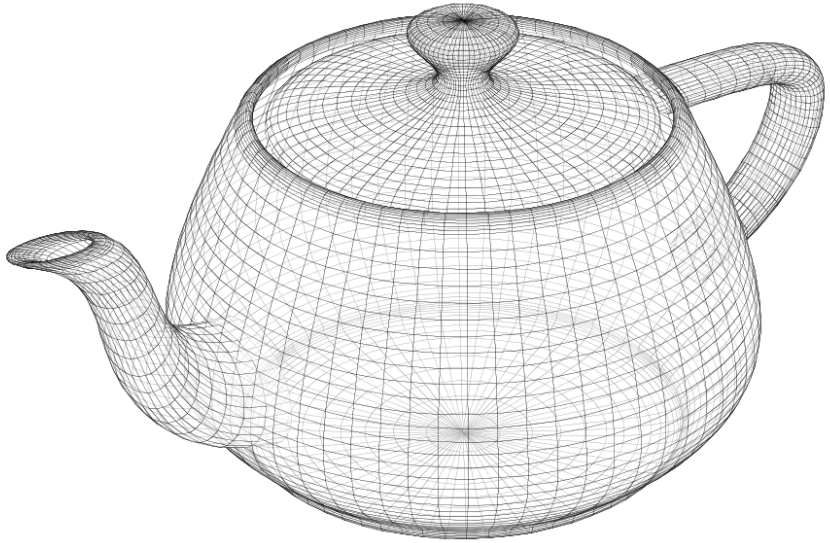


Texture Filtering

Minification

Interactive Computer Graphics

Professor Eric Shaffer

ILLINOIS

Filtering Textures

- Minification occurs when we have more texels than fragments

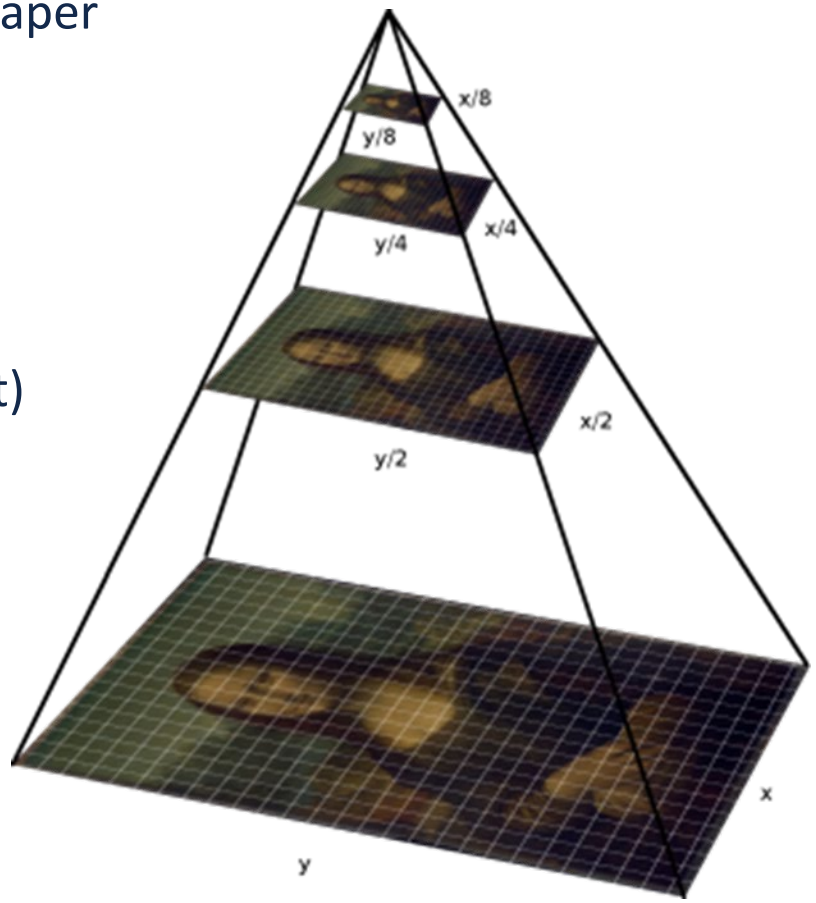
- Using NN or Bilinear Filtering can lead to aliasing
- Why?
  - Sparse sampling will can cause us to miss features
  - e.g. a checkerboard pattern could be turned into solid color

