Lecture #12

Texture Aliasing and Antialiasing

Computer Graphics

Winter Term 2016/17

Marc Stamminger / Roberto Grosso

Texture Mapping

- Up to now
  - for each pixel, we interpolate texture coordinates → (s,t)
  - fragment shader reads pixel at this position and uses this color

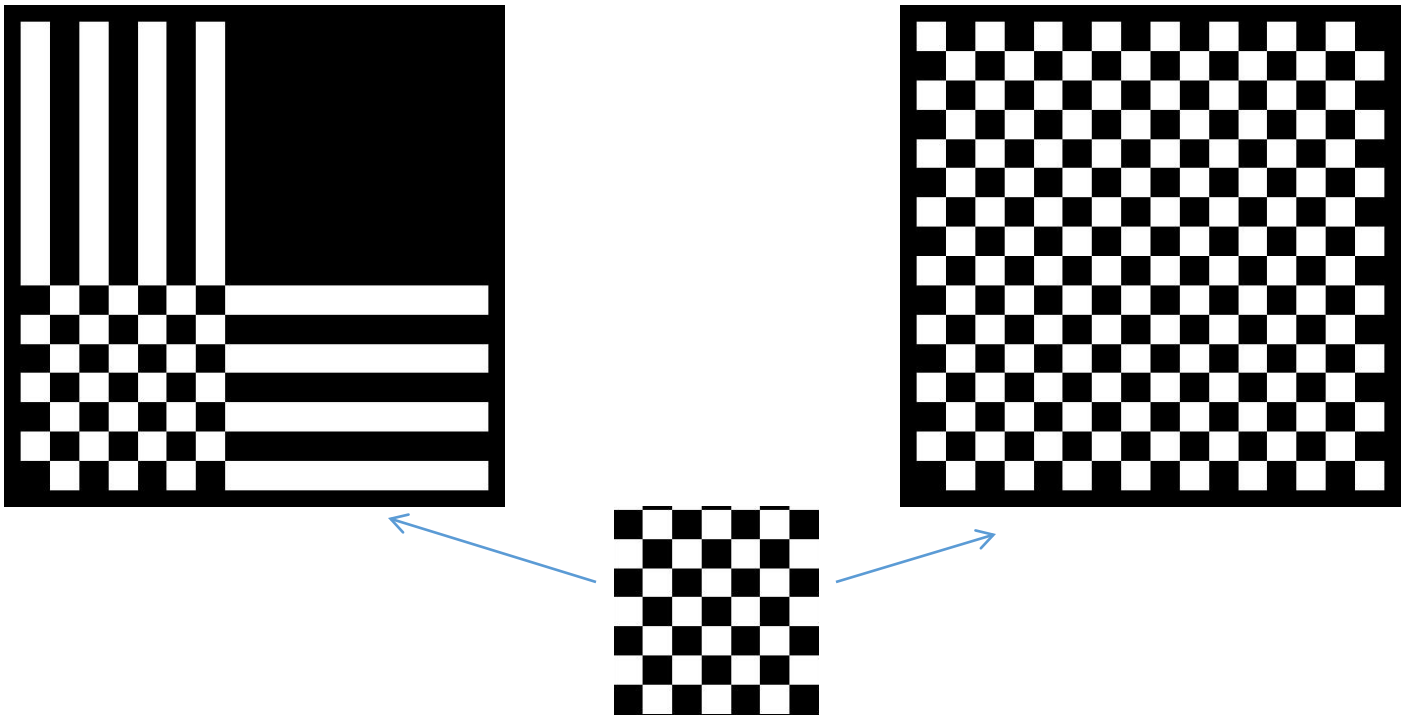
- But: how to map (s,t) to corresponding texture value?
  - Issue #1: how to handle values outside [0,1] ?
  - Issue #2: how to handle value between pixel centers ?

Texture Mapping

- Issue #1: how to handle values outside [0,1] ?

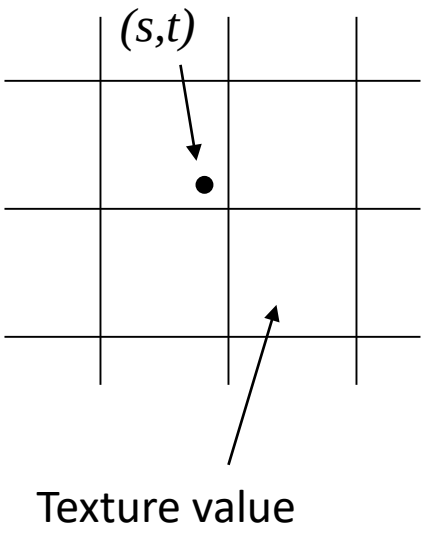
- **Clamping:**
  $(s,t) \rightarrow (clamp(s), clamp(t))$
  $clamp(x) = \max(0, \min(1, x))$

