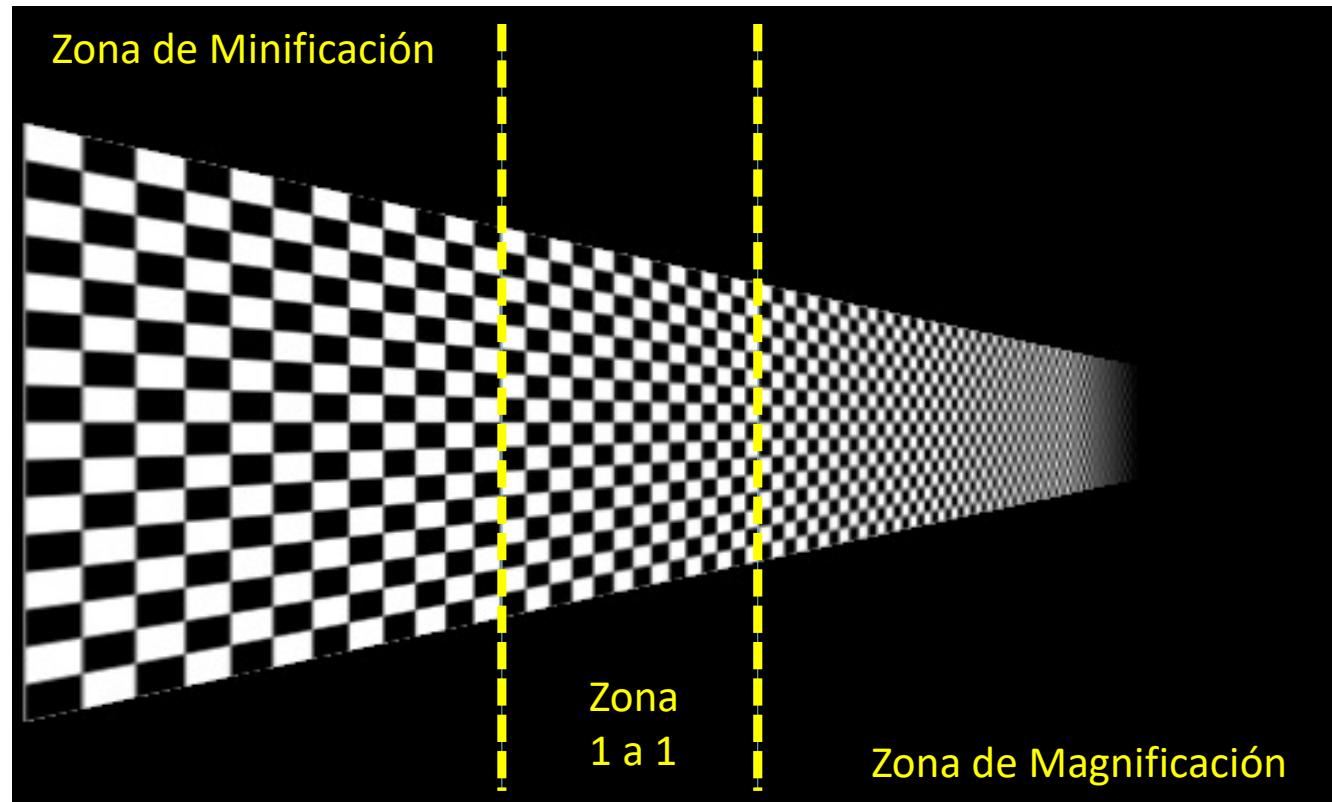
Se utiliza un filtro BiLineal en cada mipmap. Equivale al valor promedio de los 4 texeles más cercanos en cada mipmap.

**Coste:**

- 8 accesos a Memoria
- ReSampling (7 MUL y 14 ADD)

El Ancho de banda con Memoria se multiplica por 8.

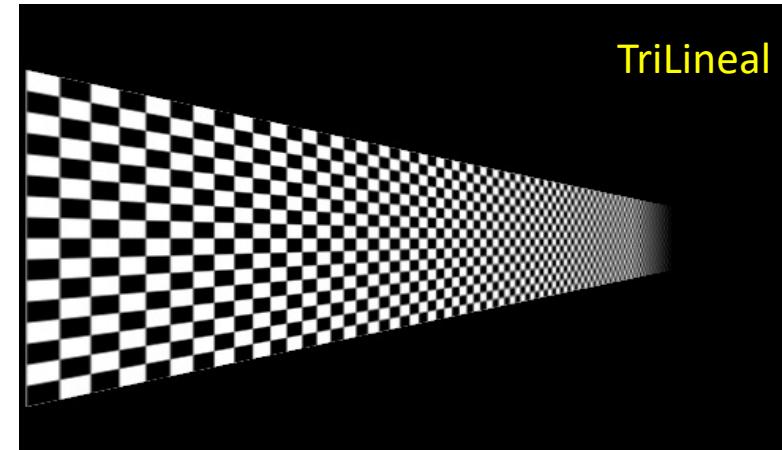
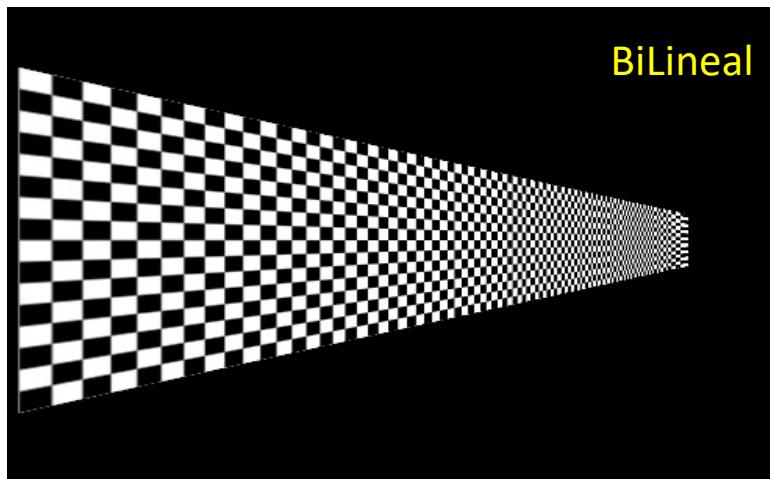
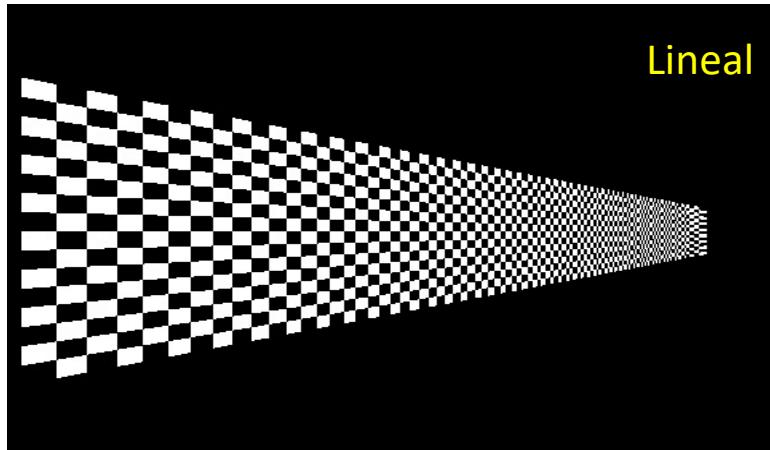
# Aplicación de Texturas: Filtro TriLineal



El incremento de calidad es evidente.



# Comparando Filtros

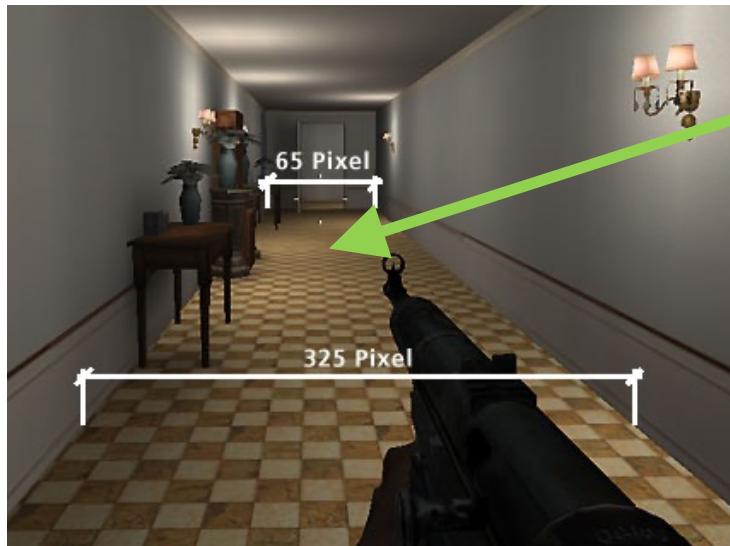


## Coste:

- Lineal: 1 acceso a Memoria
- BiLineal: 4 accesos a Memoria + Interpolación
- TriLineal: 8 accesos a Memoria+ Interpolación

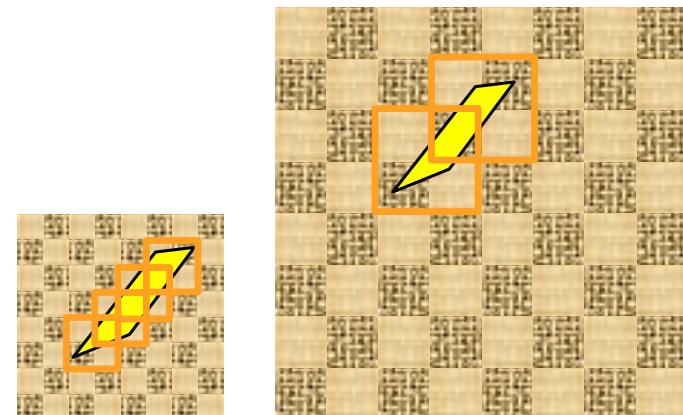


# Aplicación de Texturas: Filtro Anisotrópico



En estas zonas la relación de aspecto entre los fragmentos y las texturas puede ser muy variable.

ANISOTRÓPICO ≈ "Sin Forma"



Se aplica varias veces un filtro BiLinear o TriLinear.



# Aplicación de Texturas: Filtro Anisotrópico



Filtro Anisotrópico (x16): Se pueden interpolar hasta 128 valores.  
Equivalente a 16 TriLinear.

Coste: entre 8 y 128 accesos a Memoria + Interpolación.



# Aplicación de texturas



# Texture Unit

Información que define una textura:

- Dirección en memoria
- Tipo de textura: 1D, 2D, 3D, cubemap, ...
- Dimensiones de la textura.
- Tipo de textura: comprimida/NO comprimida, color, normal, profundidad, ...
- Filtro a utilizar: point sample, bilinear, trilinear, aniso x ...
- Wrapping mode: clamp, repeat, ...
- LOD (level of detail) bias, niveles mipmap, ...