- What would a better strategy be?
  - Average all of the texels that map into a fragment

- What is the maximum number of texels fetched per fragment?
  - The entire texture

Mipmapping

- Mipmapping is a method of pre-filtering a texture for minification
  - History: 1983 Lance Williams introduced the word “mipmap” in his paper “Pyramidal Parametrics”
  - mip = “multum in parvo”…. latin: many things in small place(?)
- We generate a pyramid of textures
  - Bottom-level is the original texture
  - Each subsequent level reduces the resolution by ¼ (by ½ along s and t)



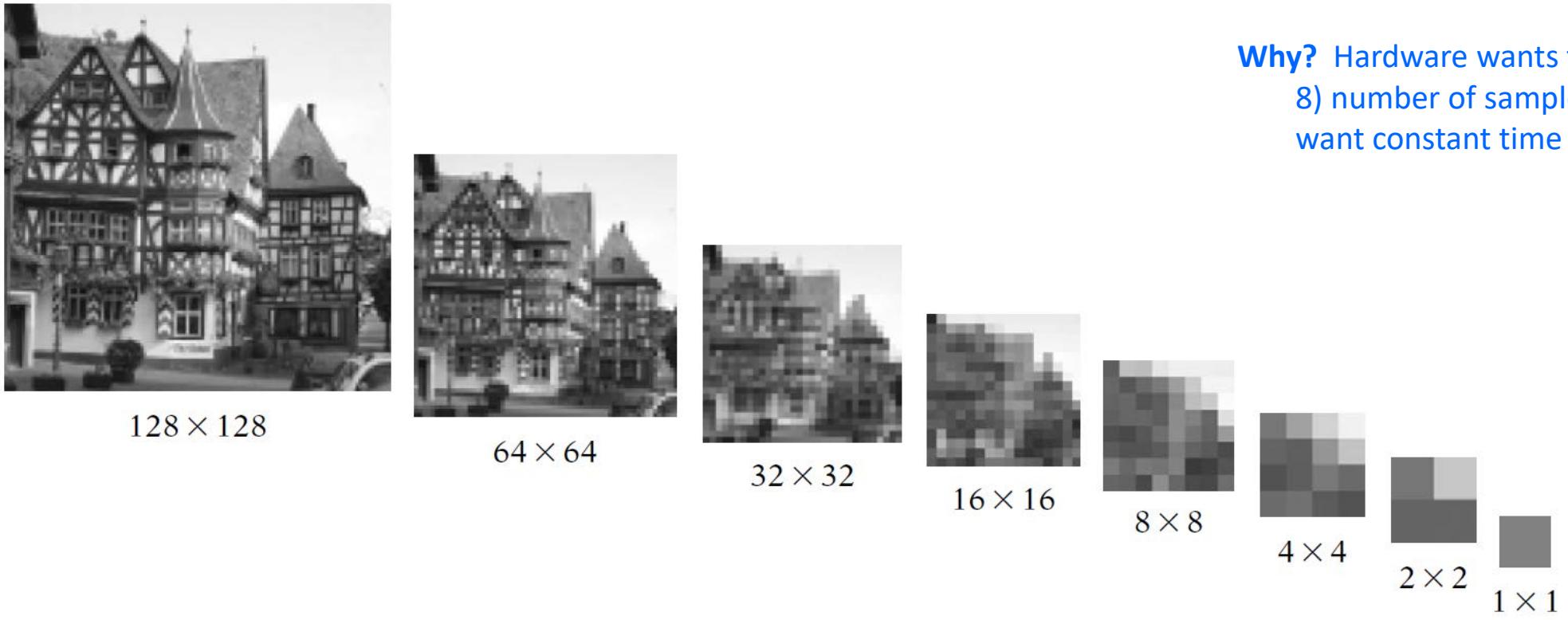
ILLINOIS

Pre-filtered Image Versions

- Base texture image is say 128x128
  - Then down-sample 64x64, 32x32, all the way down to 1x1