Repeating:
$(s,t) \rightarrow (frac(s), frac(t))$
$frac(x) = x - floor(x)$



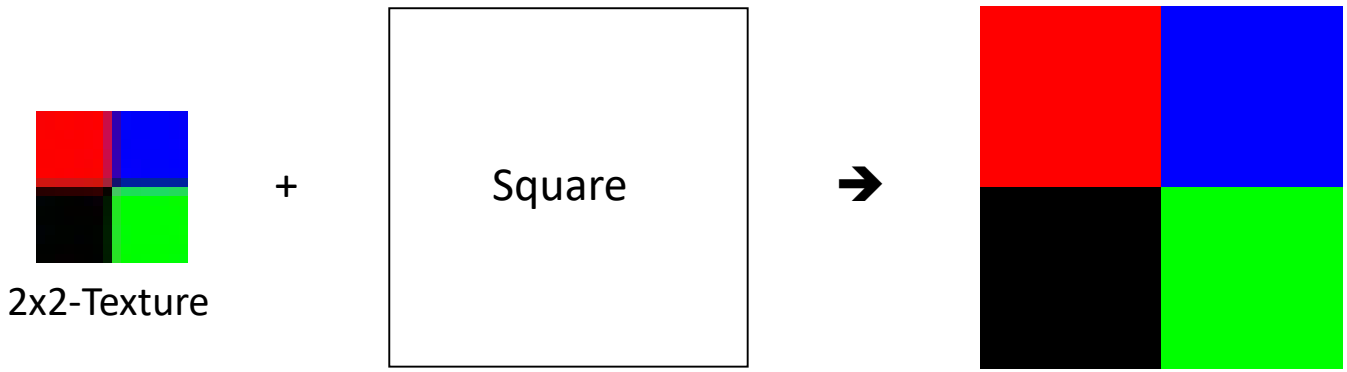
Texture Mapping

- Issue #2: how to handle value between pixel centers ?
  - Nearest neighbor
  - Bilinear interpolation
  - Trilinear interpolation (see later)



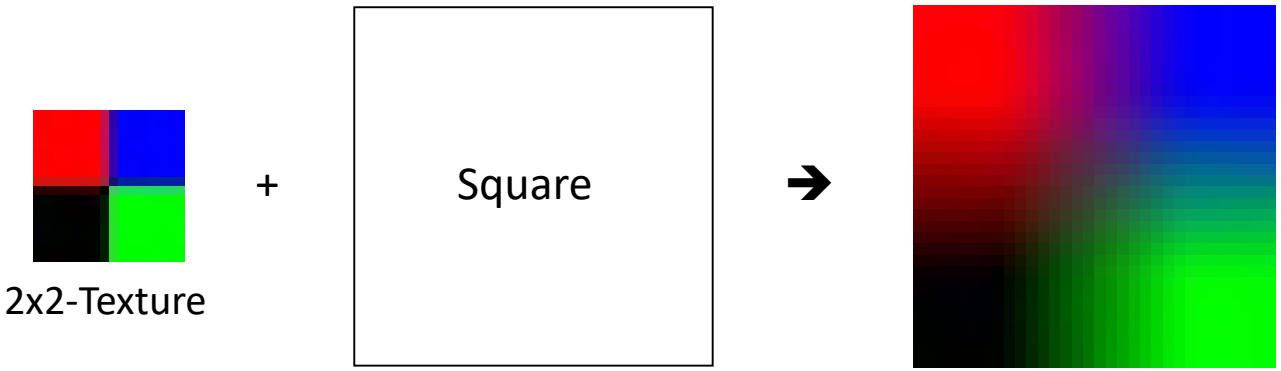
Texture Interpolation

- Texture value interpolation
  - “Nearest Neighbor”
    - Treat found image pixel as constant colored rectangular tile
    - $c(s,t) = c_{ij}; i = \lfloor sn_x \rfloor; j = \lfloor tn_y \rfloor$



