Texturas. Sampling.

<a.soroa@ehu.eus>

EHU

Texture mapping

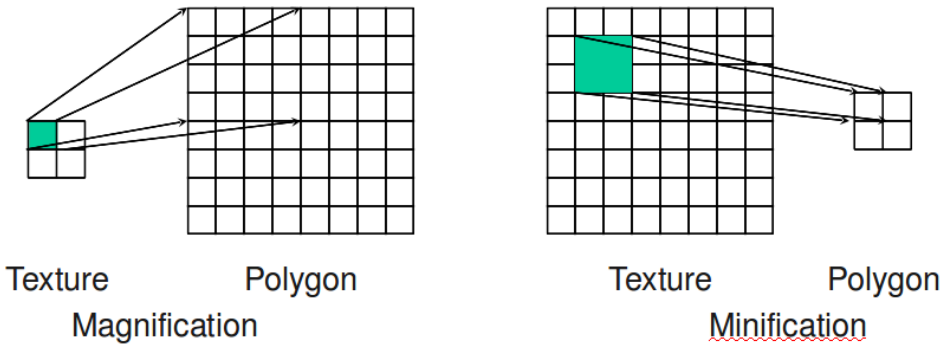
- Para asignar una imágen 2D en un objeto se distinguen dos fases:
  - función de *mapping*, que asigna una coordenada (u, v) a cada vértice:
    - $f(x, y, x) \rightarrow (u, v)$
  - función de *sampling*, que dado una coordenada (u, v) devuelve un color:
    - $g(u, v) \rightarrow (r, g, b)$

Función de *sampling*

- Los vértices del triángulo tienen asignadas unas coordenadas de textura.
- El triángulo ocupa una serie de fragmentos en la pantalla.
- Cada fragmento tiene una coord. de textura (u, v) (por interpolación).
  - Qué color asignar al fragmento?

Función de *sampling*

- A veces (u, v) concuerda completamente con la posición de un *texel*
- pero en general (u, v) no coincide exactamente con un *texel*
  - Si el área del fragmento es mas *pequeña* que el *texel*: ***magnification***
  - Si el área del fragmento es mas *grande* que el *texel*: ***minification***



Magnification.

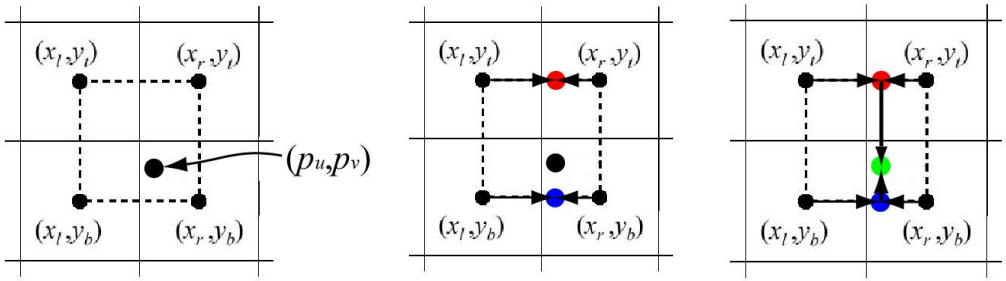
- El área del fragmento es menor que el *texel*: $\frac{\#\text{texel}}{\#\text{fragment}} < 1$
  - Los *texel* se verán muy grandes.
- Dos técnicas anti-aliasing:
  - *Nearest neighbor*: utilizar el *texel* más cercano.
  - *Bilinear interpolation*: interpolar los *texels* más cercanos.

Nearest neighbor

- La técnica más rápida.
- En general, no produce buenos resultados.
  - Líneas dentadas (*jagged*)
- Coordenada de tectura: $(u, v) \in [0, 1]$
- $(m \times n)$ texels
- El texel más cercano: $(\lfloor m \cdot u \rfloor, \lfloor n \cdot v \rfloor)$

Bilinear interpolation

- Idea: calcular el color como combinación lineal de los cuatro *texel* más cercanos.
- En general da buenos resultados para resolver el *magnification*