La Texture Unit recibe:

- 1 Dirección de textura:
  - 1D: {s}, 2D: {s, r}, 3D: {s, r, t}
- Un identificador de textura
- Y un lod bias por fragmento



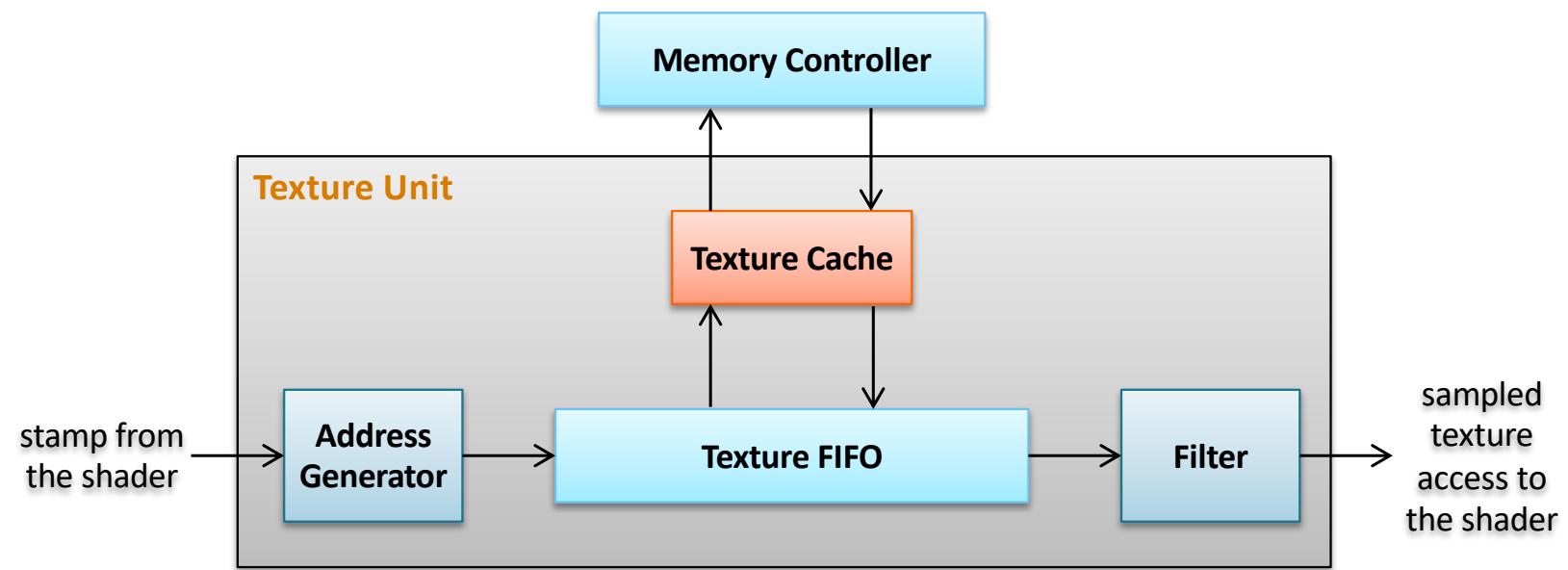
# Texture Unit

**La Texture Unit ha de obtener/calcular: 1 texel (point sampling) o N texels (bilinear / trilinear) para procesar 1 fragmento:**

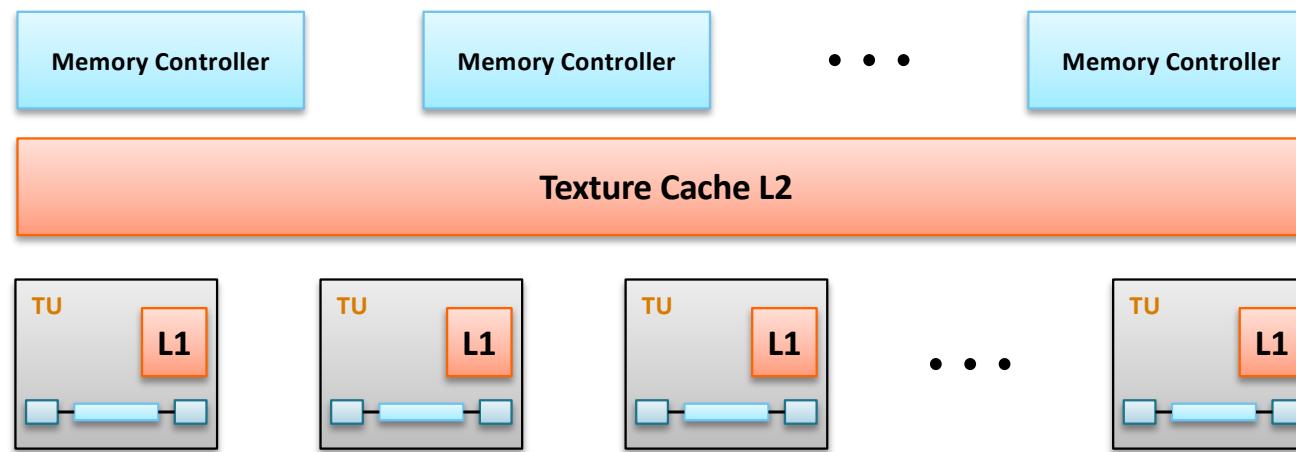
- Seleccionar mipmap (1, o 2) dependiendo del lod bias y del filtro a aplicar.
- Calcular las coordenadas del texel en el mipmap correspondiente.
  - Traducir la coordenada de textura a la dirección lógica
- Enviar la textura a la cache de texturas.
- Una vez obtenidos los texels hay que aplicar el filtro seleccionado:
  - Point sample (NEAREST): NO filtrar.
  - Bilinear (LINEAR).
  - Trilinear (LINEAR\_MIPMAP\_LINEAR).
  - ...
- Hay que tener una cola para almacenar las peticiones pendientes. A pesar de la cache, las latencias pueden ser muy altas.
- Una vez aplicado el filtro se pasan los datos al shader.



# Texture Unit



# Texture Unit



La cache de texturas ha de soportar un acceso trilineal (2 mipmaps) en 1 ciclo.  
Accesos paralelos para otros fragmentos se solucionan replicando la cache.

## Referencias

- Ziyad S. Hakura and Anoop Gupta, "The Design and Analysis of a Cache Architecture for Texture Mapping". ISCA 1997
- Homan Igehy, Matthew Eldridg and Kekoa Proudfoot, "Prefetching in a Texture Cache Architecture". Graphics Hardware 1998



# Texture Cache Architecture

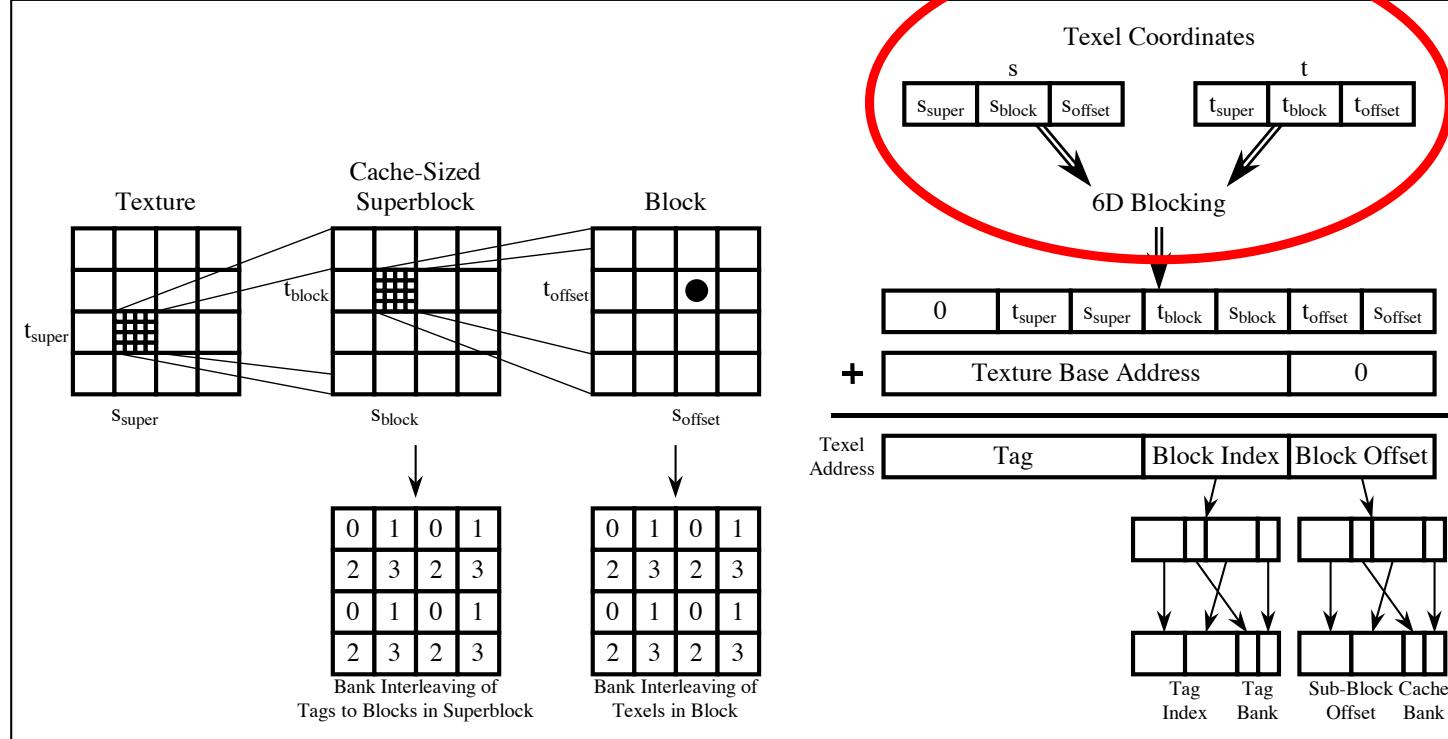
## Soportar 1 acceso trilinear en 1 ciclo es una tarea **NO TRIVIAL**

- Necesitamos leer 8 datos en 2 mipmaps diferentes.
- Los 2 mipmaps se han de acceder sin conflicto: asociatividad, multibanco , ...
- Hay que leer 4 texels de cada mipmap.
  - Pueden estar contiguos en memoria, se acceden fácilmente
  - Pueden estar en líneas de cache diferentes: la cache necesita puertos de lectura independientes [COSTOSO]
  - No es suficiente para evitar los conflictos
- Se soluciona almacenando las texturas de forma coherente con su uso. Buscando “localidad espacial” en aplicaciones gráficas.
- Las texturas se pueden almacenar usando “6D blocking” + Morton.
- Morton garantiza que 4 texels contiguos se almacenan en bancos diferentes y así evitamos conflictos.