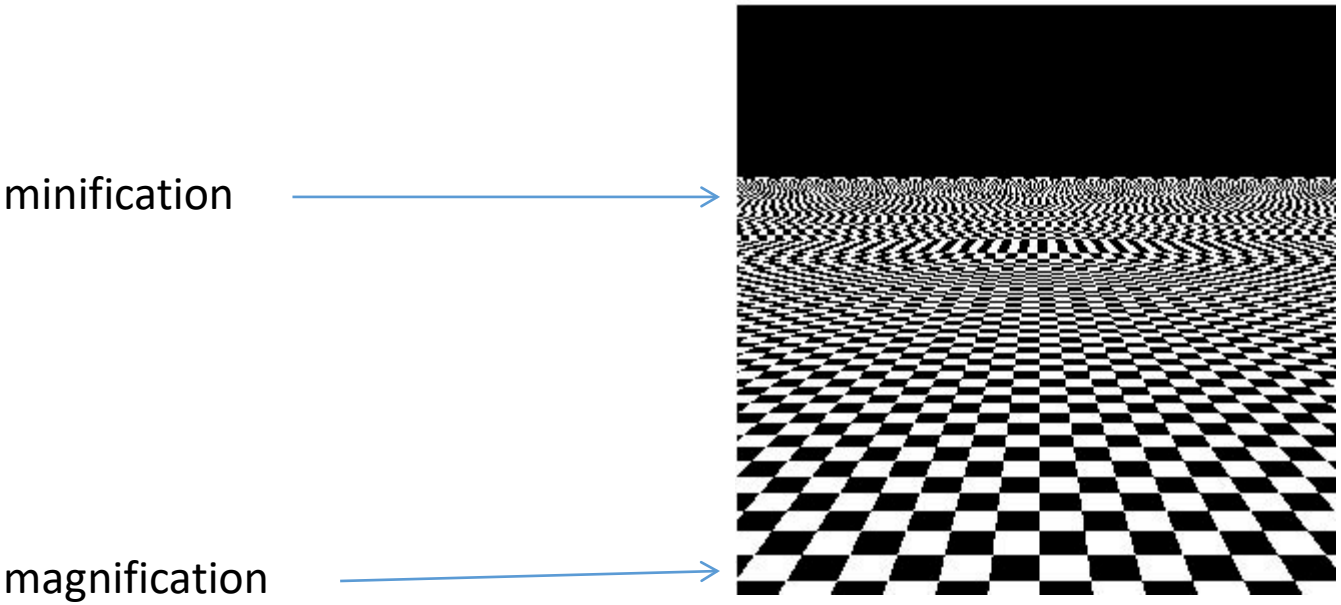
Texture Interpolation

- Texture value interpolation
  - Bilinear interpolation: Color information of image pixel is mixed with neighboring tiles, depending on relative distances
  - $c(s,t) = (1-s')(1-t')c_{ij} + s'(1-t')c_{(i+1)j} + (1-s')t'c_{i(j+1)} + s't'c_{(i+1)(j+1)}$
  - $s' = sn_x - \lfloor sn_x \rfloor; t' = tn_y - \lfloor tn_y \rfloor$



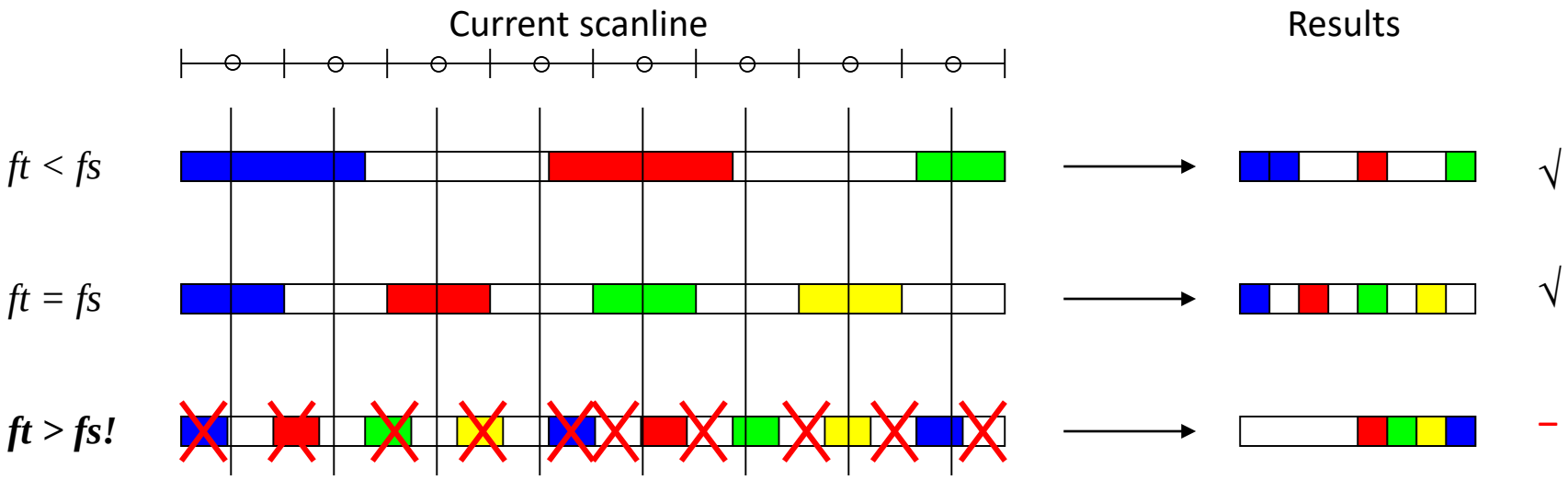
Texture Interpolation

- Effects of nearest neighbor vs. bilinear filtering become visible for **texture magnification**
- Another problem appears for **texture minification -> Aliasing**



