COMP3421

Textures

Robert Clifton-Everest

Email: robertce@cse.unsw.edu.au

Debugging Meshes

- Draw only the outline

```
glPolygonMode(GL.GL_FRONT_AND_BACK,
GL3.GL_LINE);
```

- Turn off backface culling to see full mesh

```
gl.glDisable(GL.GL_CULL_FACE);
```

- If lighting looks weird, it's most likely a problem with the normals!

Texturing

- Textures are a way to add detail to our models without requiring too many polygons.
- Textures are used to add:
  - Colour
  - Reflections
  - Shadows
  - Bumps
  - Lighting effects
  - etc…

Textures

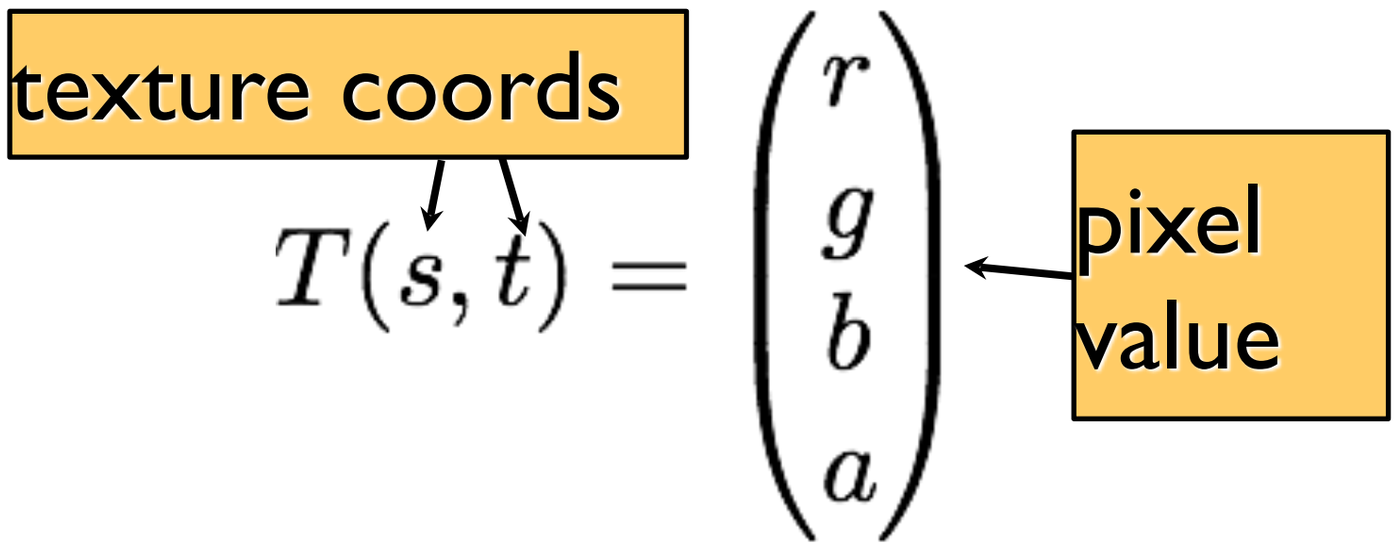
- A texture is basically a function that maps texture coordinates to pixel values.

$$T(s,t) = \begin{pmatrix} r \\ g \\ b \\ a \end{pmatrix}$$

- Texture coordinates are usually in the range (0,1).

Textures

- A texture is basically a function that maps texture coordinates to pixel values.



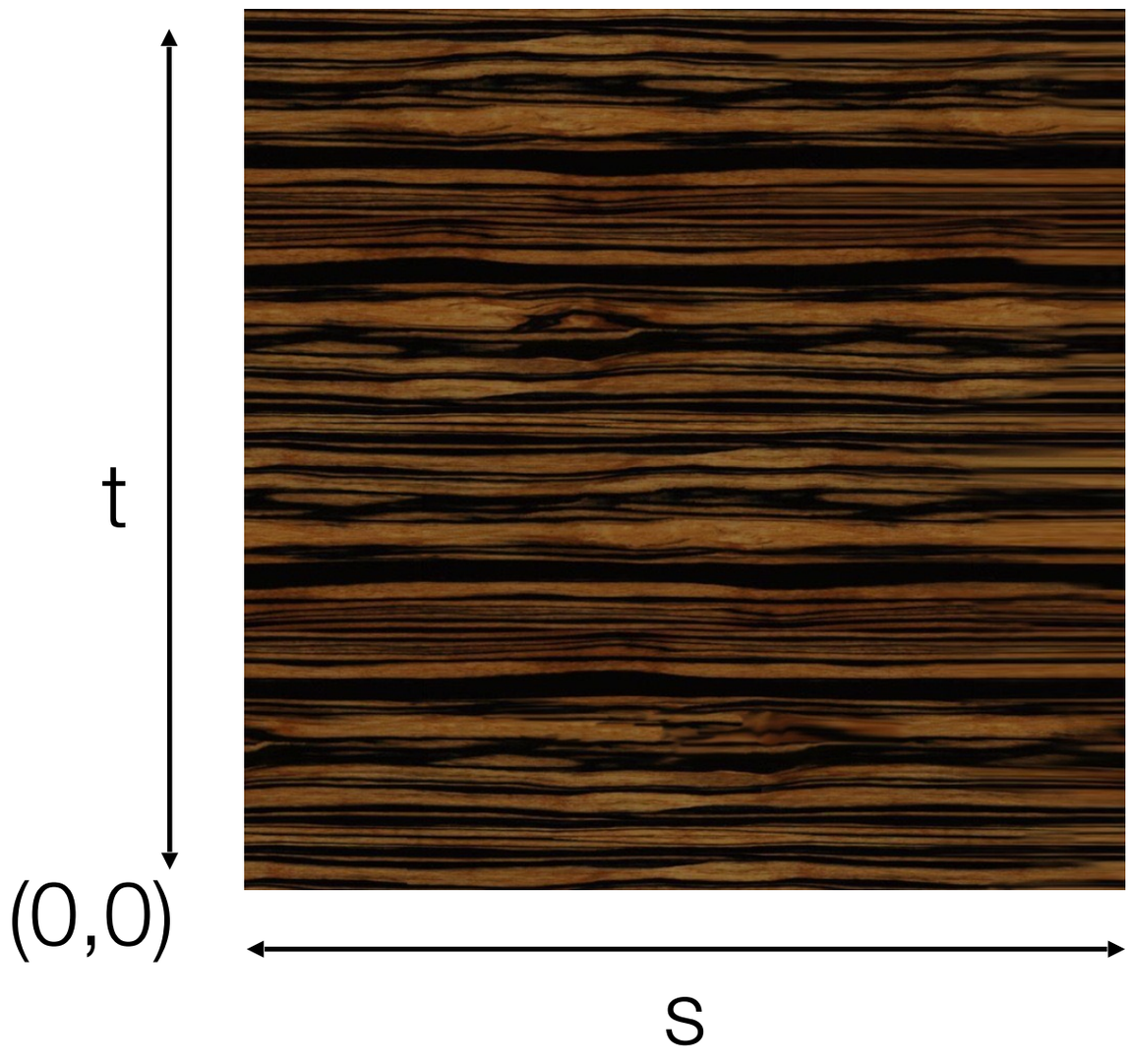
- Texture coordinates are usually in the range (0,1).

Textures

Textures are most commonly represented by bitmaps; i.e. 2D image files

T(s,t) = texel value at (s,t)

texel = pixel on a texture



Procedural textures

- It is also possible to write code to compute the texture value at a point.

- This can be good to generate materials like marble or woodgrain.

Using Textures

1. Load or creating textures
2. Passing the texture to a shader
3. Mapping texture co-ordinates to vertices

Loading textures in OpenGL

- Similar to vertex buffers, we have to create buffers on the GPU and copy into them.
- We can use JOGL to help us with that.
- See Texture.java

Loading textures in JOGL

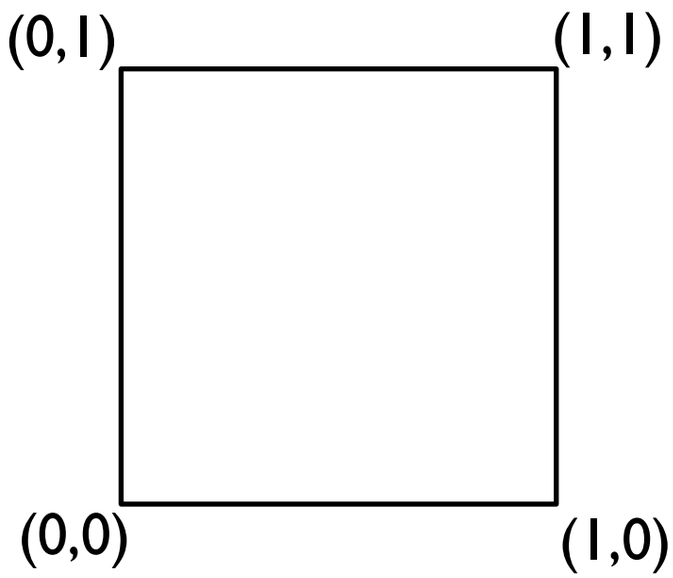
```
// Setting data to current texture
gl.glTexImage2D(
     GL.GL_TEXTURE_2D,
     0,// level of detail: 0 = base
     data.getInternalFormat(),
     data.getWidth(),
     data.getHeight(),
     0,  // border (must be 0)
     data.getPixelFormat(),
     data.getPixelType(),
     data.getBuffer());
```