Bilinear interpolation

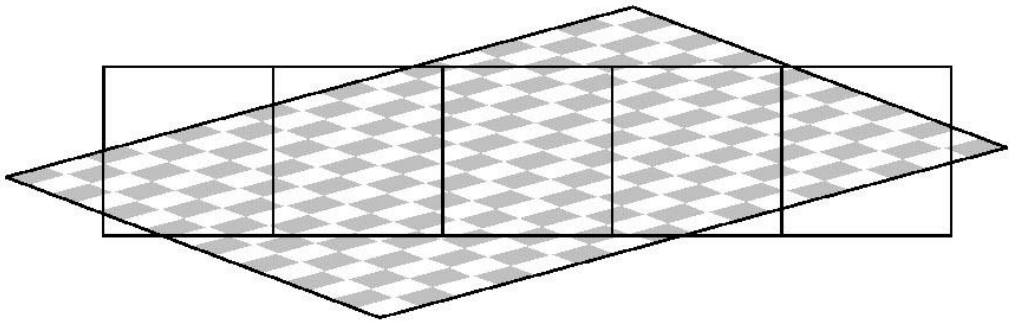
Formulación:

- $\mathbf{t}(i,j)$: color del texel en $(i,j) \in \mathcal{N}, 1 \leq i \leq n, 1 \leq j \leq m$
- $\mathbf{b}(u,v)$ color asignado a la coor. de textuxa (u,v)
- $x_l = \lfloor p_u \rfloor$; $x_r = \lfloor p_u + 1 \rfloor$;
- $y_b = \lfloor p_v \rfloor$; $y_y = \lfloor p_v + 1 \rfloor$;
- $(u', v') = (p_u - \lfloor p_u \rfloor, p_v - \lfloor p_v \rfloor)$

$$\begin{aligned} \mathbf{b}(u,v) \quad = \quad & (1-u')(1-v')\mathbf{t}(x_l, y_b) + \\ & u'(1-v')\mathbf{t}(x_r, y_b) + \\ & (1-u')v'\mathbf{t}(x_l, y_t) + \\ & u'v'\mathbf{t}(x_r, y_t) \end{aligned}$$

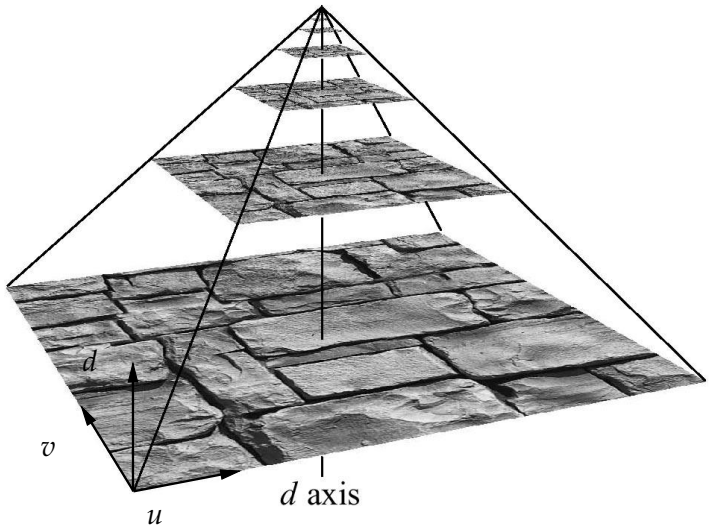
Minification

- El área del fragmento es mayor que la del *texel*: $\frac{\#\text{texel}}{\#\text{fragment}} > 1$.
- Problema más difícil que el Magnification
  - *Nearest neighbor*: resultados muy pobres.
  - *Bilinear interpolation*: no da buenos resultados.
- Técnica más usada: *MipMapping*



Mipmapping

- Se crea una pirámide con la imagen.
  - El nivel 0 ($d = 0$) corresponde a la imagen original.
  - La imágen del nivel $i + 1$ es la mitad de la imagen del nivel i.



Mipmapping. Trilinear interpolation.

- Se calcula el valor de d para un fragmento:
  - teniendo en cuenta $\frac{\#\text{texel}}{\#\text{fragment}}$.
  - OpenGL lo hace automáticamente.
- Se calcula i tal que $i < d < i + 1$
- Se calcula la interpolación bilineal en i y en $i + 1$
- Se interpolan los valores para obtener el valor deseado.