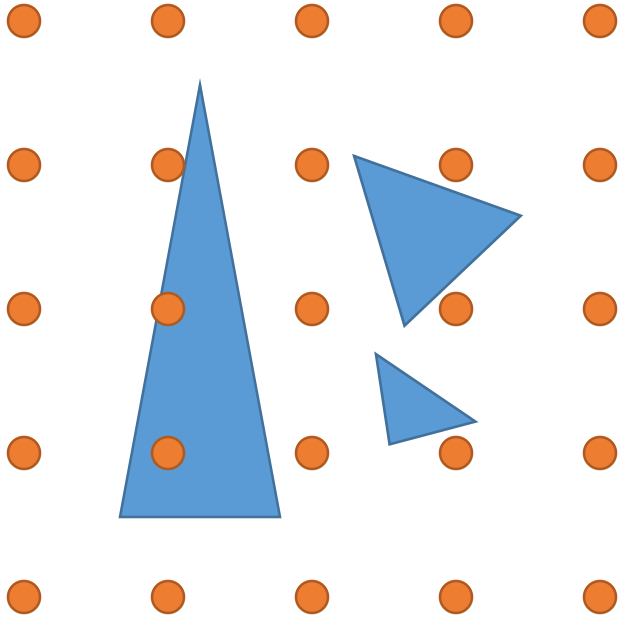
Texture Interpolation

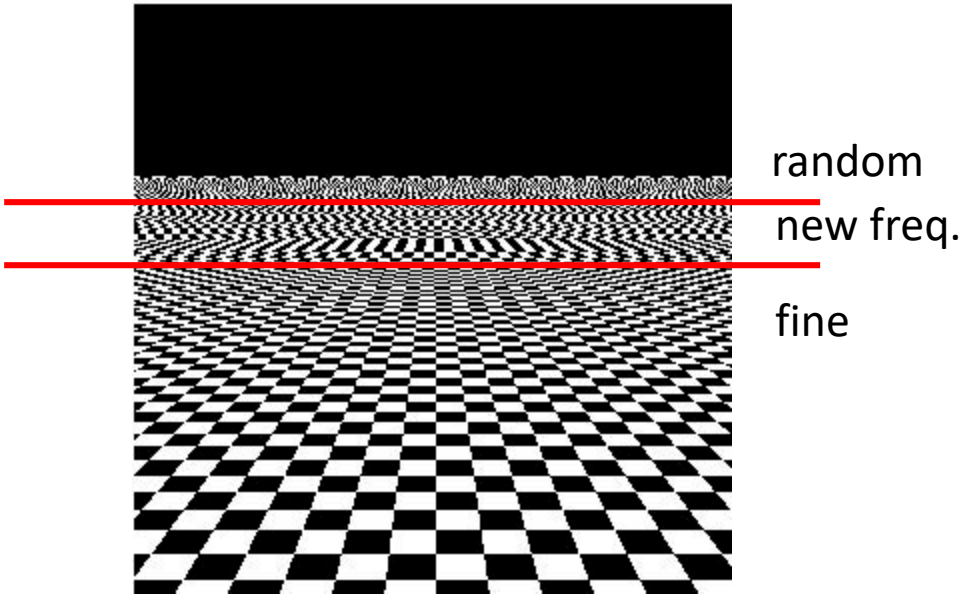
- Sampling problem



Aliasing

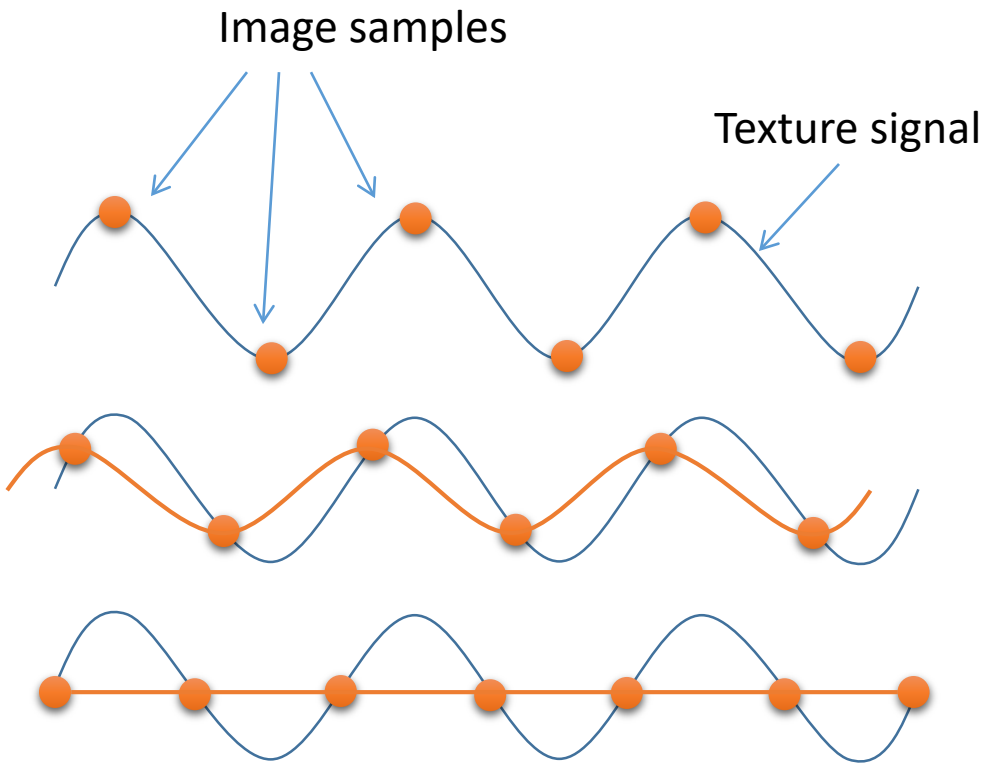
- Sampling problems:
  missing small objects appearance of new frequencies/ flickering