# 6D Blocking + Morton

Dimensiones textura:  $2^n \times 2^n$



Homan Igehy, Matthew Eldridg and Kekoa Proudfoot, "Prefetching in a Texture Cache Architecture". Graphics Hardware 1998



# 6D Blocking + Morton

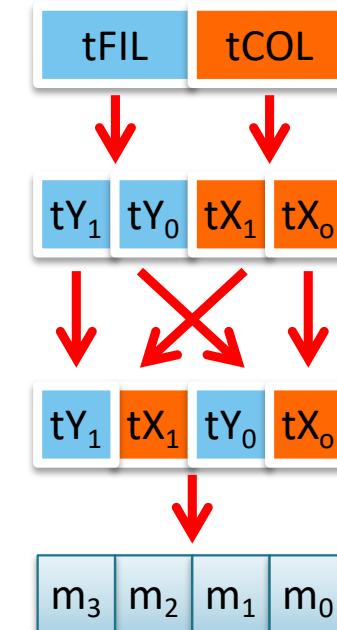
|   | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | A | B | E | F |
| 1 | C | D | G | H |
| 2 | I | J | M | N |
| 3 | K | L | O | P |

textura

|   | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | A | B | C | D |
| 1 | E | F | G | H |
| 2 | I | J | K | L |
| 3 | M | N | O | P |

memoria

| tex | tFIL | tCOL | mFIL | mCOL |
|-----|------|------|------|------|
| A   | 00   | 00   | 00   | 00   |
| B   | 00   | 01   | 00   | 01   |
| E   | 00   | 10   | 01   | 00   |
| F   | 00   | 11   | 01   | 01   |
| C   | 01   | 00   | 00   | 10   |
| D   | 01   | 01   | 00   | 11   |
| G   | 01   | 10   | 01   | 10   |
| H   | 01   | 11   | 01   | 11   |
| I   | 10   | 00   | 10   | 00   |
| J   | 10   | 01   | 10   | 01   |
| M   | 10   | 10   | 11   | 00   |
| N   | 10   | 11   | 11   | 01   |
| K   | 11   | 00   | 10   | 10   |
| L   | 11   | 01   | 10   | 11   |
| O   | 11   | 10   | 11   | 10   |
| P   | 11   | 11   | 11   | 11   |



Banco en la Cache



# 6D Blocking + Morton

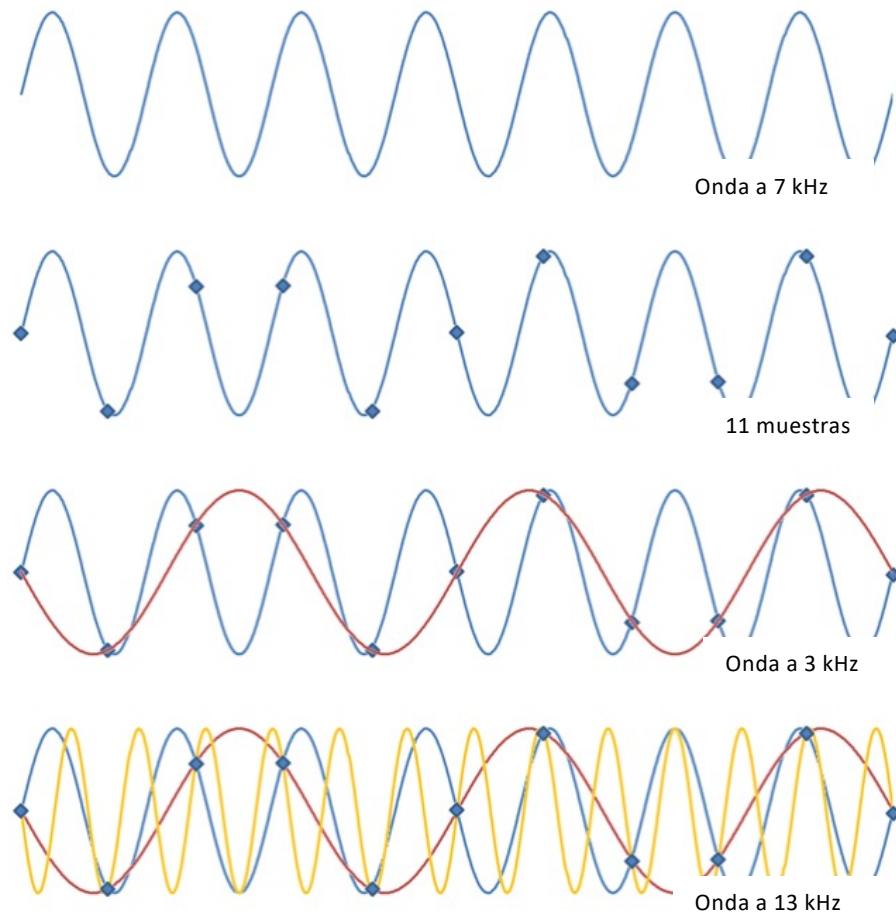
|    |    |    |    |
|----|----|----|----|
| 00 | 01 | 00 | 01 |
| 10 | 11 | 10 | 11 |
| 00 | 01 | 00 | 01 |
| 10 | 11 | 10 | 11 |

Banco cache

¡Cualquier bilineal (4 accesos contiguos) se reparte entre los 4 bancos de cache!

|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 |
| 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 |
| 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 |
| 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 |
| 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 |
| 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 |
| 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 |
| 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 |
| 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 |
| 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 |
| 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 | 00 | 01 |
| 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 | 10 | 11 |

# Aliasing



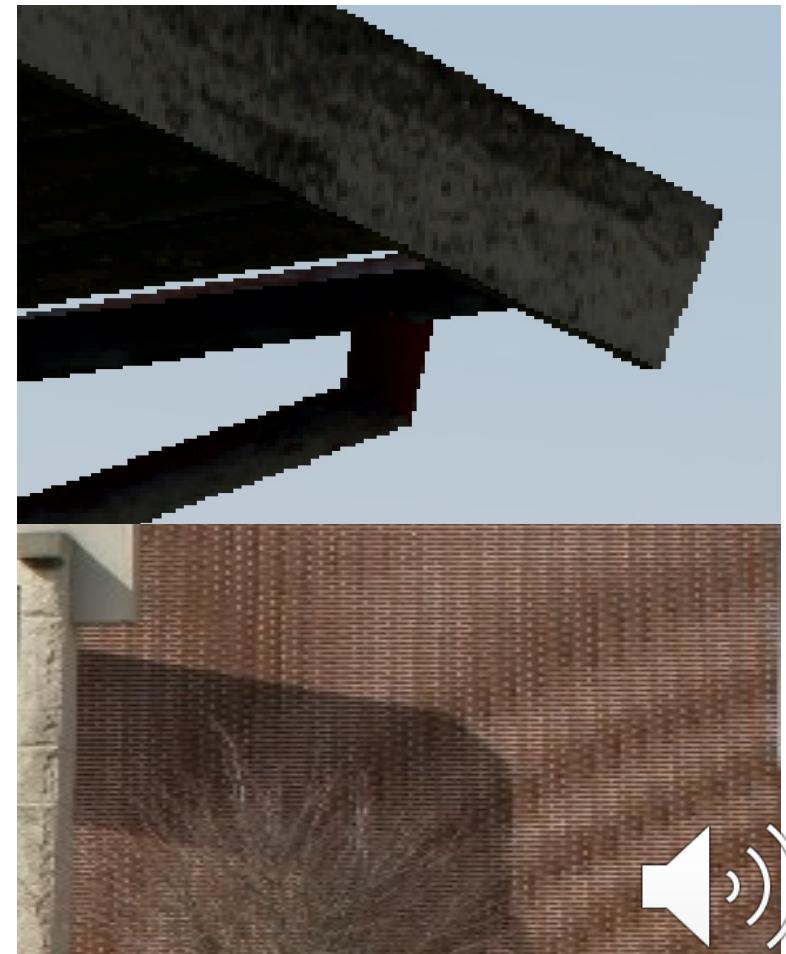
Cuando se discretiza una señal analógica pueden aparecer problemas de aliasing. Para evitarlos la frecuencia de muestreo ha de doblar la frecuencia de la señal original.

[\[LINK1\]](#)

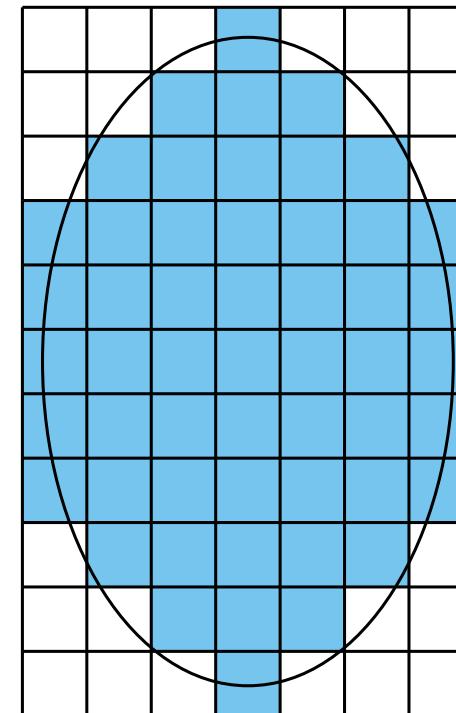
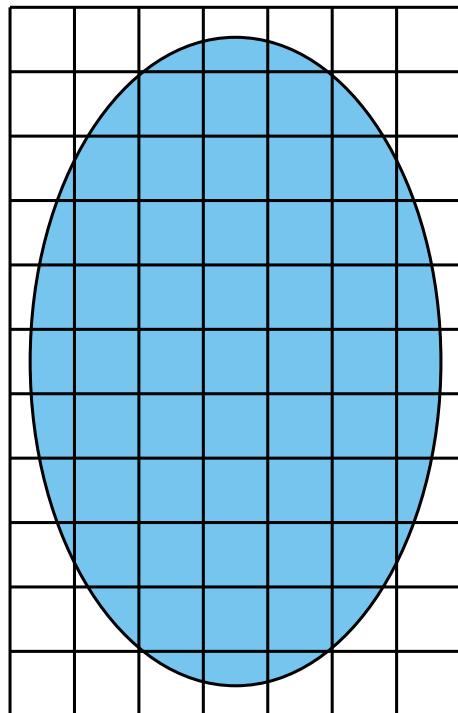


# Aliasing en gráficos 3D

- El aliasing es el artefacto gráfico característico que hace que en una pantalla ciertas curvas y líneas inclinadas presenten un efecto visual tipo "sierra" o "escalón".
- El aliasing ocurre cuando se intenta representar una imagen con curvas y líneas inclinadas, pero debido a la resolución finita del sustrato resulta que éste es incapaz de representar la curva como tal, y por tanto dichas curvas se muestran en pantalla dentadas al estar compuestas por pixeles.
- Este aliasing hay que corregirlo al rasterizar.
- Otro tipo de aliasing es el moiré [[LINK](#)]
- El uso de filtros bilinear, trilinear, ... son técnicas de antialiasing para evitar el moiré.