Trick: When sampling the texture, pick the mipmap level with the closest mapping of pixel to texel size

Why? Hardware wants to sample just a small (1 to 8) number of samples for every fetch—and want constant time access



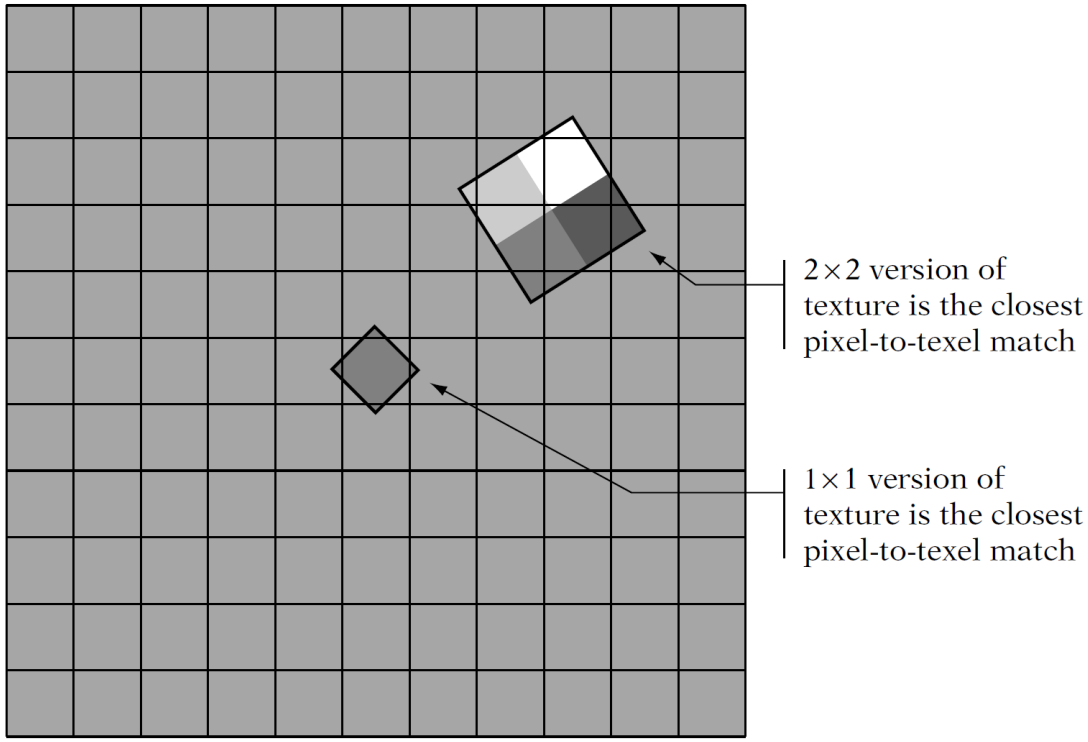
Creating a Mipmap

- In WebGL you can manually generate and upload a mipmap
- Or you can have WebGL generate it for you

```
gl.generateMipmap(GL_TEXTURE_2d)
```

- Usually, bilinear filtering is used to minify each level
- …but that's up to the implementation of the library

Level of Detail Selection



Screen-space geometry
(same mipmapped texture applied to both squares)