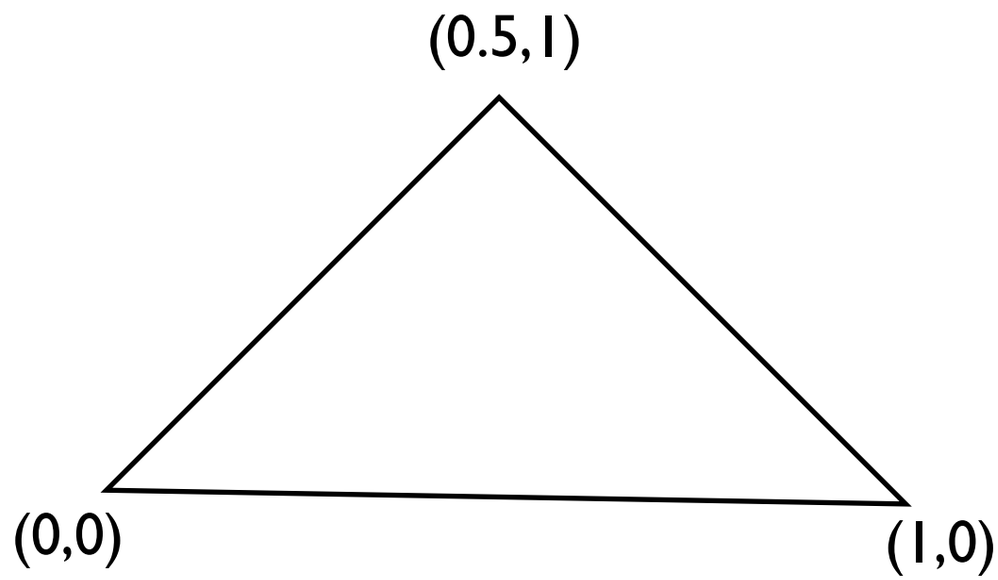
Using Textures

1. Load or creating textures ✔
2. Passing the texture to a shader
3. Mapping texture co-ordinates to vertices

Texture mapping

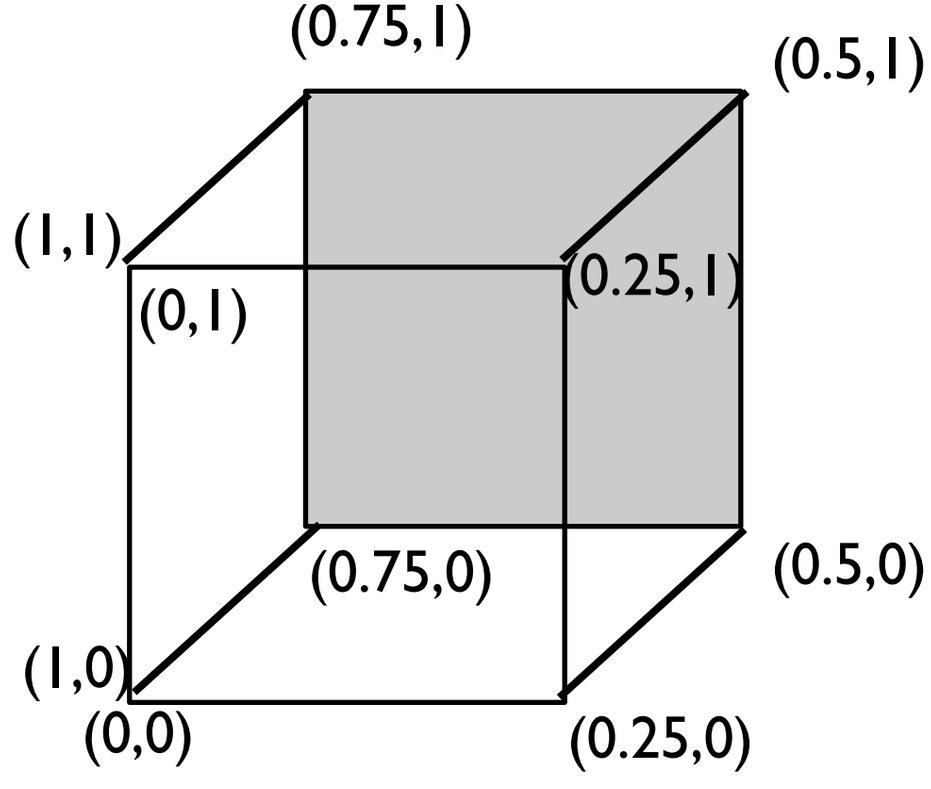
- To add textures to surfaces in on our model, we set texture coordinates for each vertex.





Texture mapping

- To add textures to surfaces in on our model, we set texture coordinates for each vertex.



Model Texture Mapping

- We can assign texture coordinates to vertices however we want. Complex models often have weird flattened textures.

Texture Wrap

- You can assign texture coordinates outside the range [0,1] and set the texture wrap to
  - GL.GL_REPEAT (default)
  - GL.GL_MIRRORED_REPEAT
  - GL.GL_CLAMP_TO_EDGE

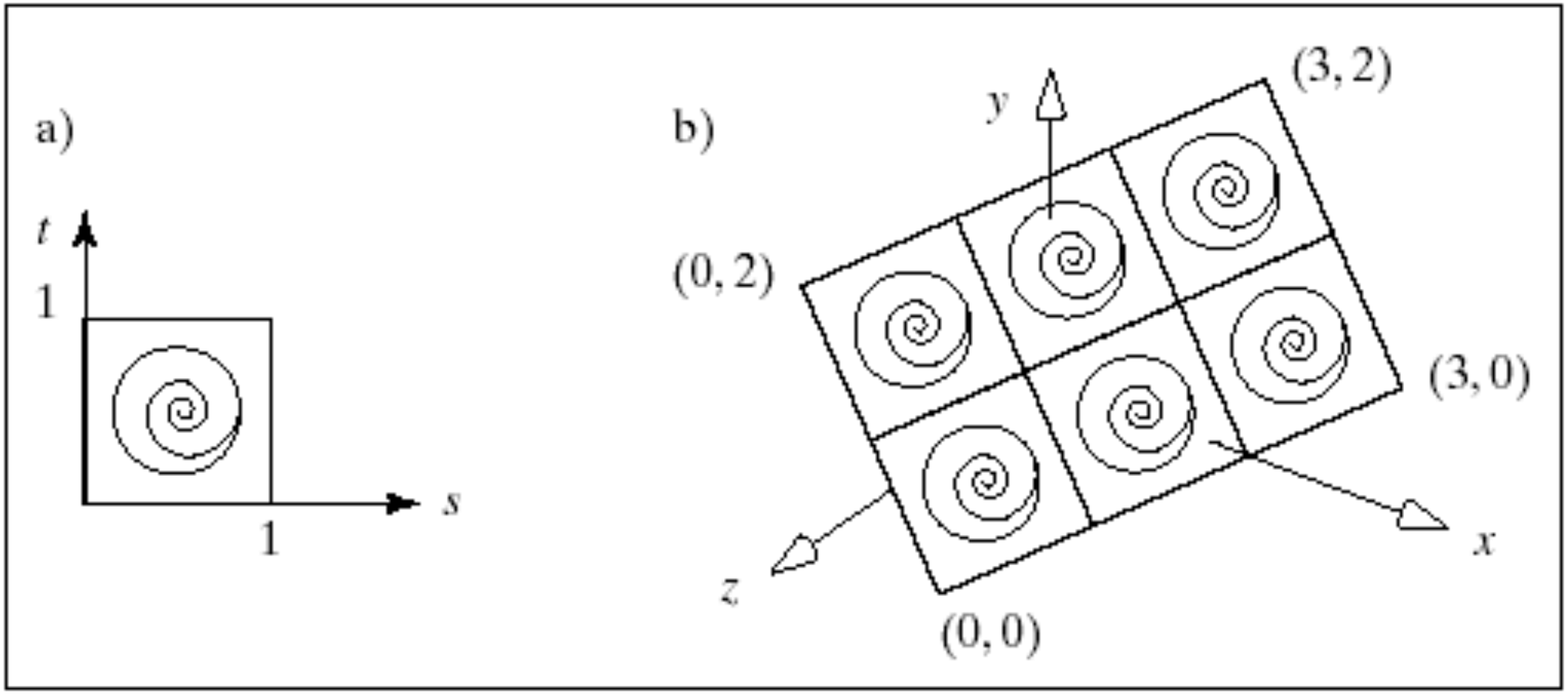
Texture WRAP

For example, setting to GL.GL_REPEAT in both s and t dimensions:

```
gl.glTexParameteri( GL.GL_TEXTURE_2D,
GL.GL_TEXTURE_WRAP_S, GL.GL_REPEAT);

gl.glTexParameteri( GL.GL_TEXTURE_2D,
GL.GL_TEXTURE_WRAP_T, GL.GL_REPEAT);
```

Repeating a Texture

- For example this shows the use of texture coordinates outside [0,1] that **repeat** the texture, if the setting is GL_REPEAT



Generating texture coordinates

- Unlike normals, there is no “natural” way to generate texture coordinates
- TriangleMesh will generate them, using functionality in JPLY, but it may not give the results you want.
- See ModelViewer

Using Textures

1. Load or creating textures ✔
2. Passing the texture to a shader
3. Mapping texture co-ordinates to vertices ✔

Binding the texture

- OpenGL supports up to 32 simultaneously "active" textures
  - Defined by constants GL_TEXTURE0..GL_TEXTURE31
- Note that these values are distinct from a texture id.