Aliasing

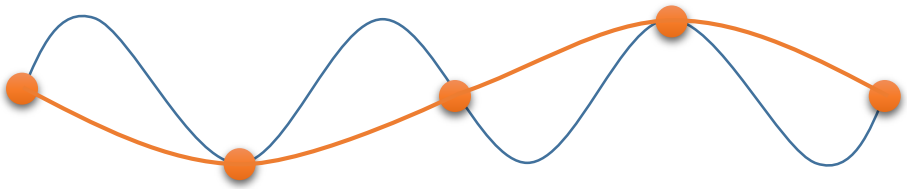
- Closer look:
- Image frequency = texture frequency
- Just by moving texture, sampled result becomes weaker and can even disappear!

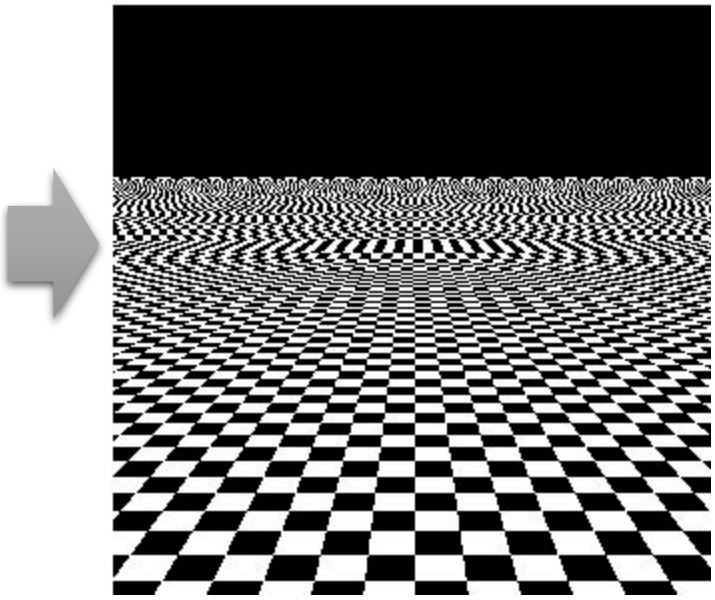


Aliasing

- Go further:
  Image frequency < texture frequency (= minification)

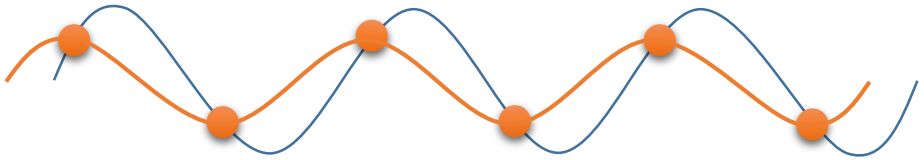
- A new, lower frequency appears!



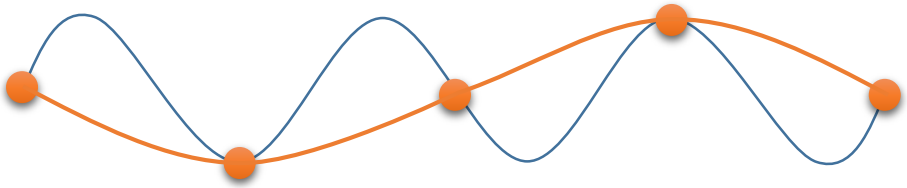


Aliasing

- Nyquist / Shannon
- Signal can only be reconstructed if sampling frequency is at least twice as high as signal frequency