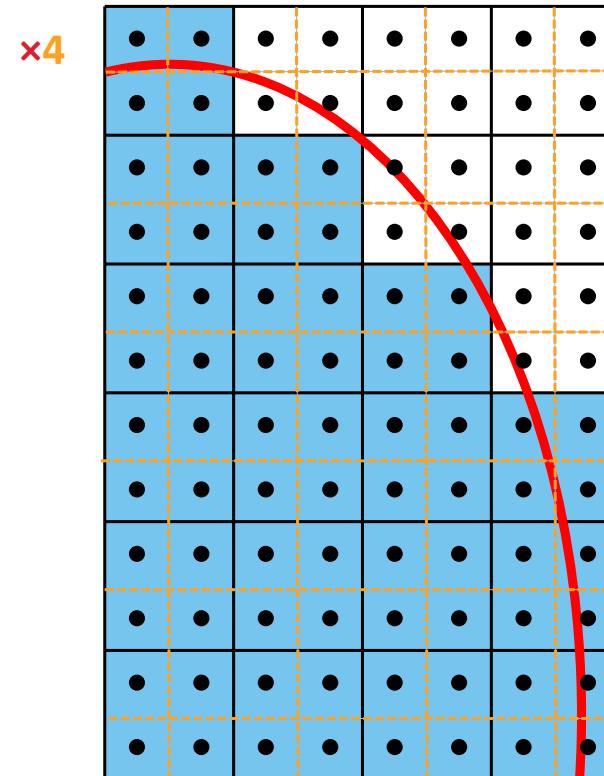
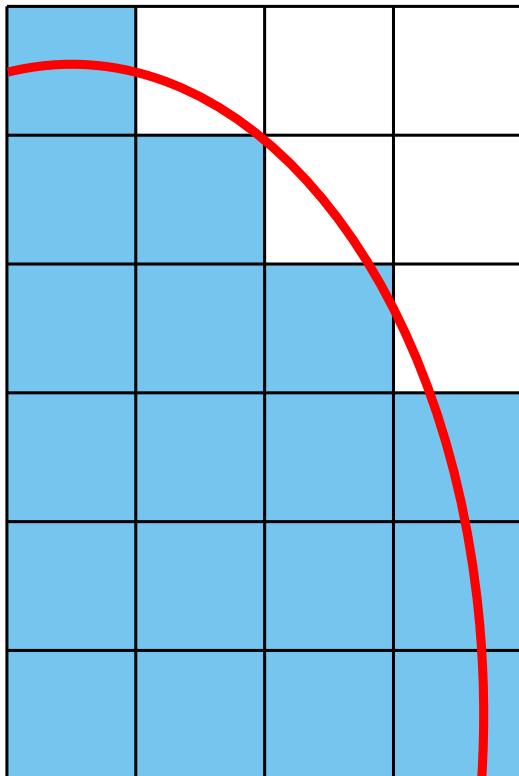
# SuperSampling



Si no hacemos nada, la calidad de la imagen se ve muy afectada.



# SuperSampling



Se aplica fuerza bruta. Si la resolución es  $M \times N$ , se renderiza a mayor resolución, p.e.  $2M \times 2N$  ( $\times 4$ ).



# SuperSampling

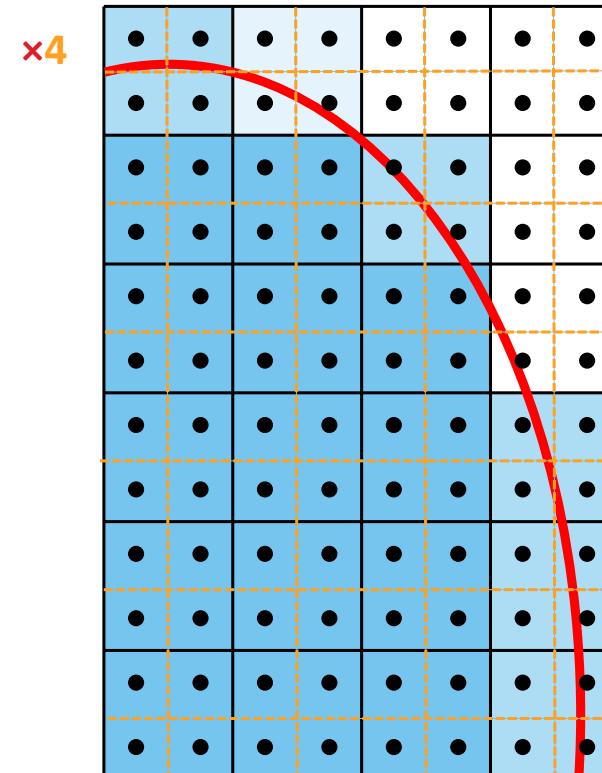
$$\frac{\bullet \square + \bullet \square + \bullet \square + \bullet \square}{4} = \square$$

$$\frac{\bullet \square + \bullet \square + \bullet \square + \square \bullet}{4} = \square$$

$$\frac{\bullet \square + \bullet \square + \square \bullet + \square \bullet}{4} = \square$$

$$\frac{\bullet \square + \bullet \square + \square \bullet + \bullet \square}{4} = \square$$

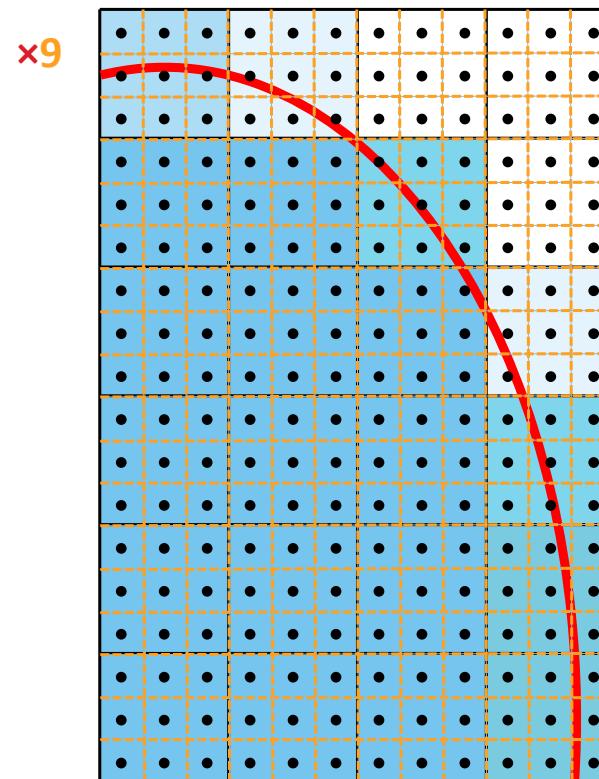
$$\frac{\bullet \square + \bullet \square + \square \bullet + \bullet \square}{4} = \square$$



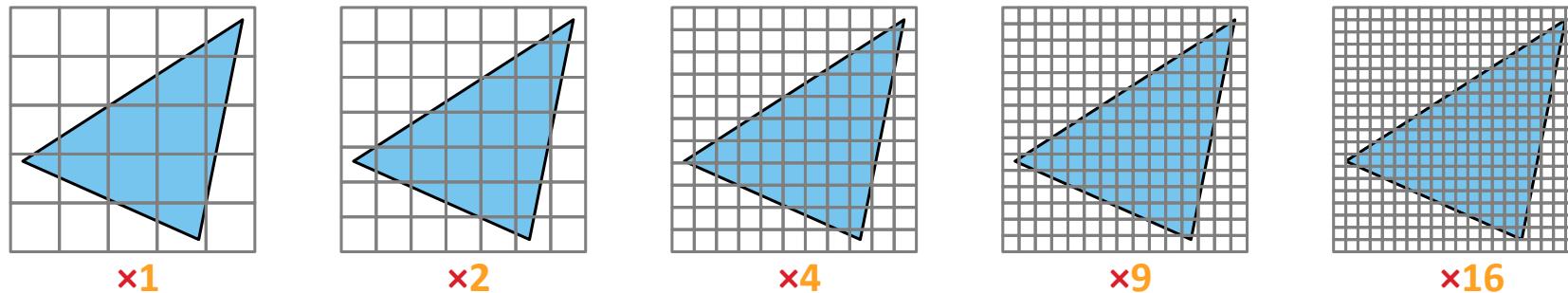
# SuperSampling

Si es necesario, se puede aumentar la resolución para mejorar la calidad de la imagen.

¡Las necesidades de cálculo y de ancho de banda se disparan!



# SuperSampling

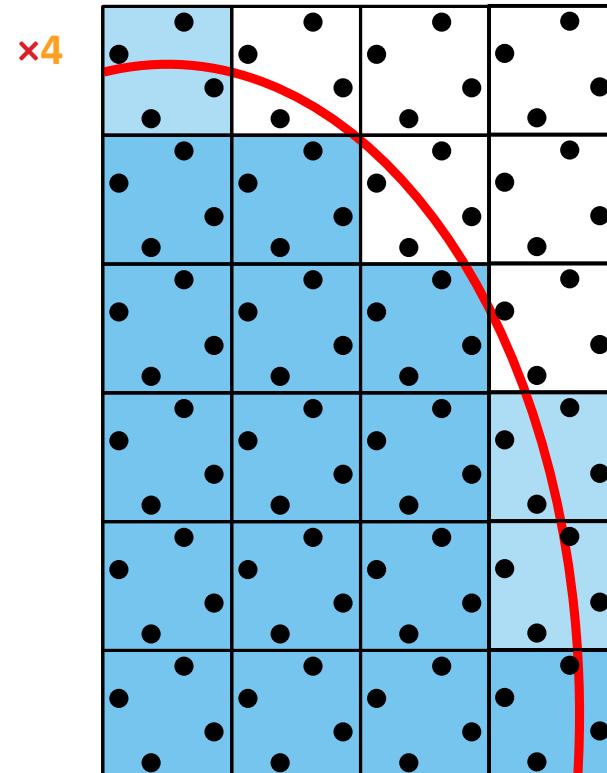
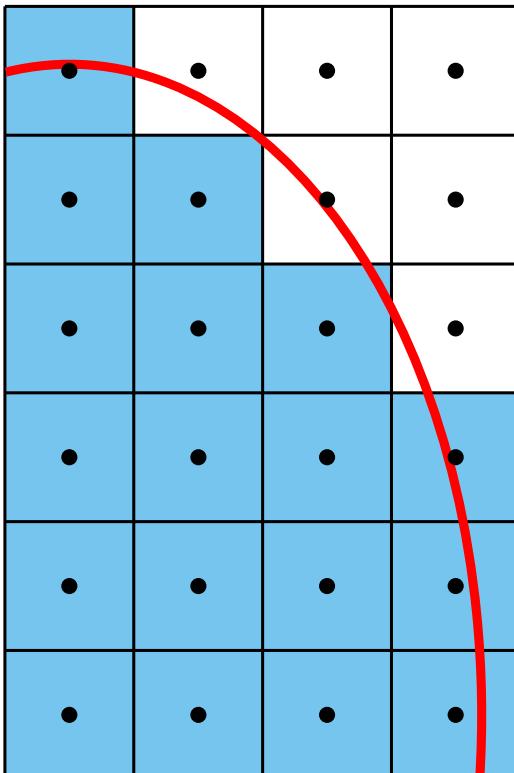


|                                | x1      | x2      | x4      | x9       | x16      |
|--------------------------------|---------|---------|---------|----------|----------|
| Triángulos ( $10^6$ )          | 1,33    | 1,33    | 1,33    | 1,33     | 1,33     |
| Vértices ( $10^6$ )            | 4       | 4       | 4       | 4        | 4        |
| Fragmentos ( $10^6$ )          | 15,6    | 33,2    | 66,3    | 149,3    | 265,4    |
| R/W vértices ( $10^6$ bytes)   | 128     | 128     | 128     | 128      | 128      |
| R/W fragmentos ( $10^6$ bytes) | 1.658,9 | 3.317,8 | 6.635,5 | 14.929,9 | 26.542   |
| Op CF vértice ( $10^6$ )       | 112     | 112     | 112     | 112      | 112      |
| Op CF fragmento ( $10^6$ )     | 1.061,7 | 2.123,4 | 4.246,7 | 9.555,1  | 16.986,9 |
| Ancho banda (GB/s)             | 53,8    | 103,6   | 203,1   | 451,9    | 800,3    |
| GFLOPS                         | 35,2    | 67,1    | 130,7   | 290,0    | 512,9    |