Binding the texture

- In the fragment shader we have:

```
uniform sampler2D tex;
```

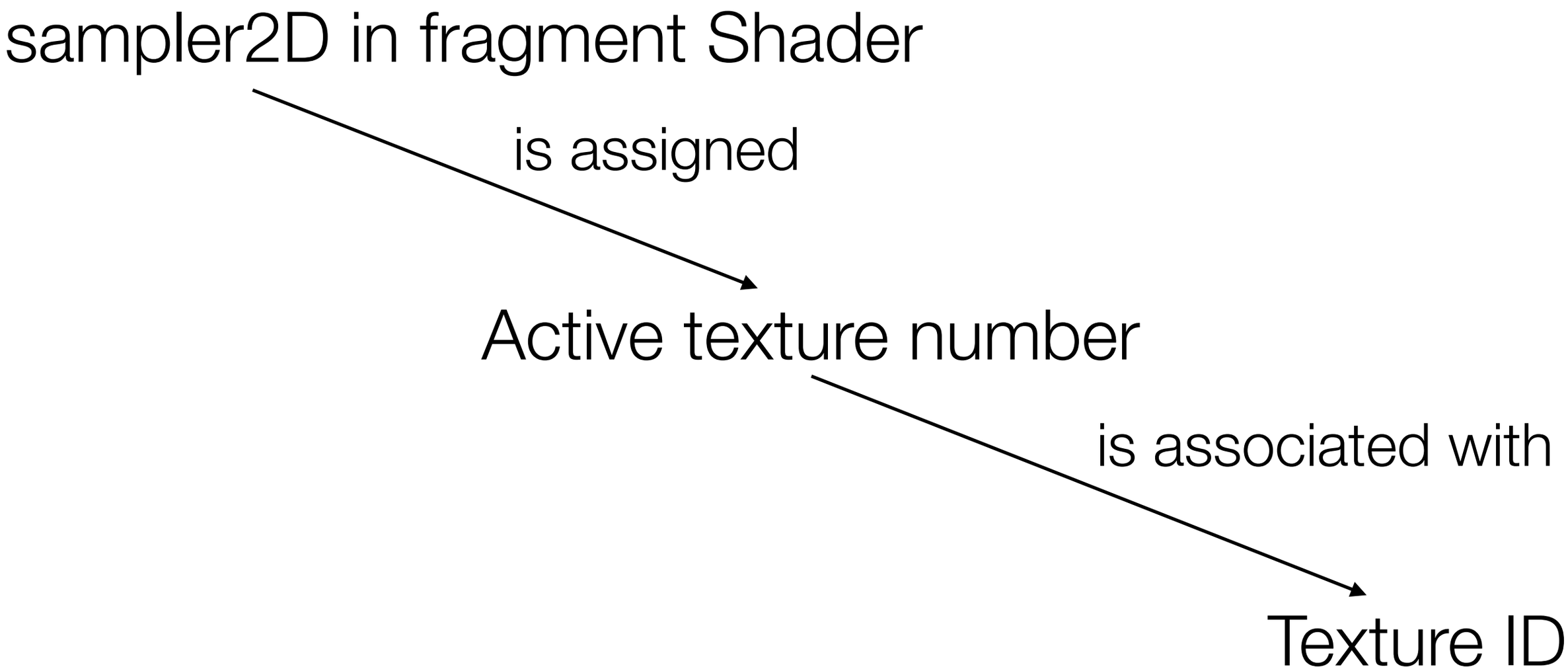
- We assign a texture to that with

```
Shader.setInt(gl, "tex", 0);
gl.glActiveTexture(GL.GL_TEXTURE0);
gl.glBindTexture(GL.GL_TEXTURE_2D, texId);
```

- See SimpleTextureExample.java

Binding the texture

- **In short:**



Textures and shading

- How do textures interact with shading?

- The simplest approach is to replace illumination calculations with a texture look-up.

$$I(P) = T(s(P), t(P))$$

- This produces objects which are not affected by lights or color.

Textures and shading

- A more common solution is to use the texture to modulate the ambient and diffuse reflection coefficients:

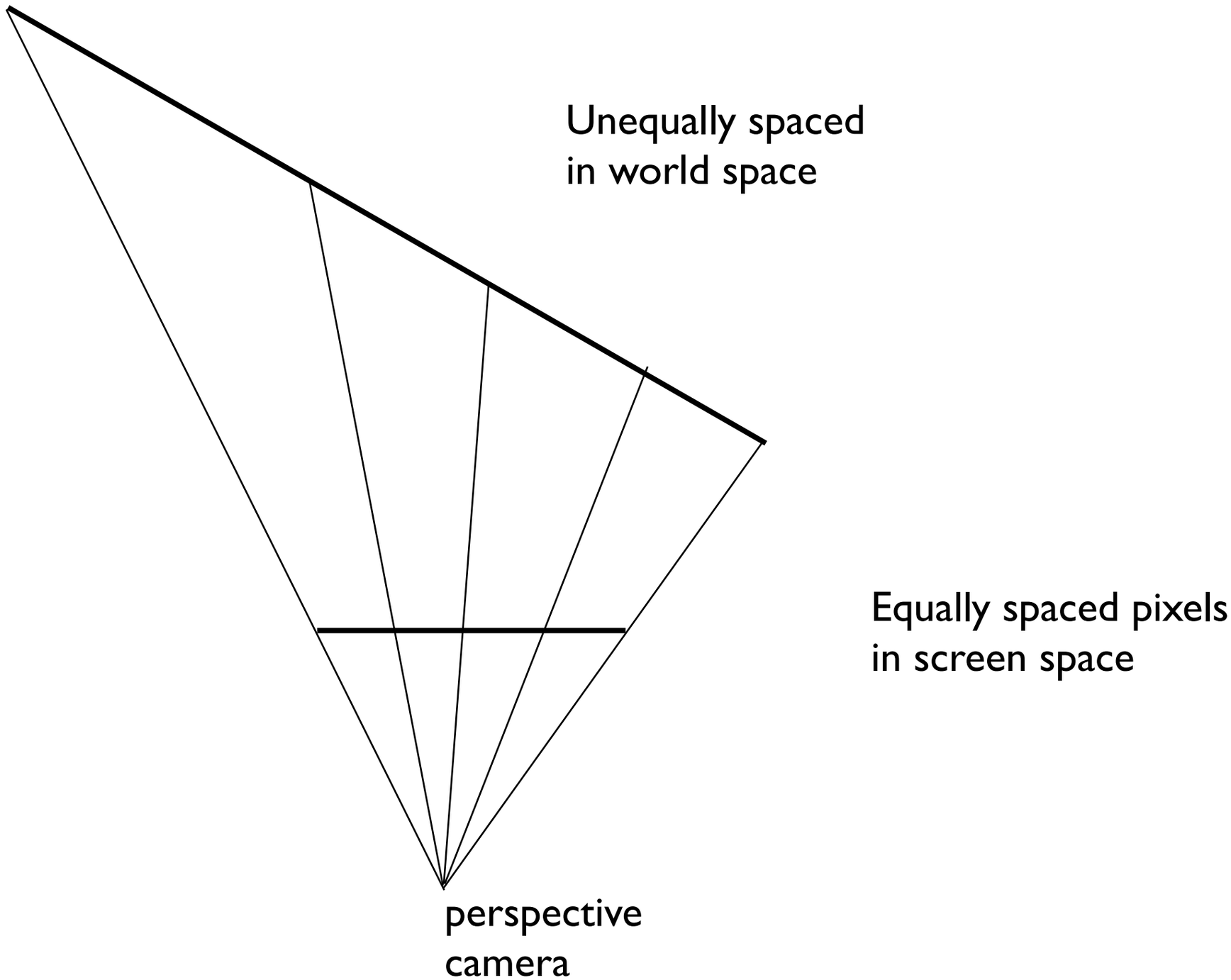
$$I(P) = T(s,t)\left[I_a\rho_a + I_d\rho_d(\hat{\mathbf{s}}\cdot\hat{\mathbf{m}})\right] + I_s\rho_s(\hat{\mathbf{r}}\cdot\hat{\mathbf{v}})^f$$

- We usually leave the specular term unaffected because it is unusual for the material colour to affect specular reflections.

Texture mapping

- When we rasterise an image, we colour each pixel in a polygon by interpolating the texture coordinates of its vertices.
- Standard bilinear interpolation does not work because it fails to take into account foreshortening effects in tilted polygons.