ILLINOIS

Mipmap Level-of-Detail Selection

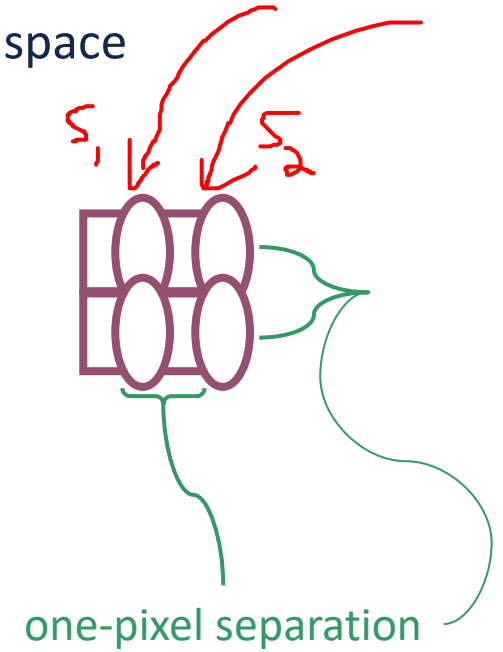
- Hardware uses 2x2 fragment entities
  - Typically called quad-fragments or quad-pixels or just *quad*
  - Finite difference with neighbors to get change in s and t with respect to window space
    - Approximation to ∂s/∂x, ∂s/∂y, ∂t/∂x, ∂t/∂y
    - Means 4 subtractions per quad (1 per pixel)

$$\frac{\partial s}{\partial x} = change\ in\ s\ coordinate\ over\ one\ unit\ change\ in\ horizontal\ direction$$

$$\frac{\partial s}{\partial y} = change\ in\ s\ coordinate\ over\ one\ unit\ change\ in\ vertical\ direction$$

$$\frac{\partial t}{\partial x} = change\ in\ t\ coordinate\ over\ one\ unit\ change\ in\ horizontal\ direction$$

$$\frac{\partial t}{\partial y} = change\ in\ t\ coordinate\ over\ one\ unit\ change\ in\ vertical\ direction$$

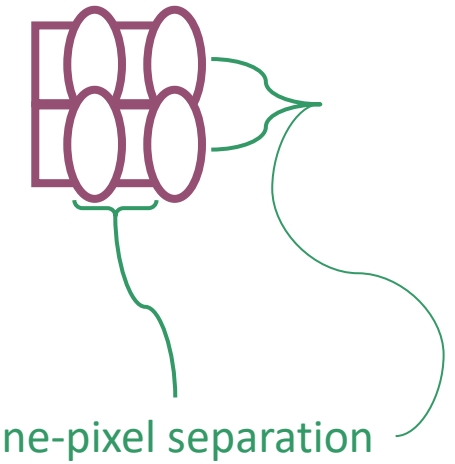


ILLINOIS

Mipmap Level-of-Detail Selection

- Hardware uses 2x2 pixel entities
  - Typically called quad-pixels or just *quad*
  - Finite difference with neighbors to get change in u and v with respect to window space
    - Approximation to $\partial s/\partial x$, $\partial s/\partial y$, $\partial t/\partial x$, $\partial t/\partial y$
    - Means 4 subtractions per quad (1 per pixel)

- Now compute approximation to gradient length
  - $p = \max(\sqrt{((\partial s/\partial x)^2+(\partial t/\partial x)^2)}, \sqrt{((\partial s/\partial y)^2+(\partial t/\partial y)^2)})$



Level-of-Detail Bias and Clamping

- Convert p length to level-of-detail
  - $\lambda = \log_2(p)$

- Now clamp λ to valid LOD range
  - $\lambda' = \max(\text{minLOD}, \min(\text{maxLOD}, \lambda))$

Determine Mipmap Levels

- Determine lower and upper mipmap levels
  - b = floor(λ’)) is bottom mipmap level
  - t = floor(λ’+1) is top mipmap level