Resolución pantalla: 1920×1080 píxeles  
Sobreescritura: 8 fragmentos por píxel  
RD/WR: 100 B por fragmento, 32 B por vértice  
Cálculo: 28 ops CF por vértice, 64 ops CF por fragmento



# MSAA (Multi Sample Anti Aliasing)

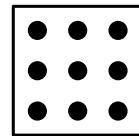


Se aplica el muestreo sin necesidad de generar los fragmentos.  
¡ Las necesidades de ancho de banda se contienen !

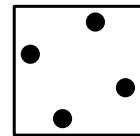


# MSAA (Multi Sample Anti Aliasing)

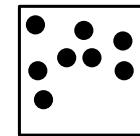
- Existen muchos posibles muestreos



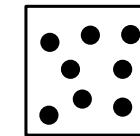
grid



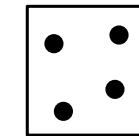
rotated grid



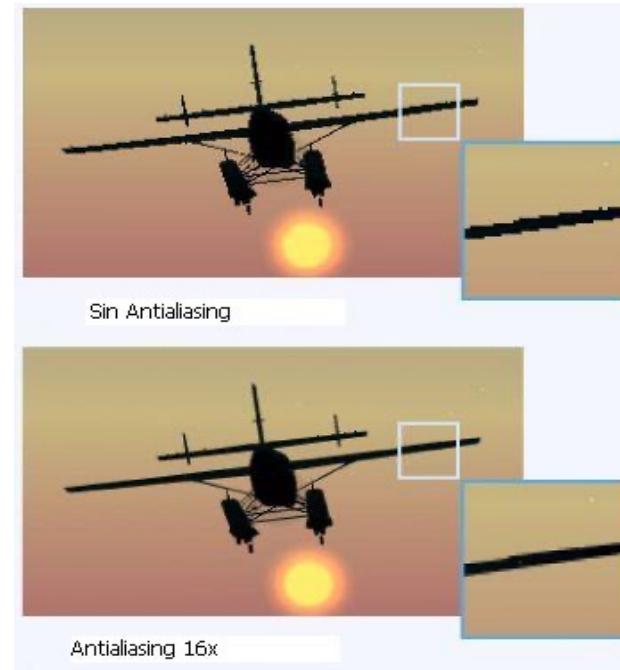
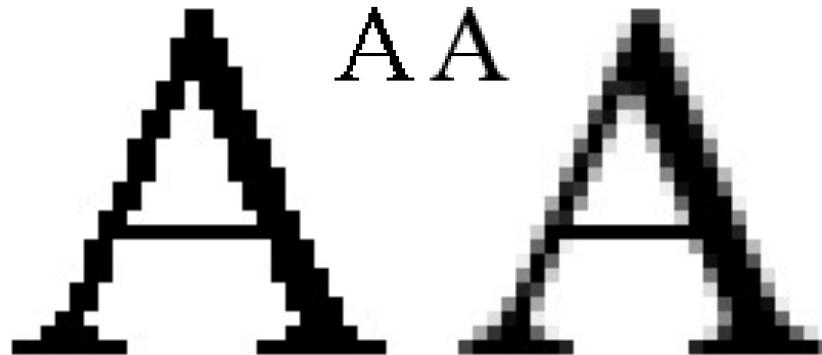
random



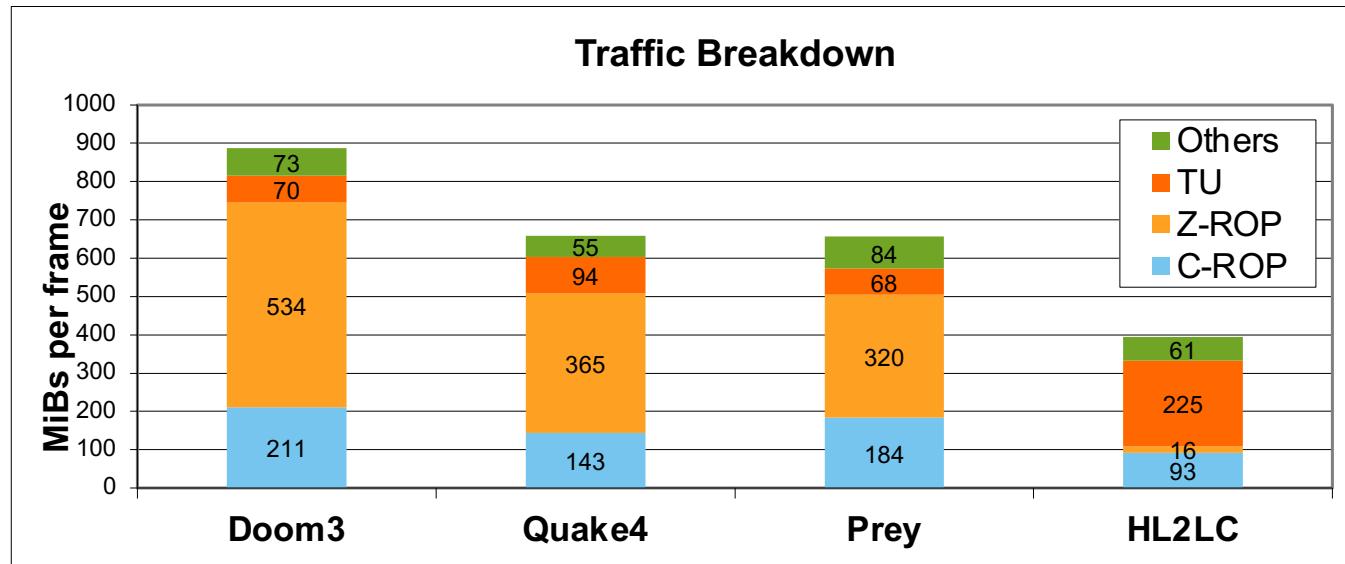
poisson disc



jitter



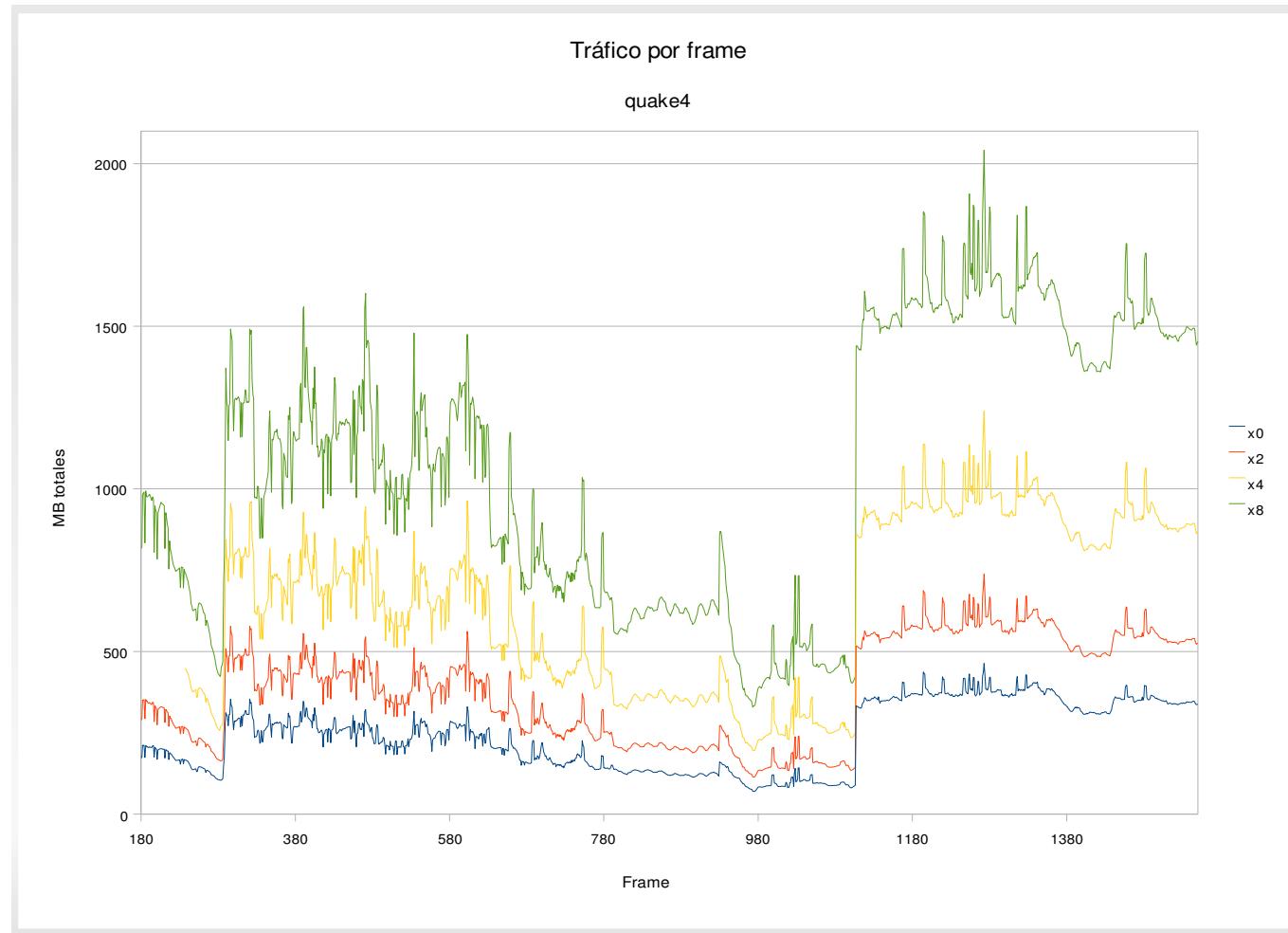
# Tráfico con Memoria



- Los elementos que más tráfico generan son TU, C-ROP y Z-ROP
- La influencia de las ROP es debida al MSAA
- HL2LC tiene el MSAA desactivado
- La distribución no es uniforme
- La cantidad de información que se lee/escribe en memoria es muy elevada.