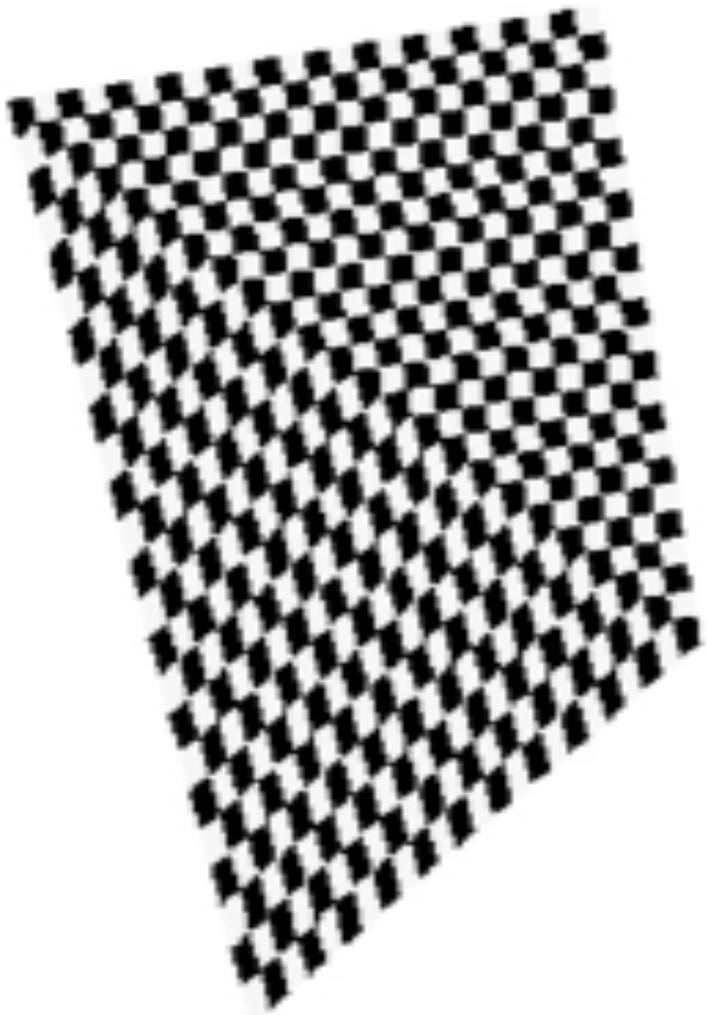
Foreshortening



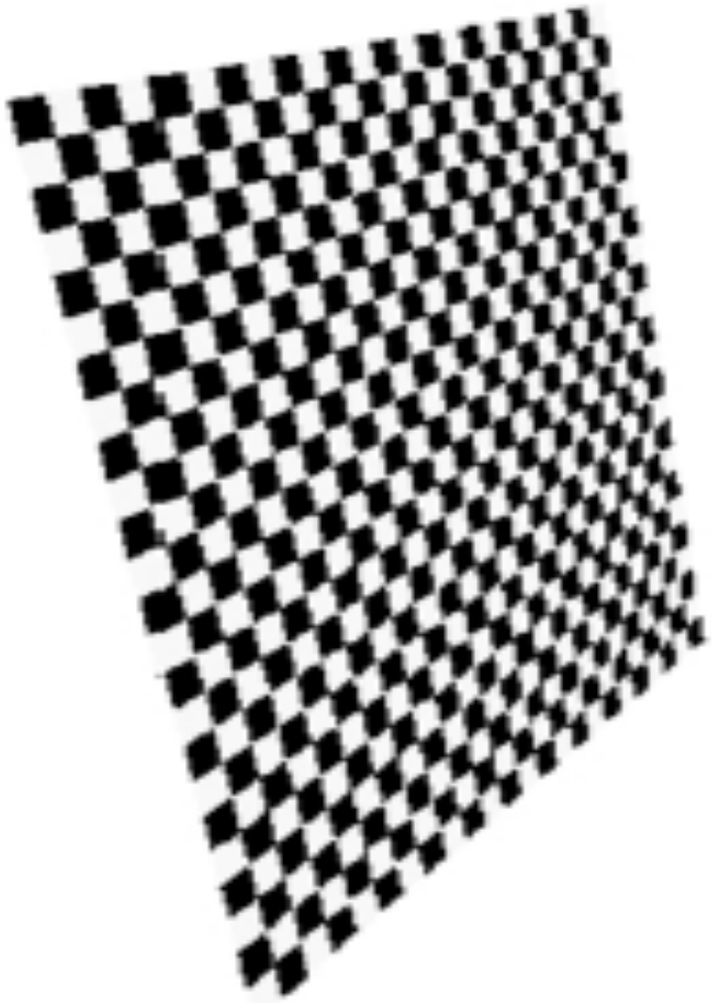
Rendering the Texture

- Linear vs. correct interpolation example:

a) linear interpolation



b) correct interpolation



Hyperbolic interpolation

- We want texture coordinates to interpolate linearly in world space.

- But the perspective projection distorts the depth coordinate so that

  linear interpolation in screen space ≠ linear interpolation in world space

- Hyperbolic interpolation fixes this (NOT PART OF THIS COURSE)

http://web.cs.ucdavis.edu/~amenta/s12/perspectiveCorrect.pdf

3D textures

- We can also make 3D textures by adding an extra texture coordinate.

- Imagine a volume of space with different colours at each point, e.g. a block of wood.

- This eliminates weird seams and distortions when a 2D texture is wrapped on a curve 3D surface.

Magnification

- Normal bitmap textures have finite detail.

- If we zoom in close we can see individual texture pixels (texels).

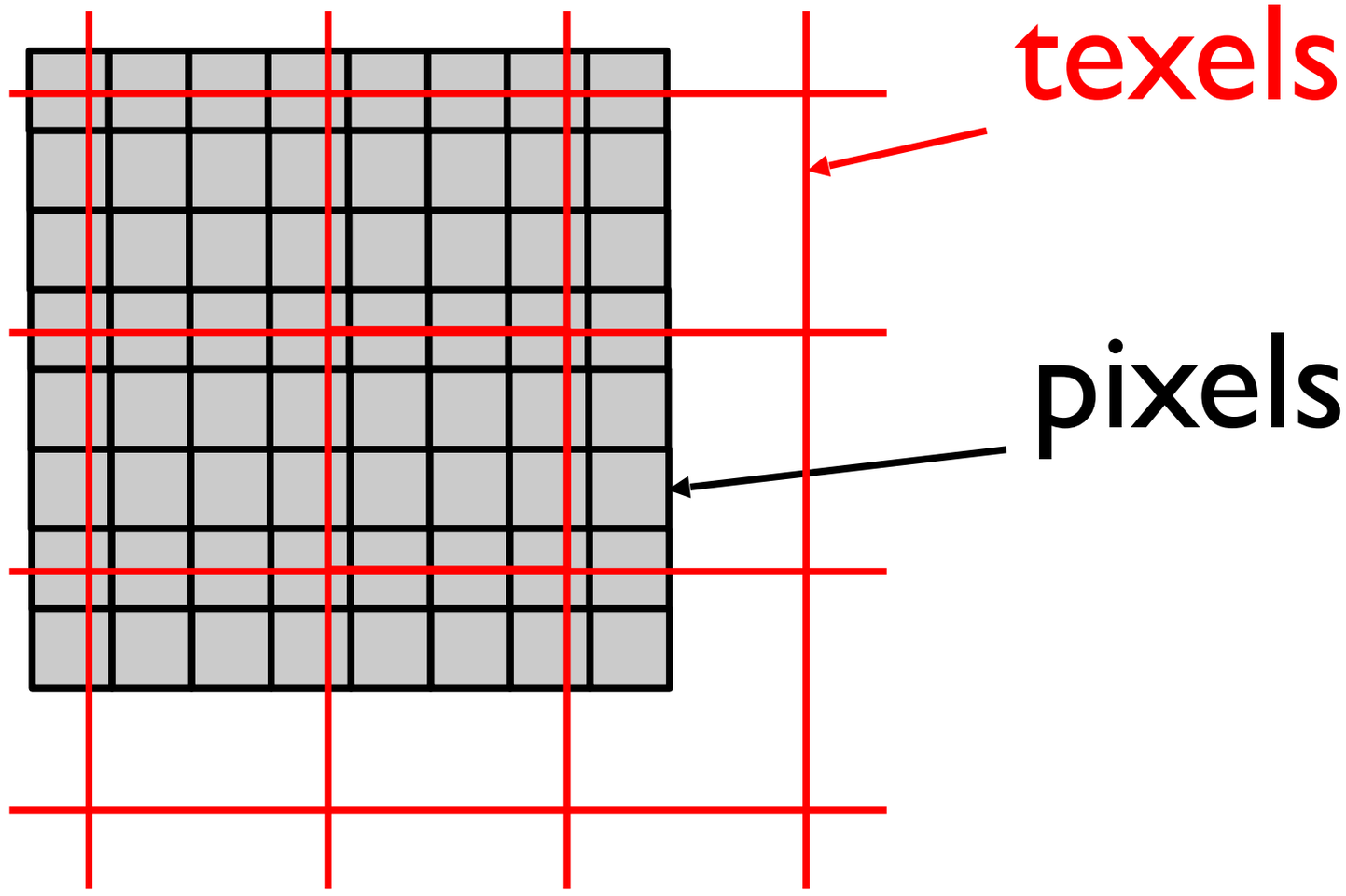
- If the camera is close enough to a textured polygon multiple screen pixels may map to the same texel.

- This results in "pixelated" effects.

Magnification

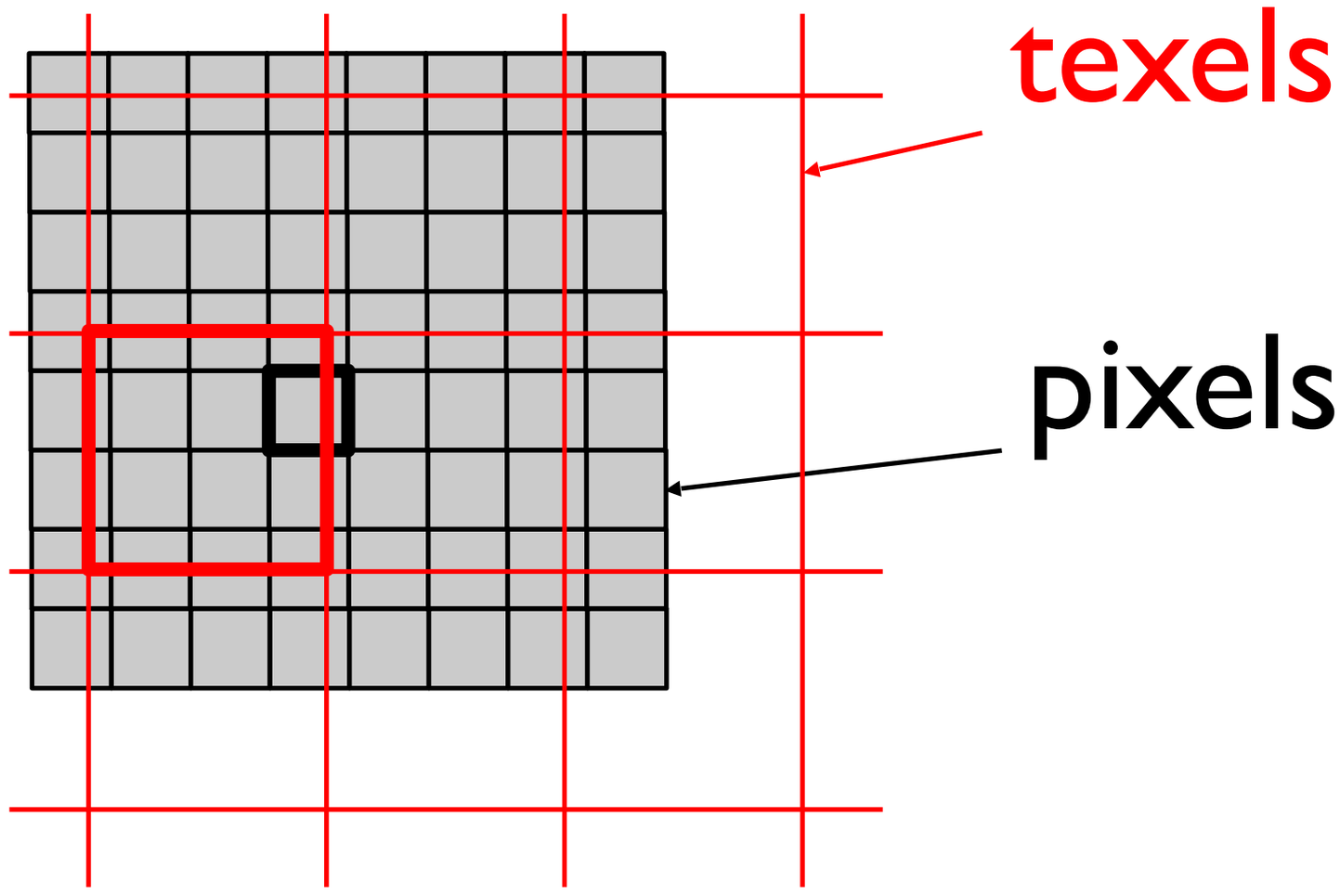
Magnification

- The alignment is probably not exact.