- Determine filter weight between levels
  - w = frac(λ’) is filter weight

WebGL Computing a Color from a Mipmap

WebGL offers 6 ways to generate a color from a mipmap

NEAREST = choose 1 texel from the biggest mip

LINEAR = choose 4 texels from the biggest mip and blend them

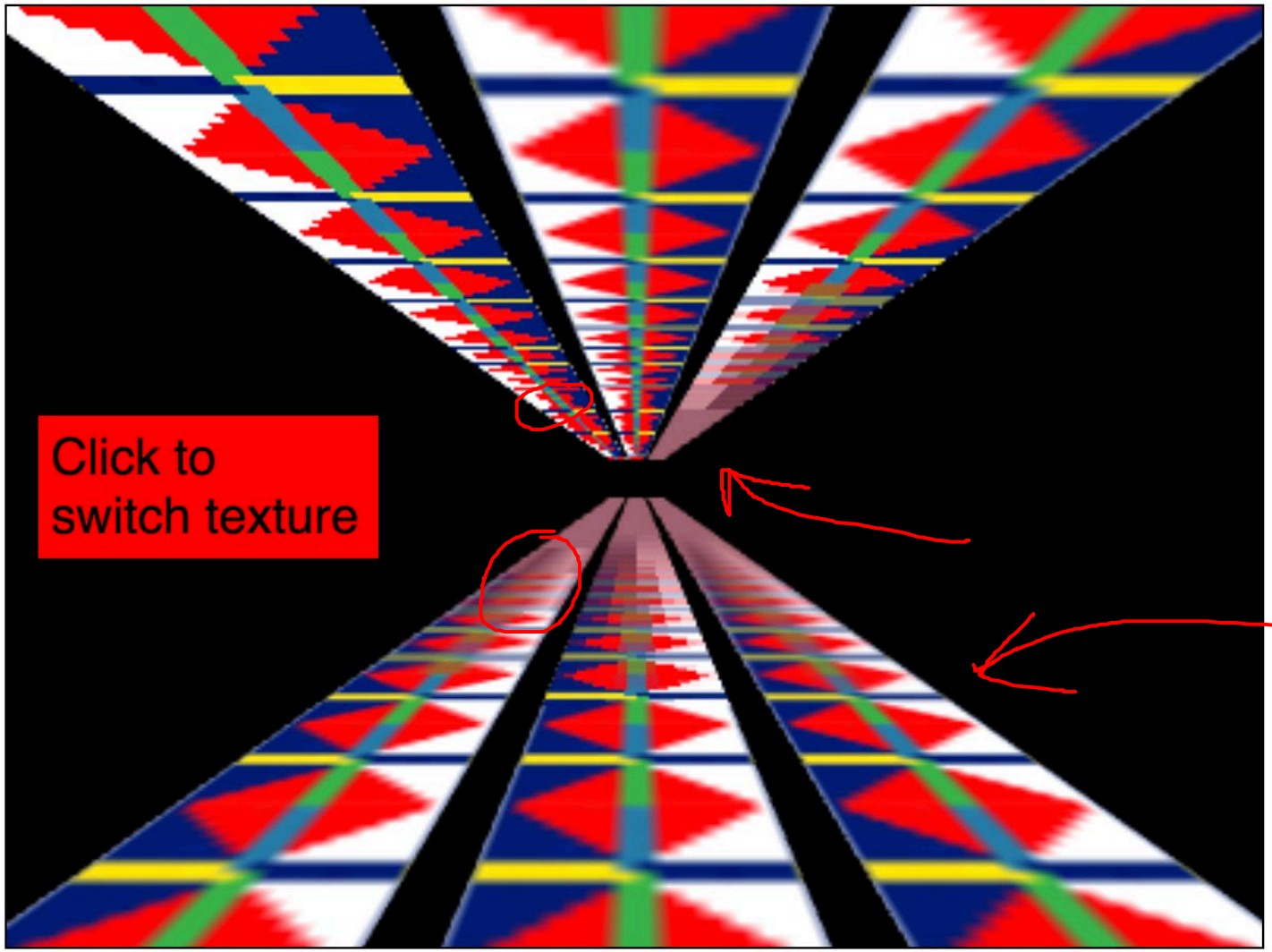
NEAREST_MIPMAP_NEAREST = choose the best mip,
then pick one texel from that mip

LINEAR_MIPMAP_NEAREST = choose the best mip,
then blend 4 texels from that mip

NEAREST_MIPMAP_LINEAR = choose the best 2 mips,
choose 1 texel from each, blend them

LINEAR_MIPMAP_LINEAR = choose the best 2 mips.
choose 4 texels from each, blend them

Mipmap Texture Filtering



WebGL: Highest Quality Filtering

```
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_LINEAR);

gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
```

Although some WebGL implementations may now support anisotropic texture filtering…which is even better