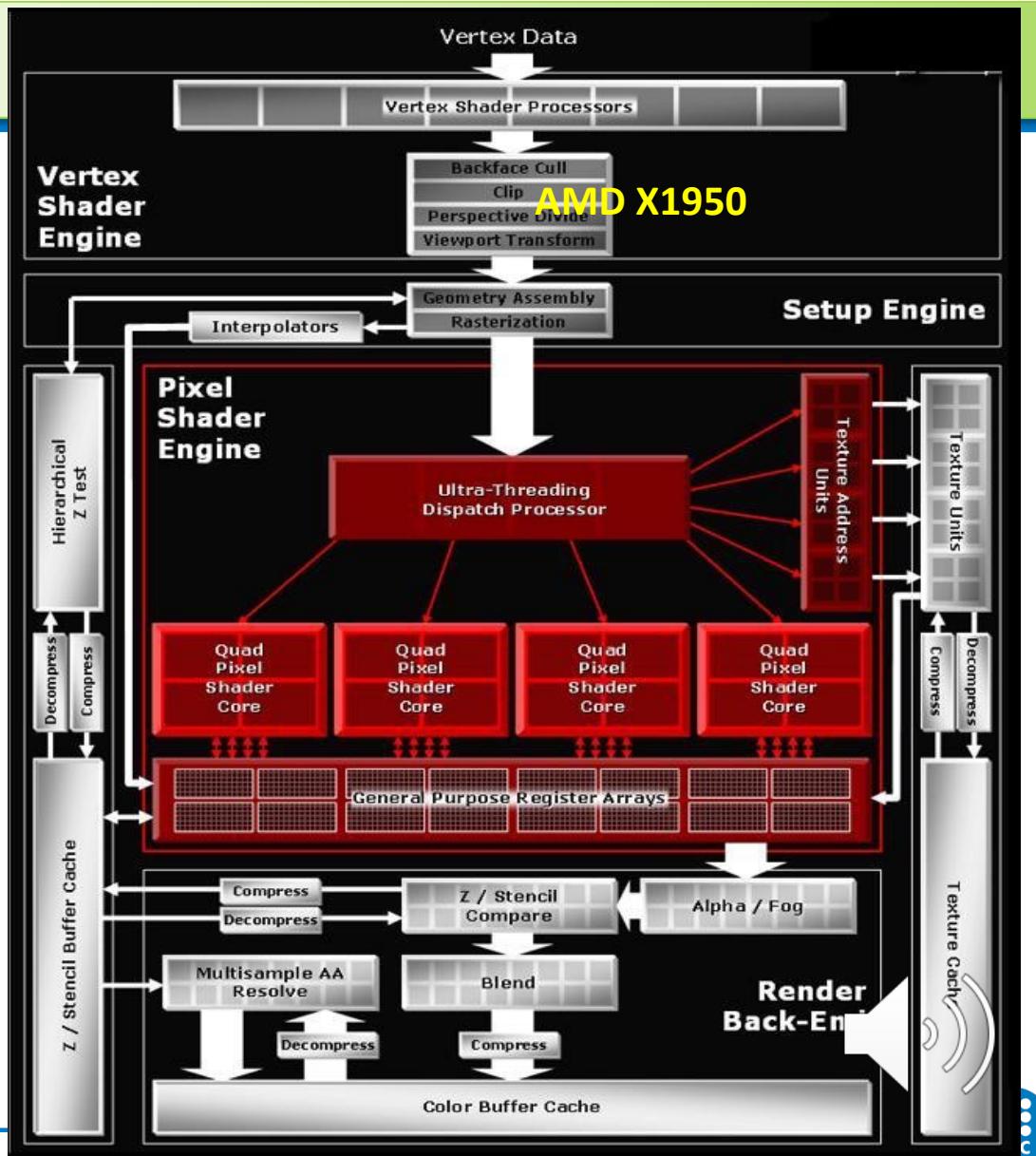


# Tráfico con Memoria en función del MSAA



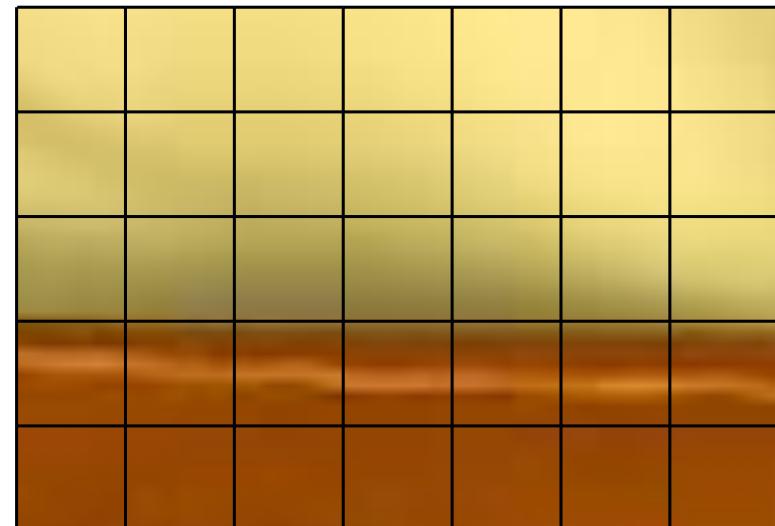
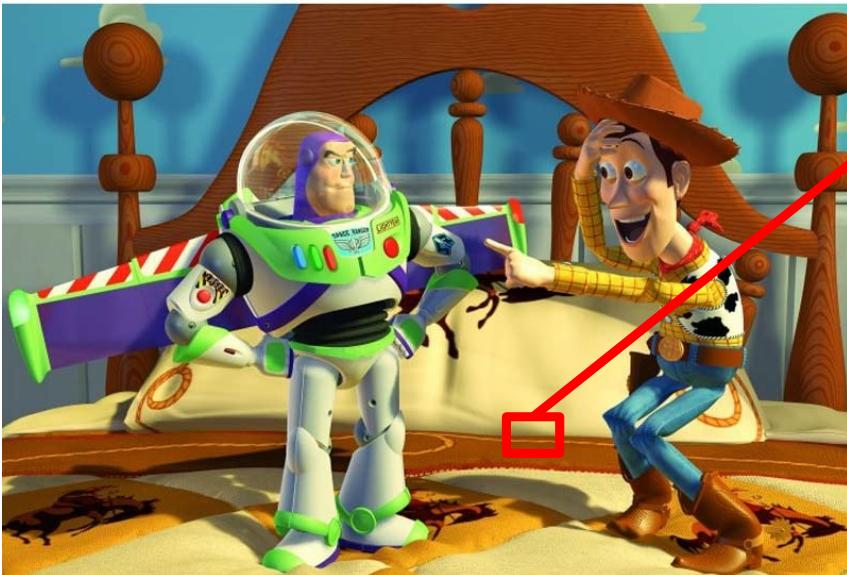
# Compresión

- Una forma de reducir las necesidades de ancho de banda es comprimir la información.
- Estamos tratando con imágenes.



# Compresión / Decompresión

- Una forma de reducir las necesidades de ancho de banda es comprimir la información.
- Estamos tratando con imágenes.



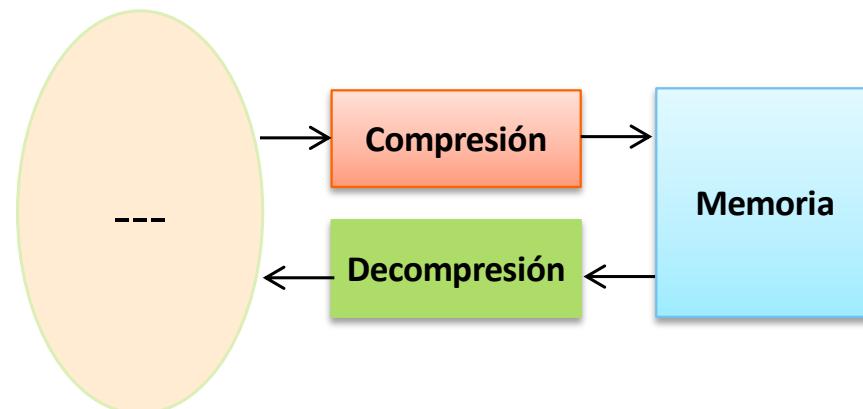
La información almacenada en una línea de cache es muy similar.

¡ Gran oportunidad para aplicar algoritmos de compresión !



# Compresión / Decompresión

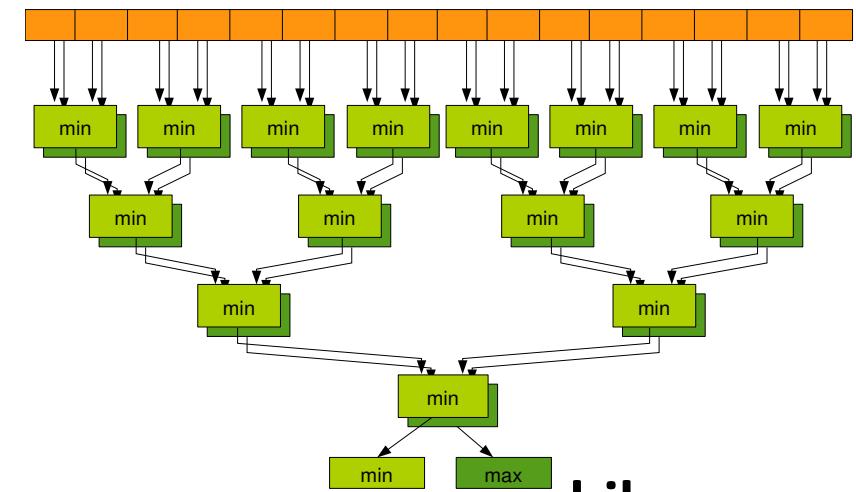
- Tamaño línea de cache (p.e.): 256B
- Tamaño 1 acceso memoria: 8B (64 bits)
- Acceso a memoria: ráfagas de 8 accesos, transacciones de 64B.
- Para leer 1 línea de cache es necesario 4 accesos.
- El objetivo de la compresión sería que:
  - Leer una línea de 256B (4 accesos)
  - Se haga con 1, 2 o 3 accesos.
- La compresión / decompresión está en el camino de acceso a memoria.
- Los algoritmos han de ser:
  - Altamente paralelizables
  - Divisibles en etapas
  - Latencia de ejecución baja
  - Bloques de salida tamaño múltiplo de transacción a memoria.