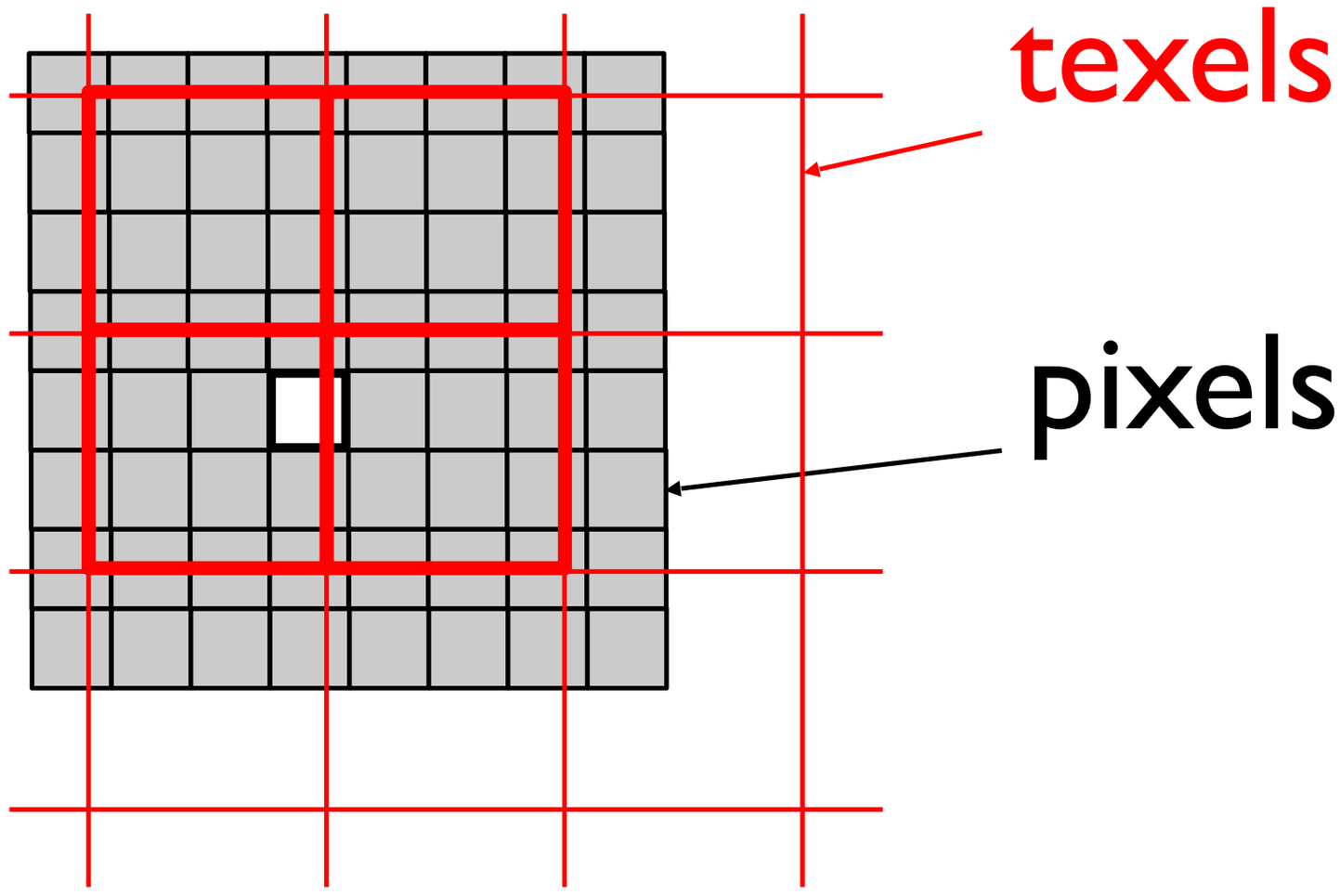
Nearest Texel

- Find the nearest texel.



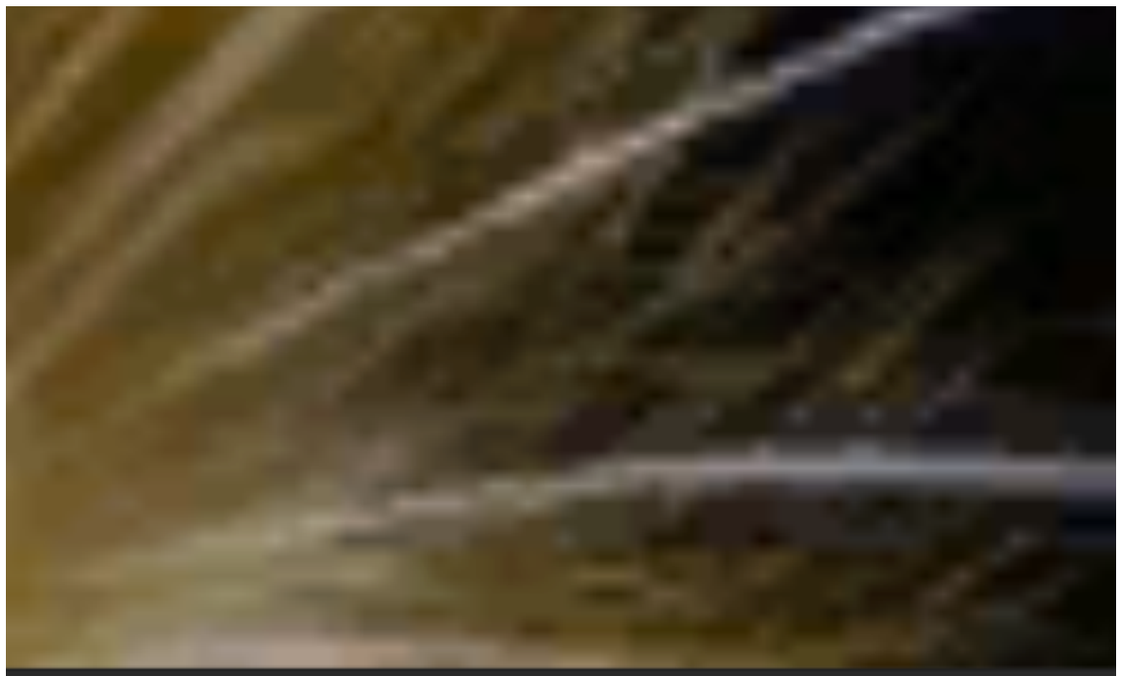
Bilinear Filtering

- Find the nearest four texels and use bilinear interpolation over them



Bilinear Filtering

No filtering



Filtering

Magnification Filtering

```
//bilinear filtering

gl.glTexParameteri(GL.GL_TEXTURE_2D,
GL.GL_TEXTURE_MAG_FILTER,
GL.GL_LINEAR);

// no bilinear filtering

gl.glTexParameteri(GL.GL_TEXTURE_2D,
GL.GL_TEXTURE_MAG_FILTER,
GL.GL_NEAREST);
```