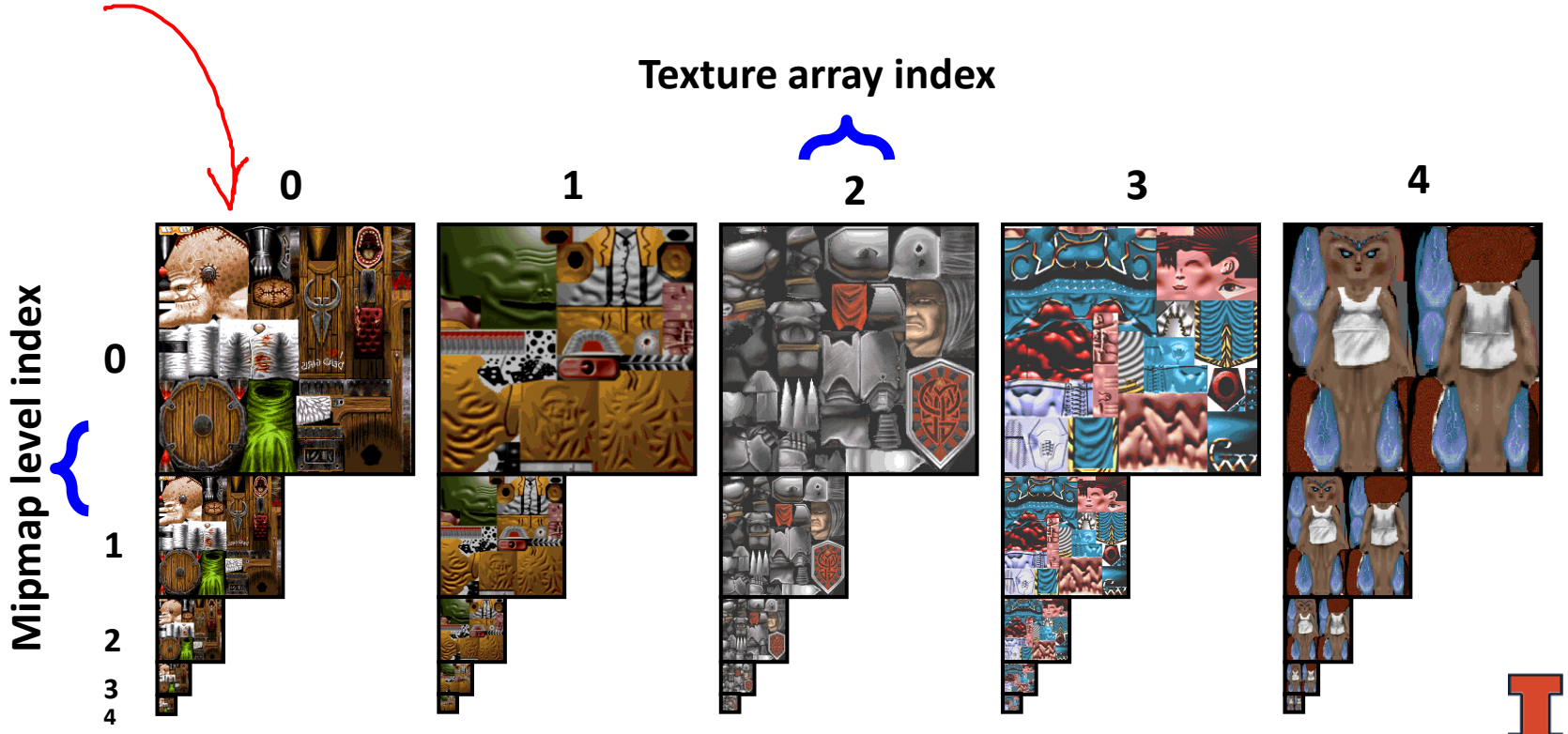
WebGL: Non-power of 2 textures

- You should use textures that are 2^k x 2^k

- You can use textures that are not powers of two
- but must
  - set the wrap mode to CLAMP_TO_EDGE
  - turn off mipmapping by setting filtering to LINEAR or NEAREST…

Texture Arrays

- **Multiple skins packed in texture array**
  - Motivation: binding to one multi-skin texture array avoids texture bind per object



ILLINOIS