# Algoritmos de Compresión/Decompresión

## Offset Compression

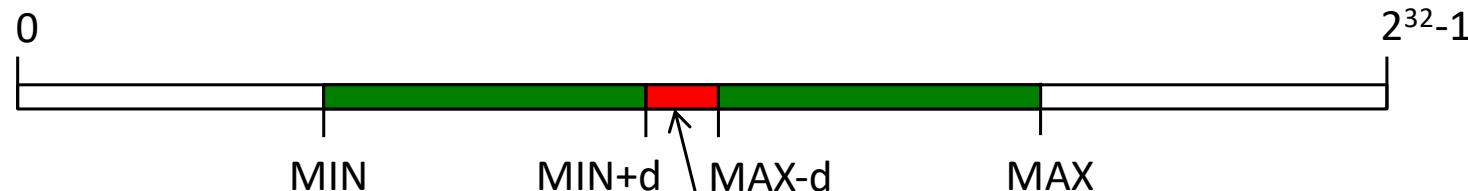
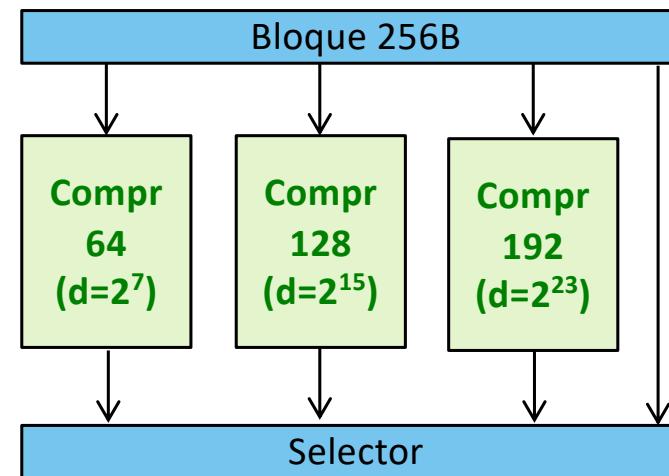
- Se comprimen todos los valores a partir de unos valores de referencia.
  - Típicamente los valores máximo y mínimo.
- Cada dato se codifica como:
  - Un índice al valor de referencia (para dos valores, 1 bit)
  - La diferencia entre el dato y el valor de referencia
- Descomprimir es fácil y rápido.
  - Se puede hacer en paralelo
- Comprimir es costoso (en tiempo y en hardware), hay que calcular un máximo y un mínimo.
  - No es necesario utilizar los 32 bits del dato para calcularlos



# Algoritmos de Compresión/Decompresión

hilo

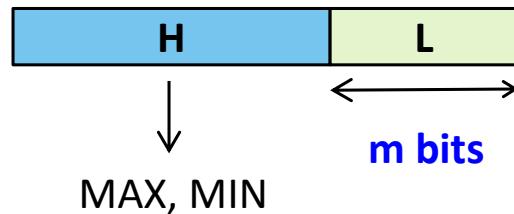
Los bits que dedicamos al desplazamiento ( $d$ ) son los que fijan el grado de compresión



Si algún valor cae en esta zona ese nivel de compresión queda descartado.



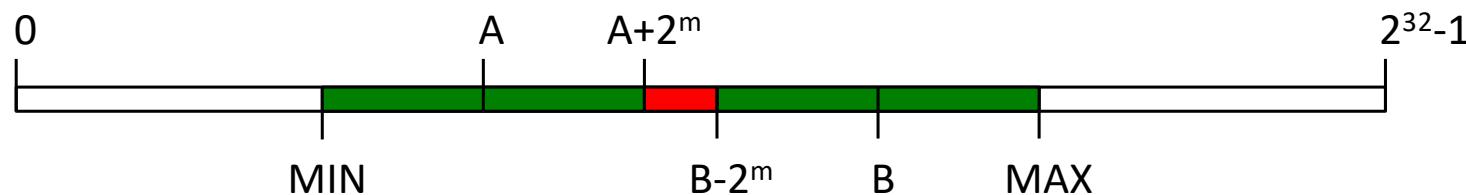
# Algoritmos de Compresión/Decompresión



$$A = \text{MIN} + 2^m$$

$$B = \text{MAX} - 2^m$$

hilore



- Para cada dato se guardan 2 bits (MIN,A,B,MAX) y un desplazamiento, que ahora puede ser más pequeño que antes.



# Algoritmos de Compresión/Decompresión

- El tipo de compresión a aplicar depende de los tipos de datos a tratar :
  - Z-buffer
  - Color
  - ...
- Si los datos a tratar son color, hay que hacer algo más



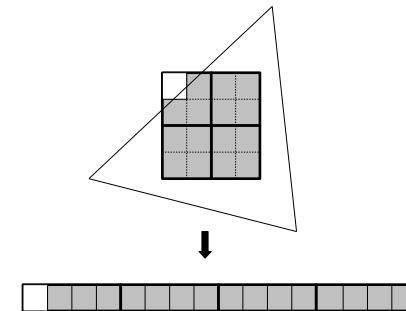
hilorebi



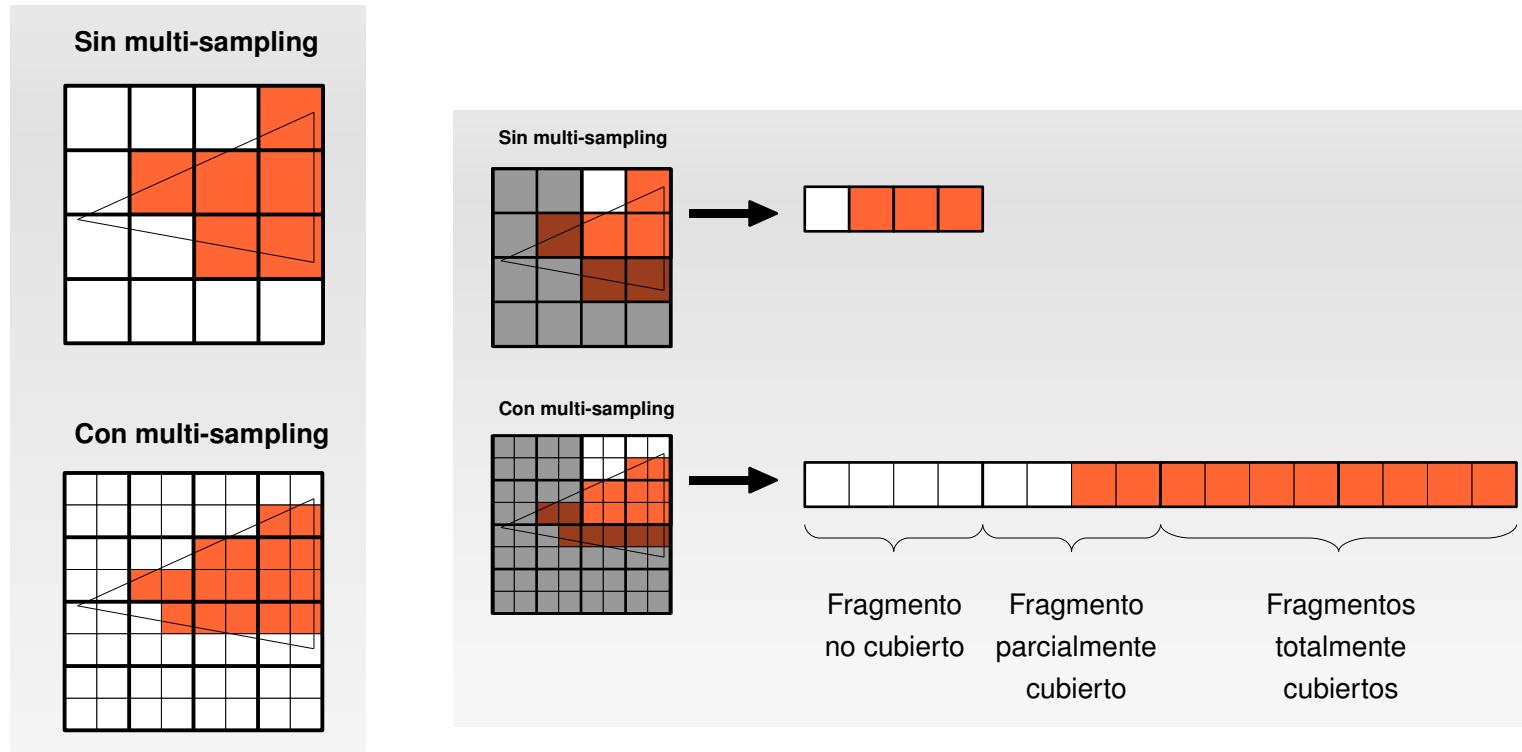
# Algoritmos de Compresión/Decompresión

## MultiSampling Compression

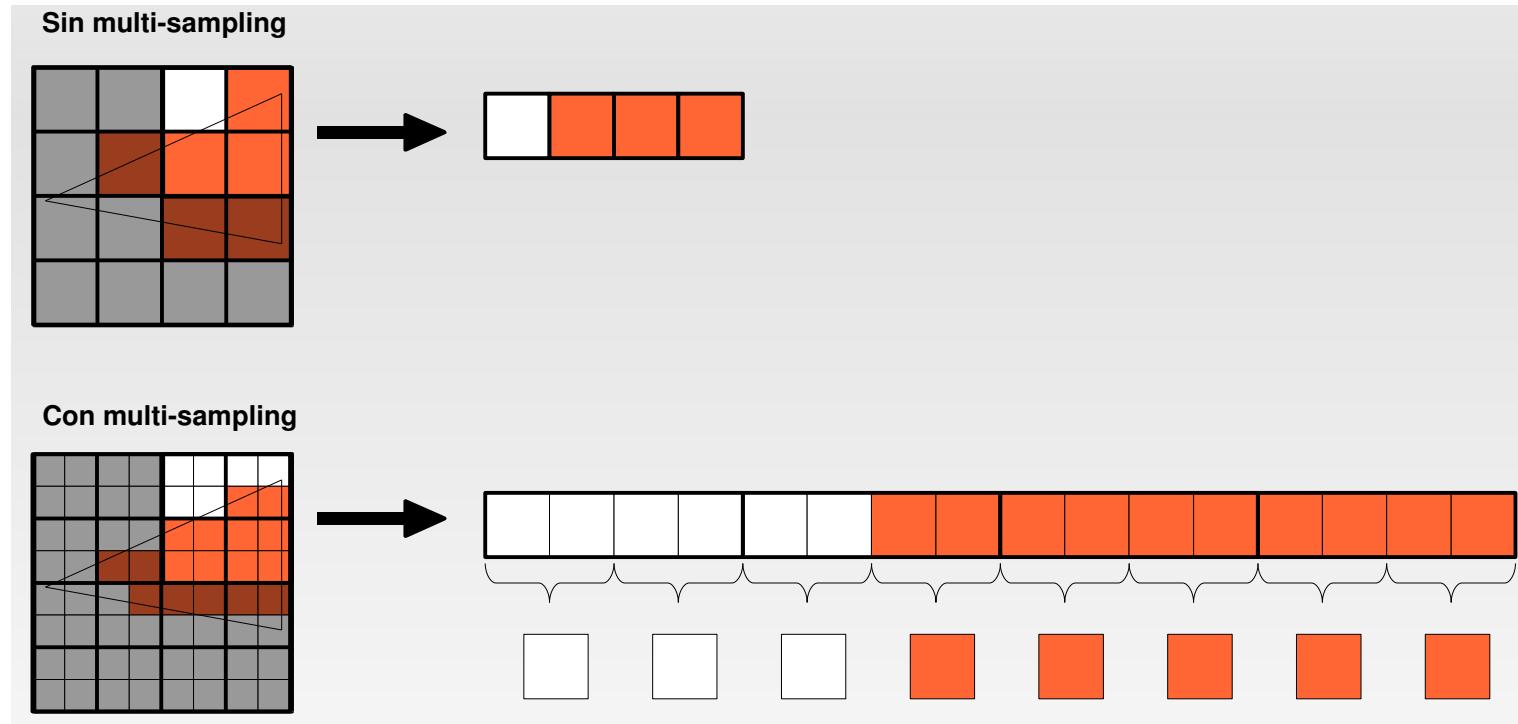
- Cuando se aplica multisampling es más probable que todas las muestras de 1 fragmento sean iguales.
- Otro caso típico es cuando hay 2 colores bien diferenciados en 1 fragmento.
- El algoritmo consiste en buscar subbloques que cumplan las condiciones anteriores.
- Hay que definir
  - Tamaño de los subbloques
  - Cuántas referencias se toman por subbloque
- Se almacena:
  - Las referencias
  - Para cada subbloque los índices a los subbloques.