Minification

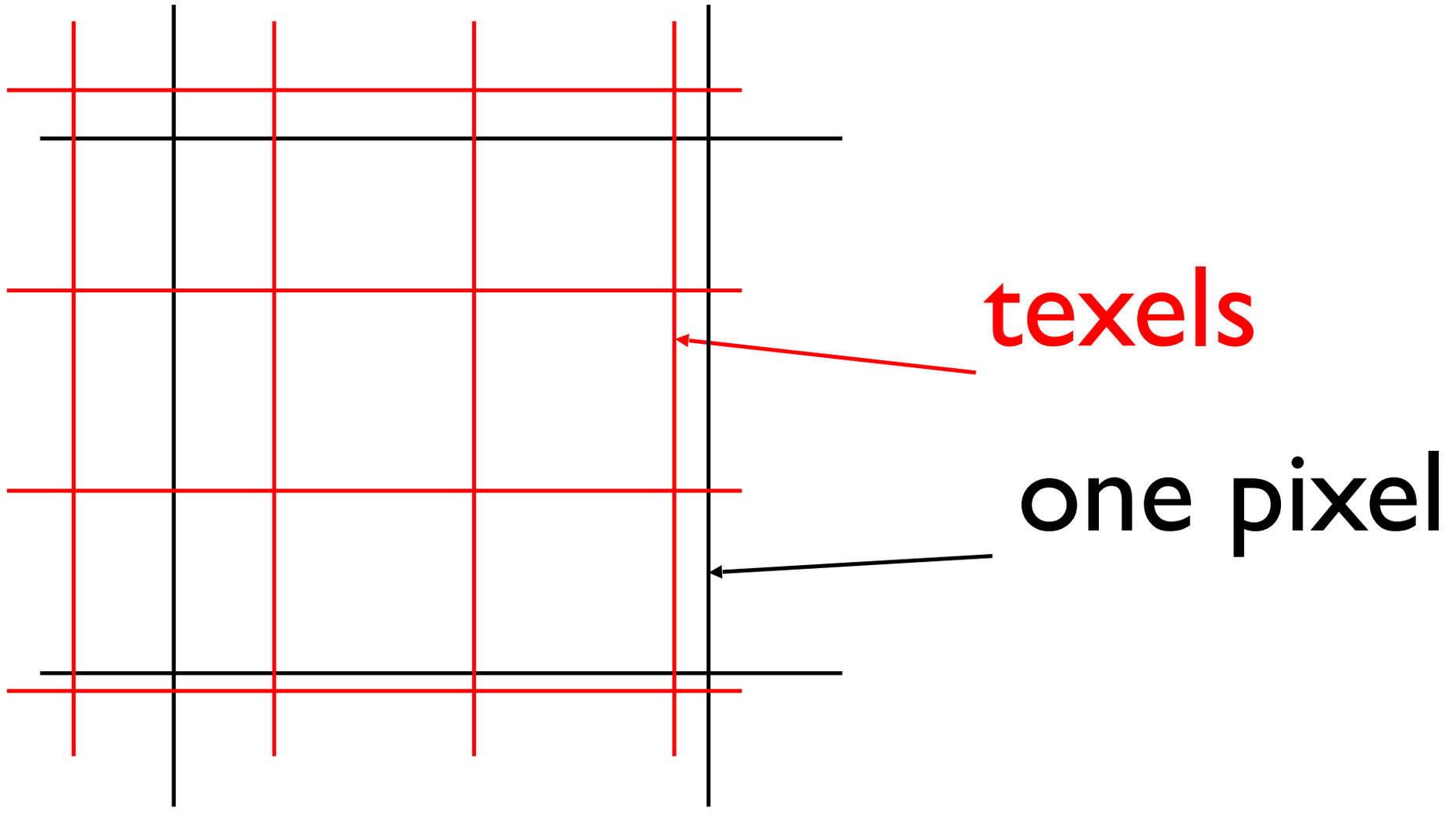
- Problems also occur when we zoom out too far from a texture.

- We can have more than one texel mapping to a pixel.

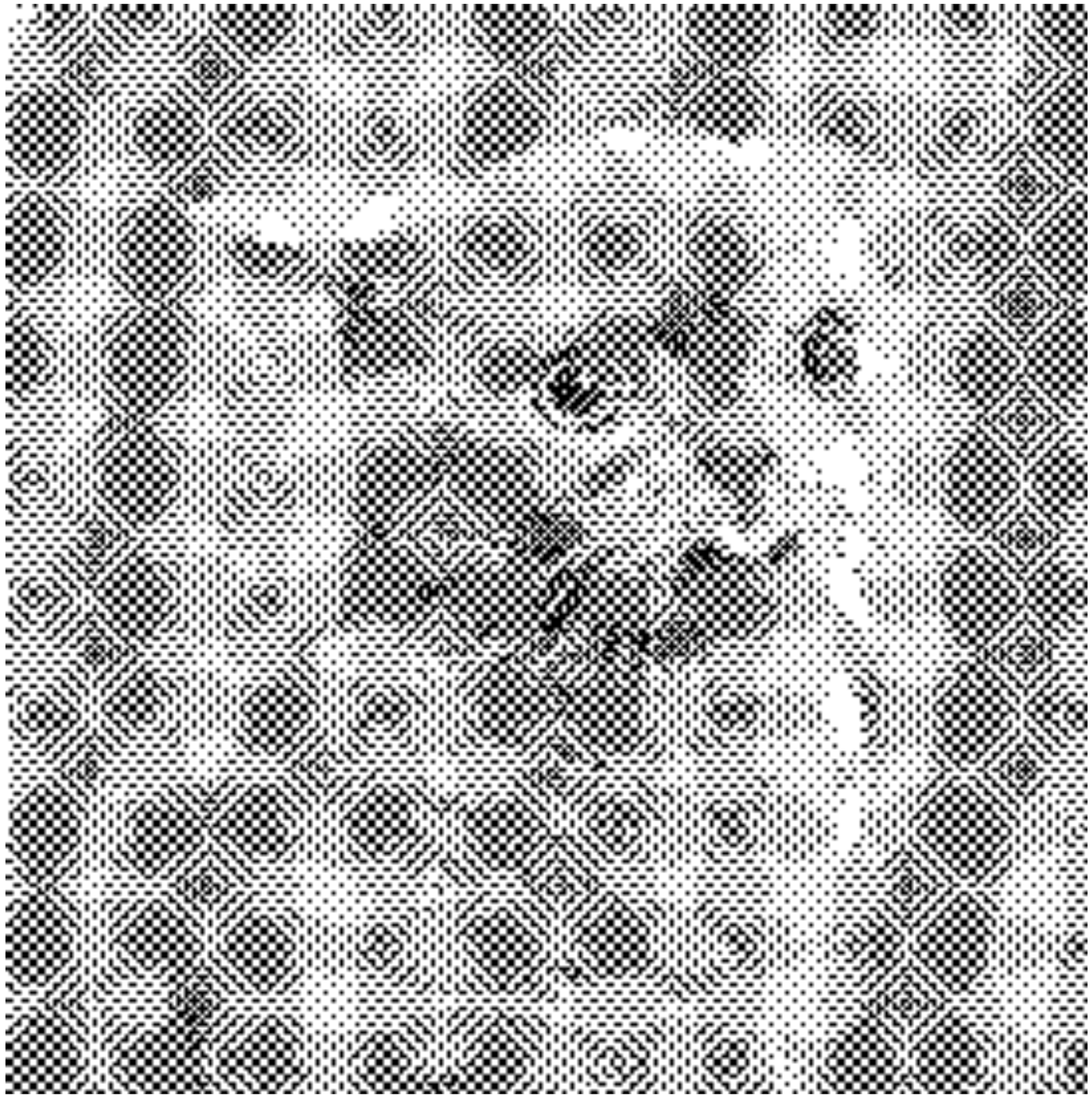
- If image pixels line up with regularities in the texture, strange artefacts appear in the output such as moire patterns or shimmering in an animation

Minification

- Again, the alignment is not exact.



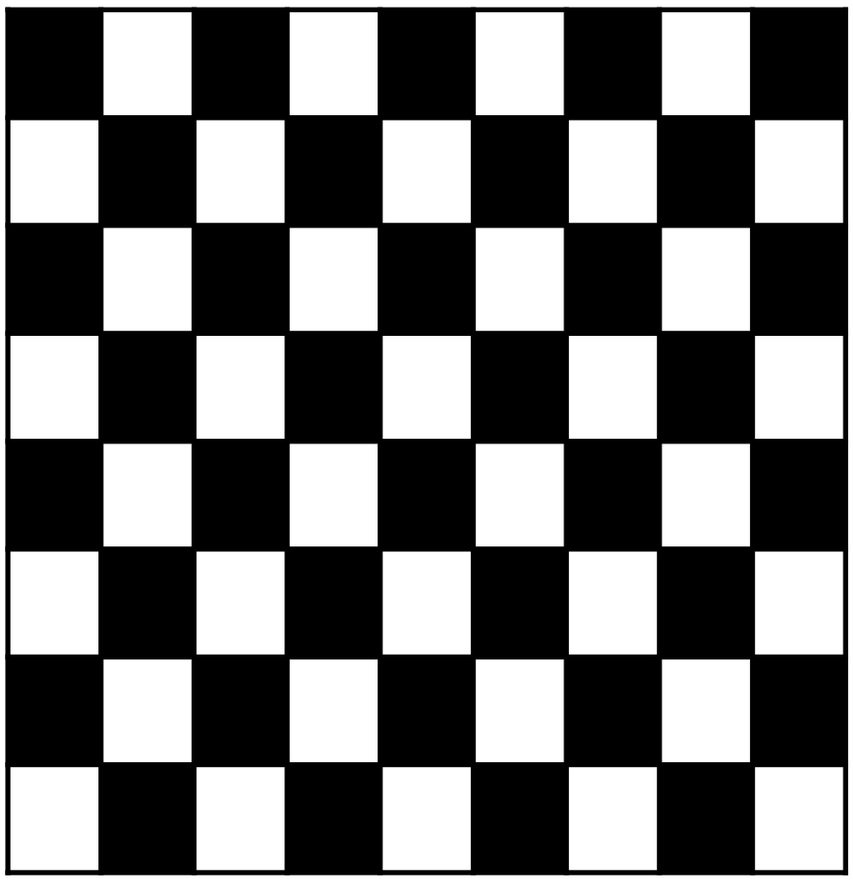
Minification



Aliasing

- This effect is called aliasing. It occurs when samples are taken from an image at a lower resolution than repeating detail in the image.