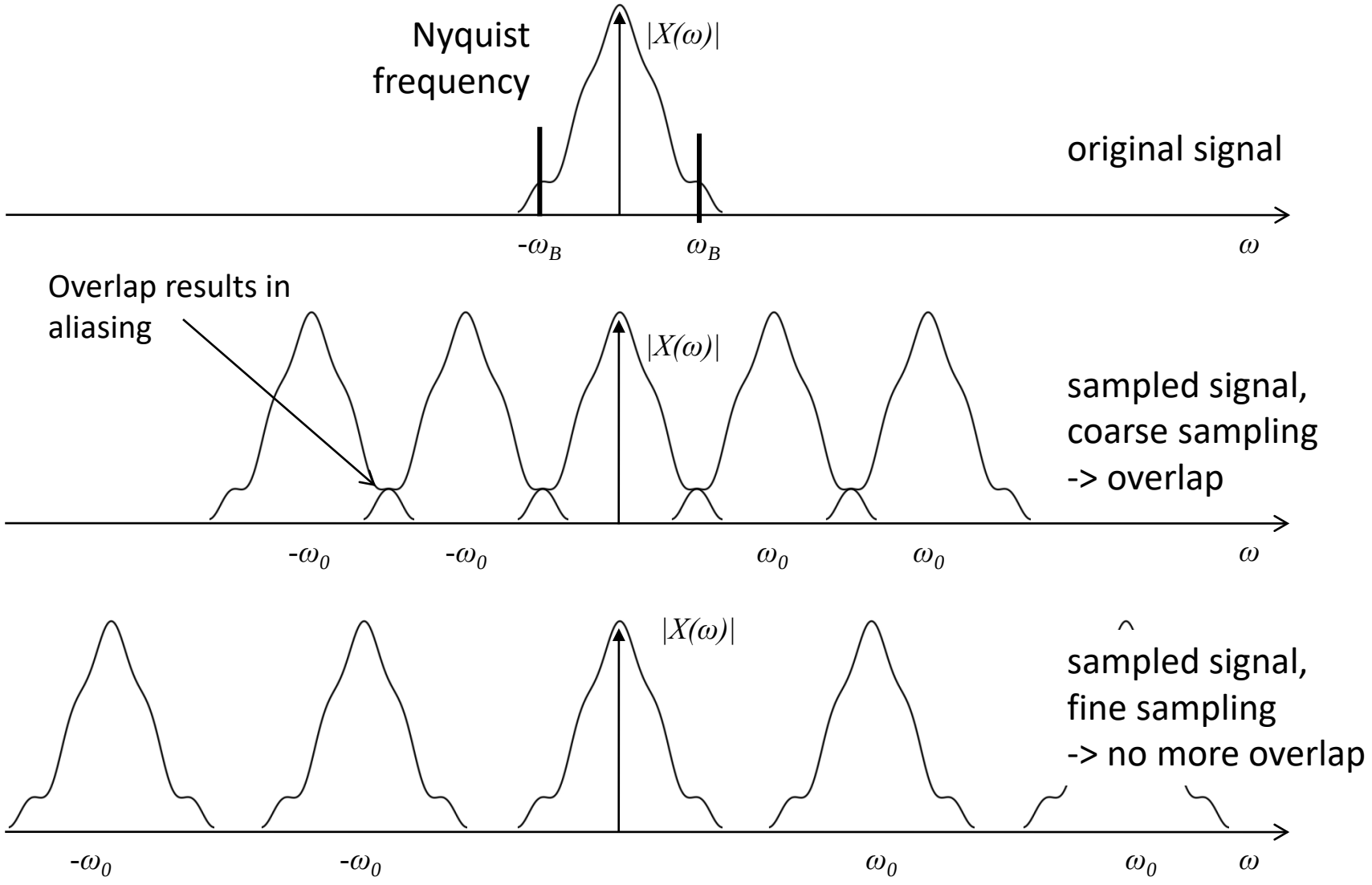


just Nyquist frequency



below Nyquist frequency

Aliasing



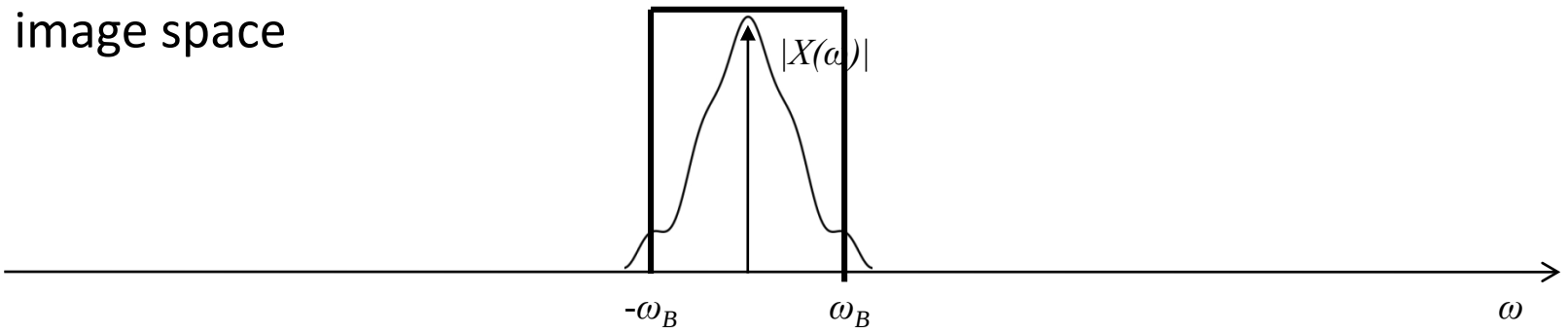
Finer sampling = no more aliasing

Aliasing

- So we must filter out frequencies beyond Nyquist
- Multiplication with box filter in frequency space = folding with

$$sinc(x) = \frac{\sin(x)}{x}$$

in image space

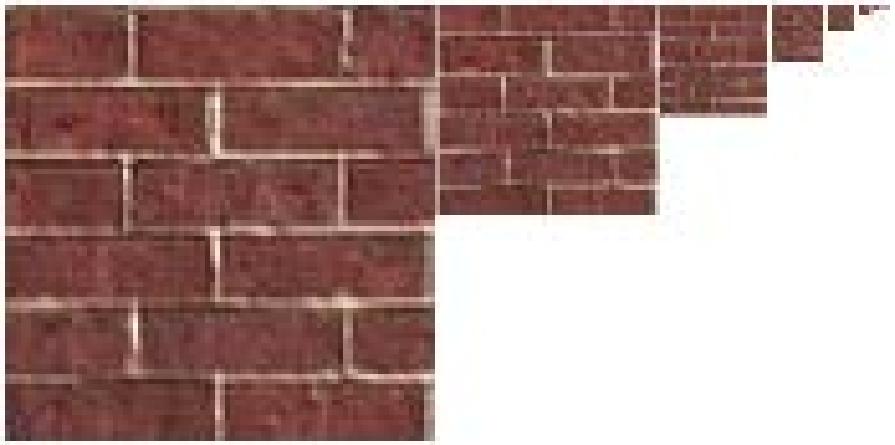


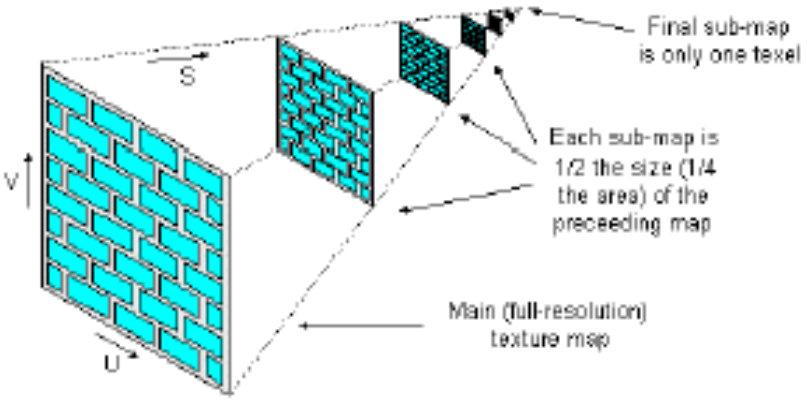
- But: $sinc$ has infinite support → Value of a pixel depends on entire image!
- More practical alternative: Gauß-Filter with finite support
  - standard deviation about one pixel
  - cut off at certain radius → finite support

Texture Interpolation

- But with textures, the relation texture resolution to screen resolution is not known in advance, and varies over the image…

- Solution: Mip-Mapping
  - Generate a hierarchy of lower-resolution textures from original texture
    → each texture is a filtered version of the previous one with double filter size
    → hierarchy of prefiltered versions
  - Use bilinear interpolation or other integration technique to create textures