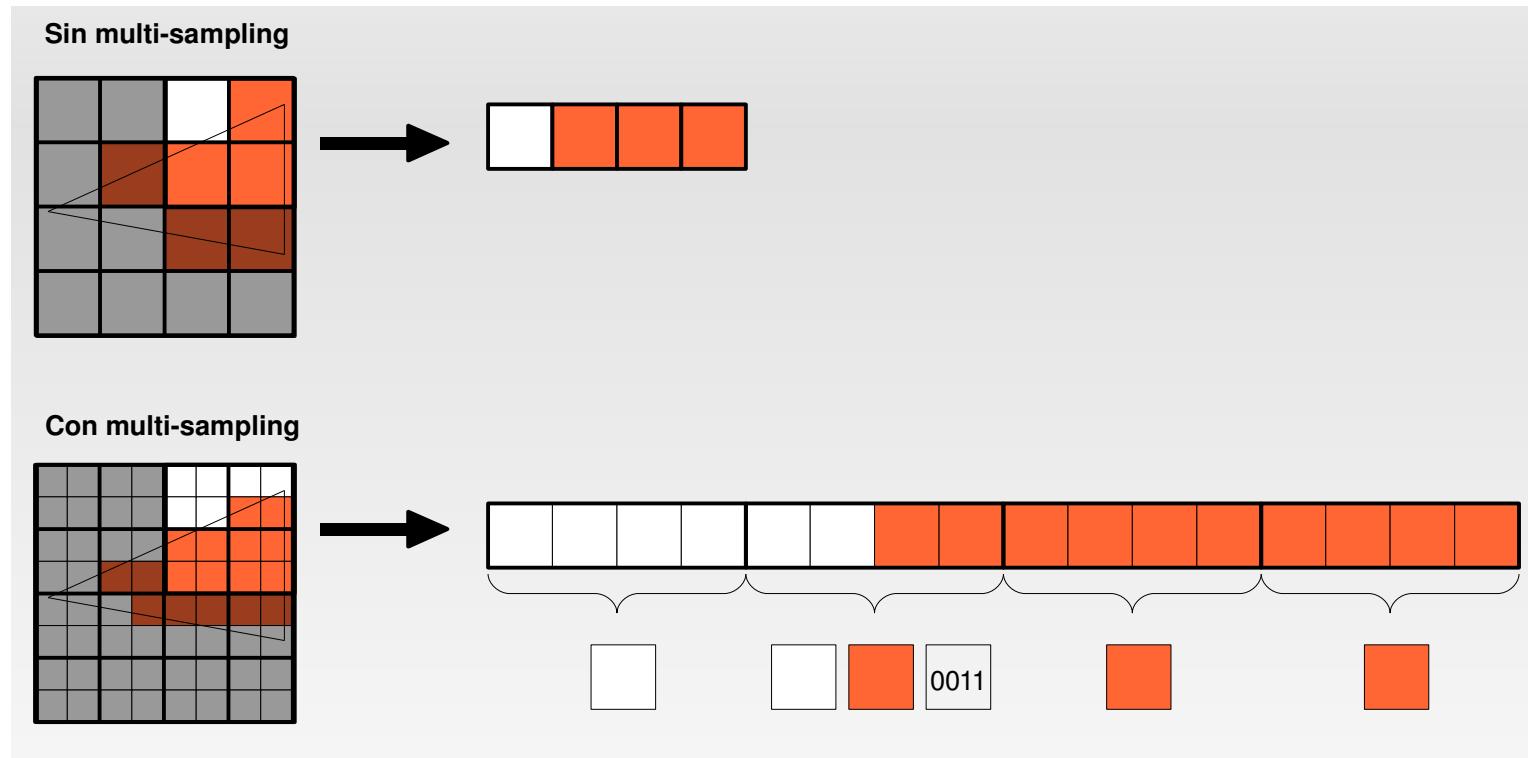
# Algoritmos de Compresión/Decompresión



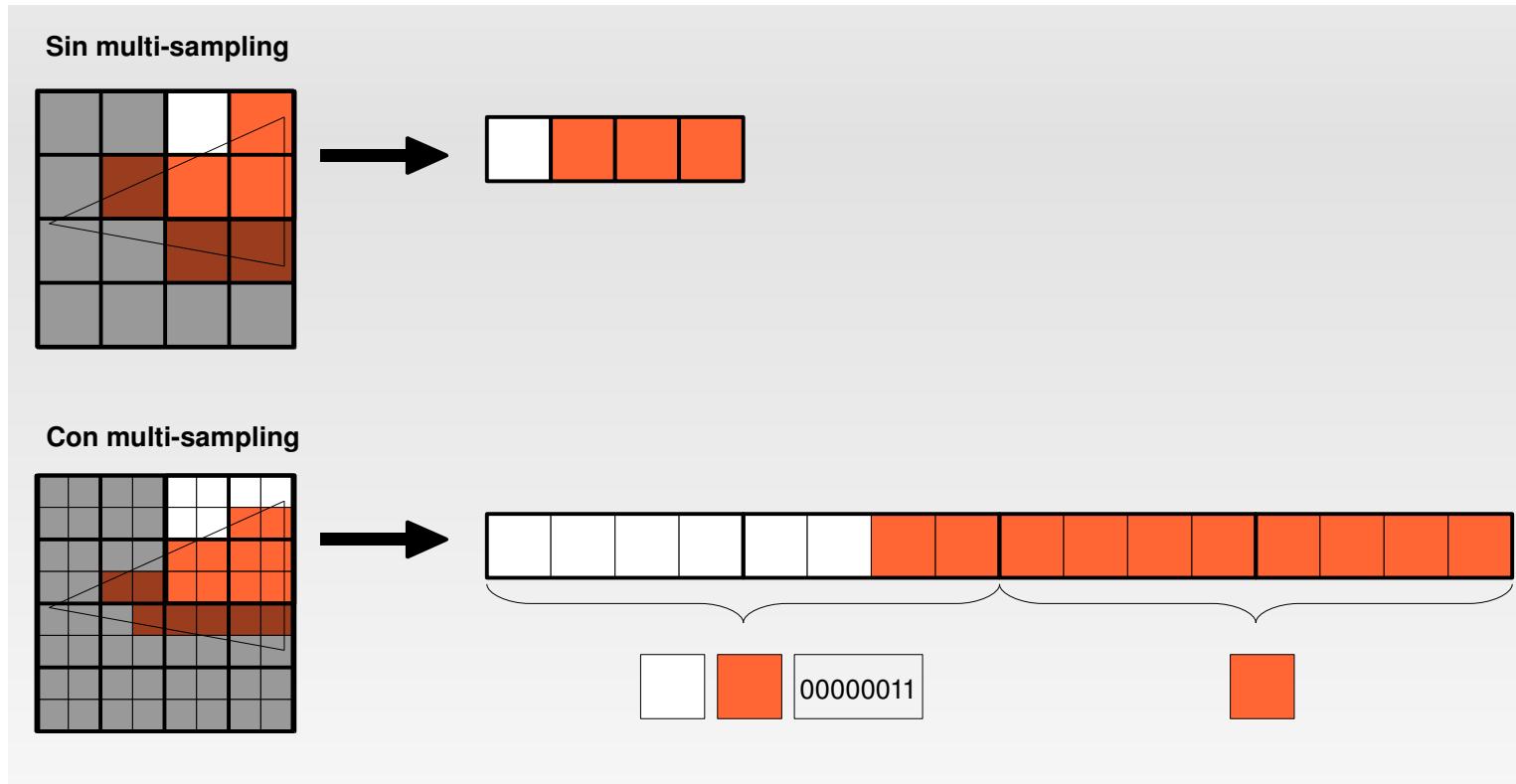
# Algoritmos de Compresión/Decompresión



# Algoritmos de Compresión/Decompresión



# Algoritmos de Compresión/Decompresión



# Algoritmos de Compresión/Decompresión

## Resultados experimentales

- GPU: Attila
- Juego: Prey
- MultiSampling: x4
- Líneas de cache de 256B, transacciones memoria: 64B

| Nivel  | nada | hilo  |       | hilore |       | hilorebi |       | multiSamp |       |
|--------|------|-------|-------|--------|-------|----------|-------|-----------|-------|
| 64     | 0    | 11,71 | 51%   | 12,53  | 54,6% | 12,48    | 54,4% | 11,78     | 51,3% |
| 128    | 0    | 2,35  | 5,1%  | 7,09   | 15,5% | 7,49     | 16,3% | 19,74     | 43%   |
| 192    | 0    | 28,49 | 41,4% | 14,21  | 20,6% | 15,48    | 22,5% | 3,67      | 5,3%  |
| 256    | 91,8 | 2,28  | 2,5%  | 8,54   | 9,3%  | 6,25     | 6,8%  | 0,32      | 0,4%  |
| R/W GB | 91,8 | 44,82 |       | 42,37  |       | 41,7     |       | 35,51     |       |

← Máximo nivel de compresión  
← Sin compresión



# Lectura Complementaria

□ Lance Williams

“Pyramidal Parametrics”

SIGGRAPH, 1983

□ Homan Igehy, Matthew Eldridg and Kekoa Proudfoot

“Prefetching in a Texture Cache Architecture”

Graphics Hardware 1998

□ James F. Blinn

“Simulation of Wrinkled Surfaces”

Computer Graphics, Vol. 12(3), pp286-292 SIGGRAPH-ACM (Aug 1978)





UNIVERSITAT POLITÈCNICA DE CATALUNYA  
BARCELONATECH

Departament d'Arquitectura de Computadors

# Tarjetas Gráficas y Aceleradores

## Texturas y Antialiasing

Agustín Fernández

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya