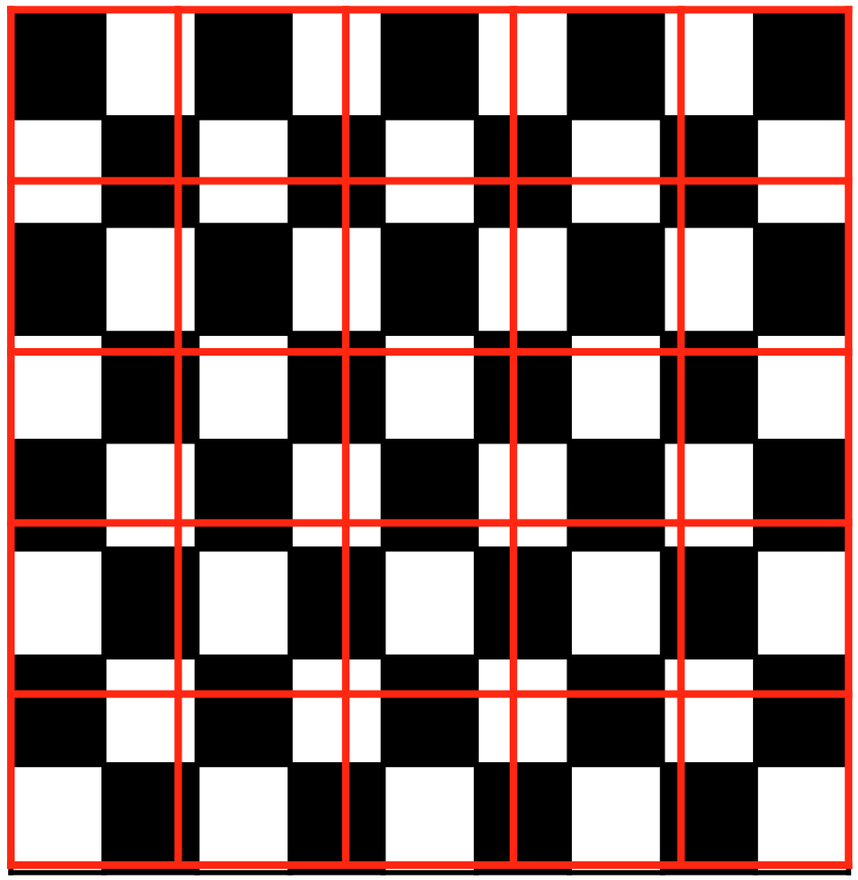
texels



Aliasing

- This effect is called aliasing. It occurs when samples are taken from an image at a lower resolution than repeating detail in the image.

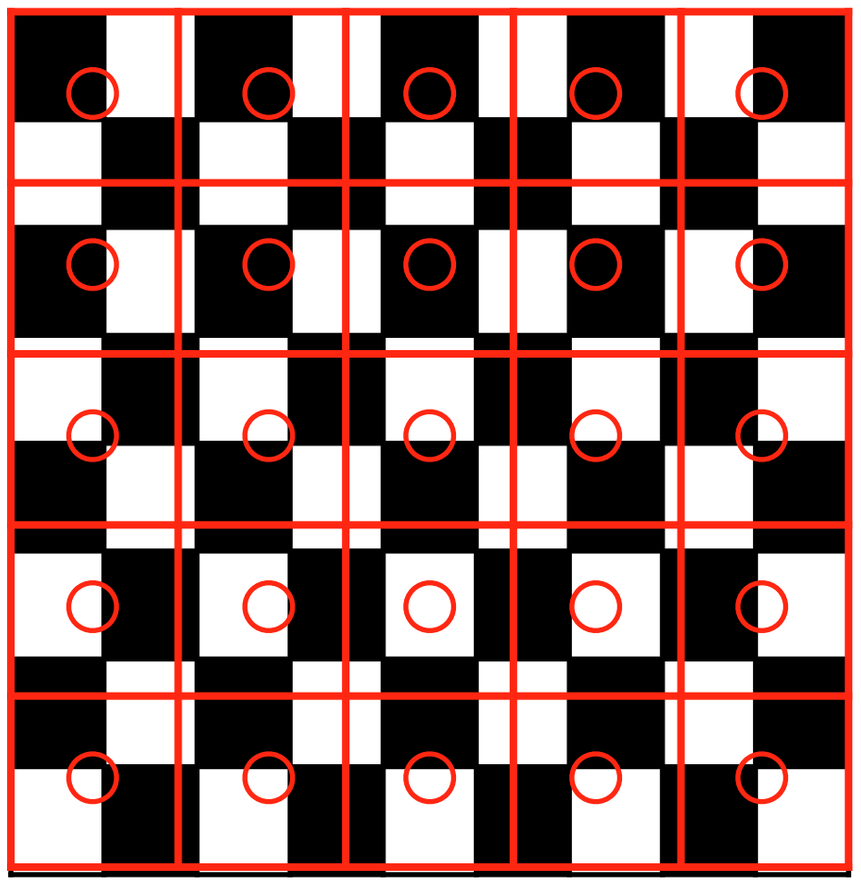
pixels



Aliasing

- This effect is called aliasing. It occurs when samples are taken from an image at a lower resolution than repeating detail in the image.

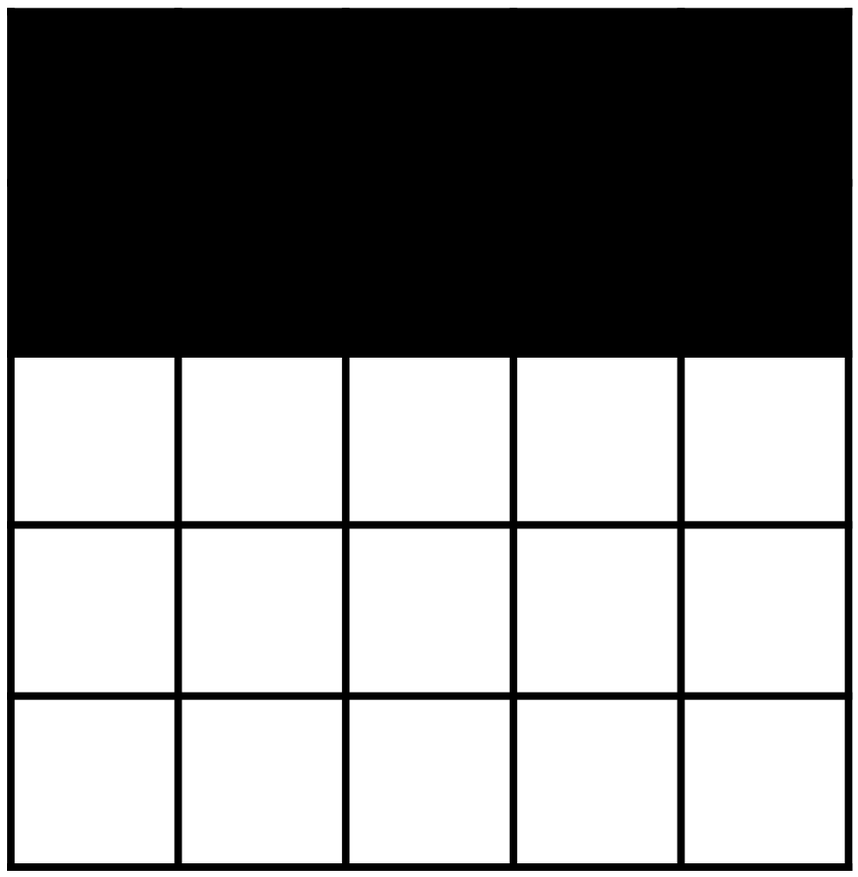
samples



Aliasing

- This effect is called aliasing. It occurs when samples are taken from an image at a lower resolution than repeating detail in the image.

result



Filtering

- The problem is that one screen pixel overlaps multiple texels but is taking its value from only one of those texels.

- A better approach is to average the texels that contribute to that pixel.

- Doing this on the fly is expensive.

Minification Filtering

```
//bilinear filtering

gl.glTexParameteri(GL.GL_TEXTURE_2D,
GL.GL_TEXTURE_MIN_FILTER,  GL.GL_LINEAR);

// no bilinear filtering

gl.glTexParameteri(  GL.GL_TEXTURE_2D,
GL.GL_TEXTURE_MIN_FILTER,  GL.GL_NEAREST);
```

MIP mapping

- Mipmaps are precomputed low-res versions of a texture.

- Starting with a 512x512 texture we compute and store 256x256, 128x128, 64x64, 32x32, 16x16, 8x8, 4x4, 2x2 and 1x1 versions.

- This takes total memory = 4/3 original.

Generating Mip-Maps

```
//get opengl to auto-generate
//mip-maps.
gl.glGenerateMipmap(GL.GL_TEXTURE_2D);
```

Using mipmaps

- The simplest approach is to use the next smallest mipmap for the required resolution.
  - E.g. To render a 40x40 pixel image, use the 32x32 pixel mipmap and magnify using magnification filter

MipMap Minification Filtering

```
// use nearest mipmap

gl.glTexParameteri(  GL.GL_TEXTURE_2D,
GL.GL_TEXTURE_MIN_FILTER,
    GL.GL_NEAREST_MIPMAP_NEAREST);
```

Trilinear filtering

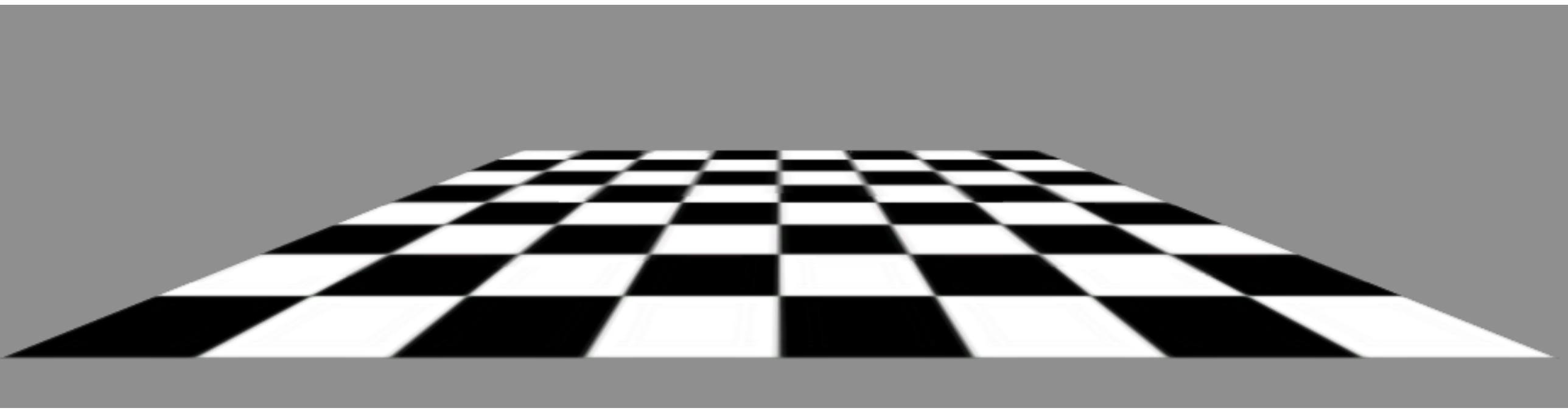
- A more costly approach is trilinear filtering:
  - Use bilinear filtering to compute pixel values based on the next highest and the next lowest mipmap resolutions.
  - Interpolate between these values depending on the desired resolution.