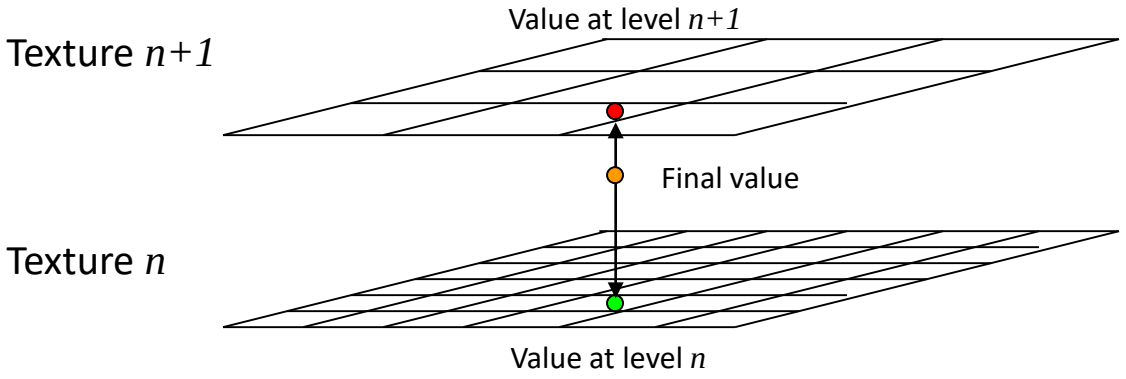




Texture Interpolation

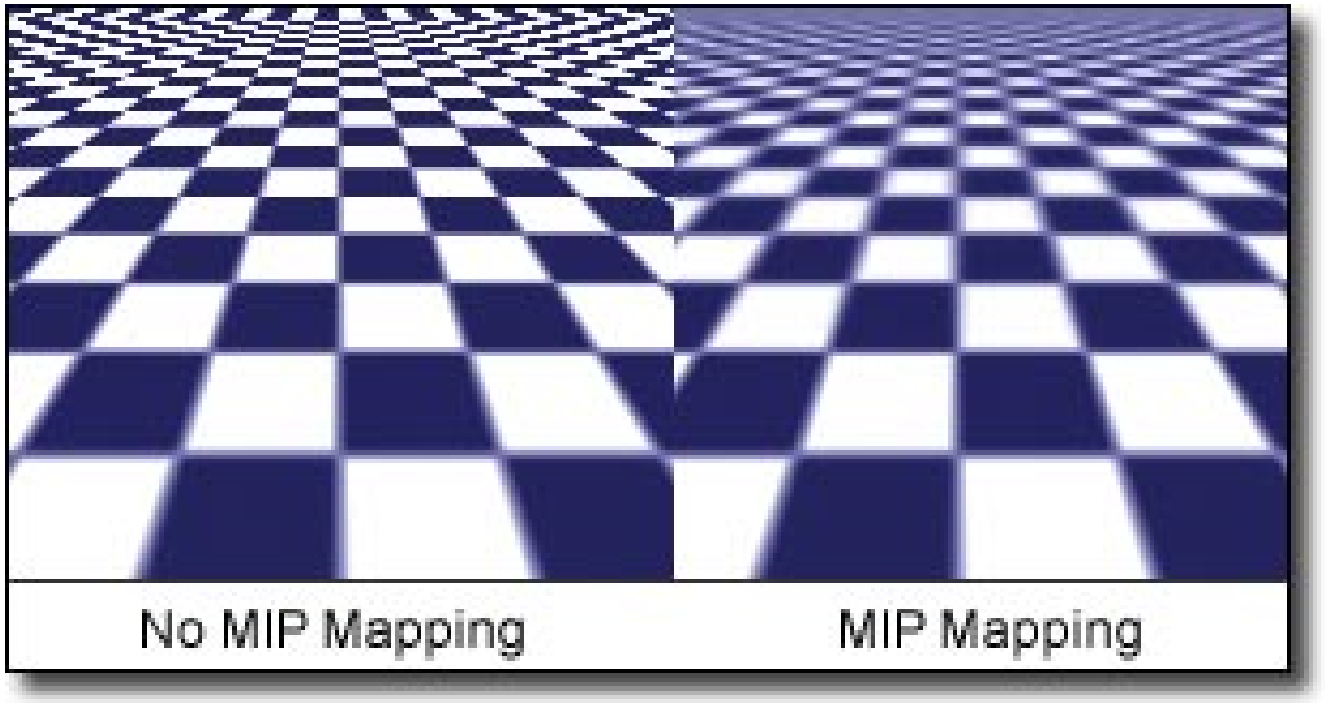
- At render time for a particular pixel:
  - determine which texture level of the mip map is the best for this pixel
  - read this corresponding texture
    → at the transition between different levels, a seem gets visible
    → interpolate between levels

- -> Trilinear interpolation
  - Possible with mip-mapping
  - Idea: Bilinear interpolation at (s,t) in two succeeding textures from the mipmap hierarchy, then linear interpolation between these two values



Texture Interpolation

- Texture value interpolation
  - Trilinear interpolation: Example



Texture Interpolation

- Mipmap filtering: 4 possible combinations according to interpolation (filtering) within a level and between levels of resolution.

GL constant	Description
GL_NEAREST_MIPMAP_NEAREST	Select nearest mipmap level and perform nearest interpolation.
GL_NEAREST_MIPMAP_LINEAR	Linear interpolation between mipmap levels and nearest neighbor filtering
GL_LINEAR_MIPMAP_NEAREST	Select nearest mipmap level and perform linear filtering
GL_LINEAR_MIPMAP_LINEAR	Linear interpolation between levels and linear filtering