MipMap Minification Filtering

```
// use trilinear filtering

gl.glTexParameteri(
GL.GL_TEXTURE_2D,
GL.GL_TEXTURE_MIN_FILTER,
GL.GL_LINEAR_MIPMAP_LINEAR);
```

Aniso Filtering

- If a polygon is on an oblique angle away from the camera, then minification may occur much more strongly in one dimension than the other.

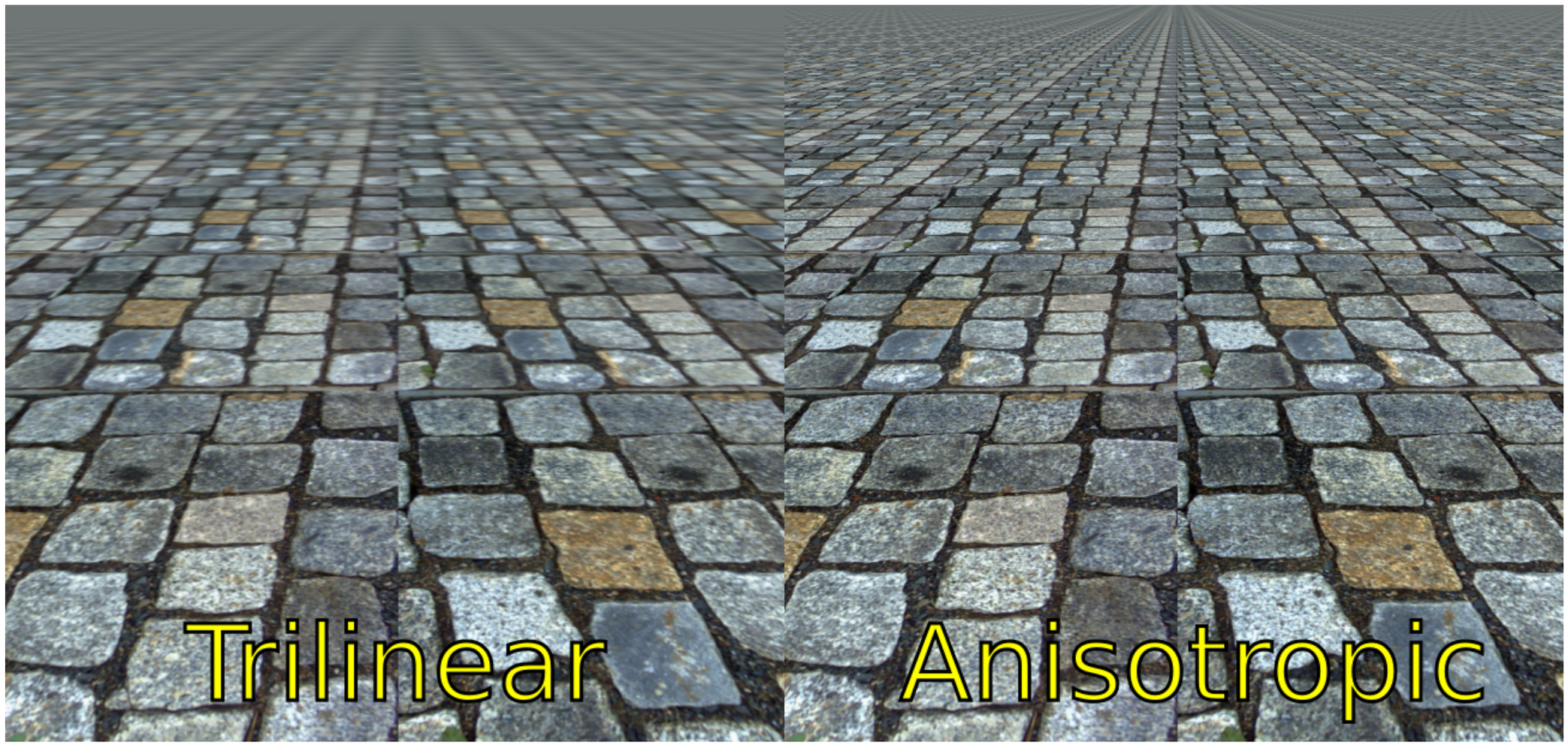


Aniso filtering

Anisotropic filtering is filtering which treats the two axes independently.

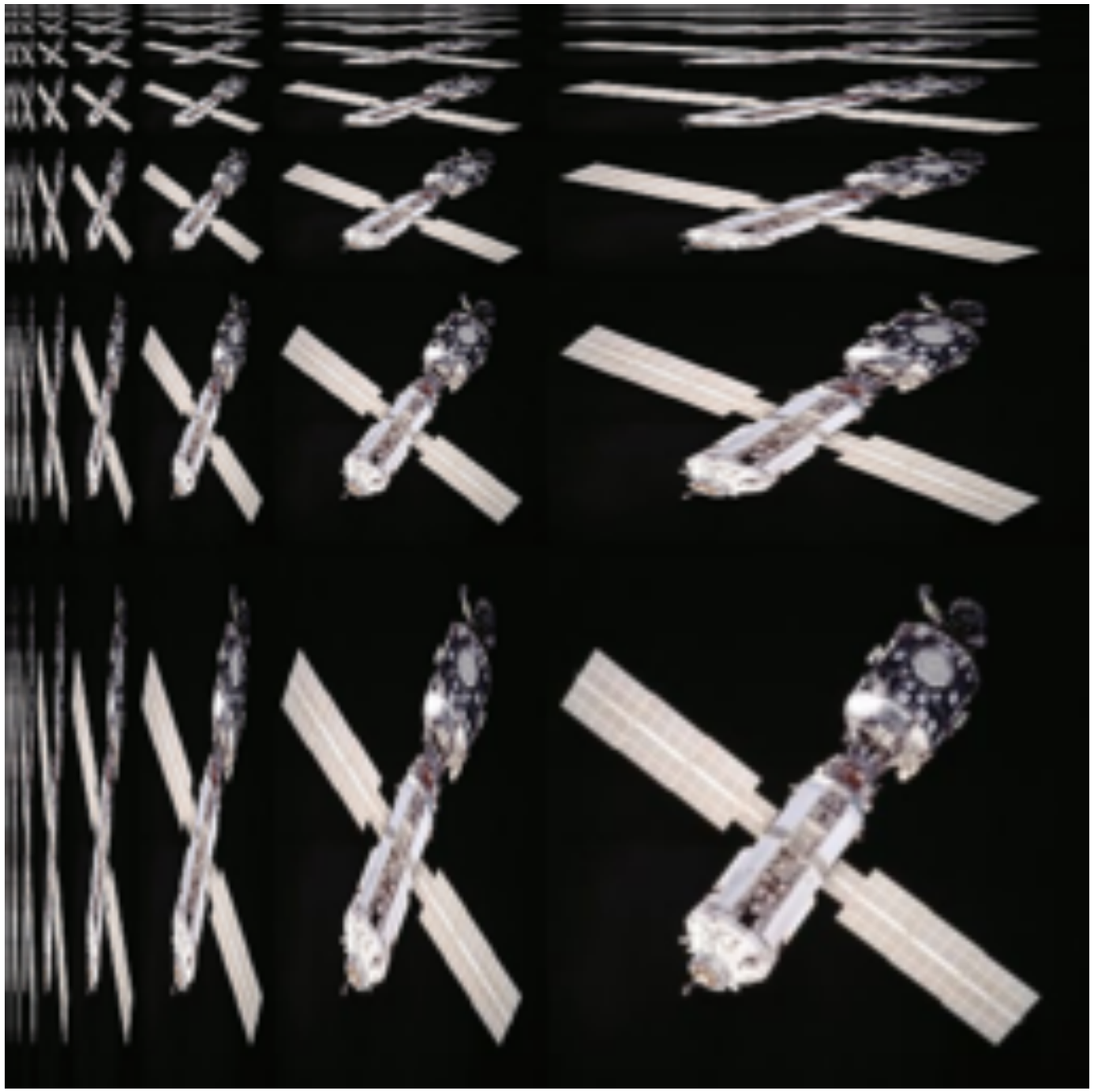
```
float fLargest[] = new float[1];
gl.glGetFloatv(GL.GL_MAX_TEXTURE_MAX_ANISOTROP
Y_EXT, fLargest,0);

gl.glTexParameterf(GL.GL_TEXTURE_2D,
GL.GL_TEXTURE_MAX_ANISOTROPY_EXT,
fLargest[0]);
```

Aniso Filtering



RIP Mapping

- RIP mapping is an extension of MIP mapping which down-samples each axis and is a better approach to anisotropic filtering
  - So a 256x256 image has copies at:
    256x128, 256x64, 256x32, 256x16, ...,
    128x256, 128x128, 128x64, ...
    64x256, 64x128, etc.



RIP Mapping

- Limitations of RIP Mapping:
  - Does not handle diagonal anisotropy.
  - More memory required for RIP maps (4 times as much).
  - Not implemented in OpenGL

Multi-texturing

Can use more than one texture on the same fragment.

```
gl.glActiveTexture(GL.GL_TEXTURE0);
gl.glBindTexture(GL.GL_TEXTURE_2D, texId1);
```

```
gl.glActiveTexture(GL.GL_TEXTURE1);
gl.glBindTexture(GL.GL_TEXTURE_2D, texId2);
```

Multi-texturing

- Have to pass two different textures to the shader.
- … and two different sets of texture coordinates

Animated textures

Animated textures can be achieved by loading multiple textures and using a different one on each frame.