CS 418: Interactive Computer Graphics

Texture Filtering

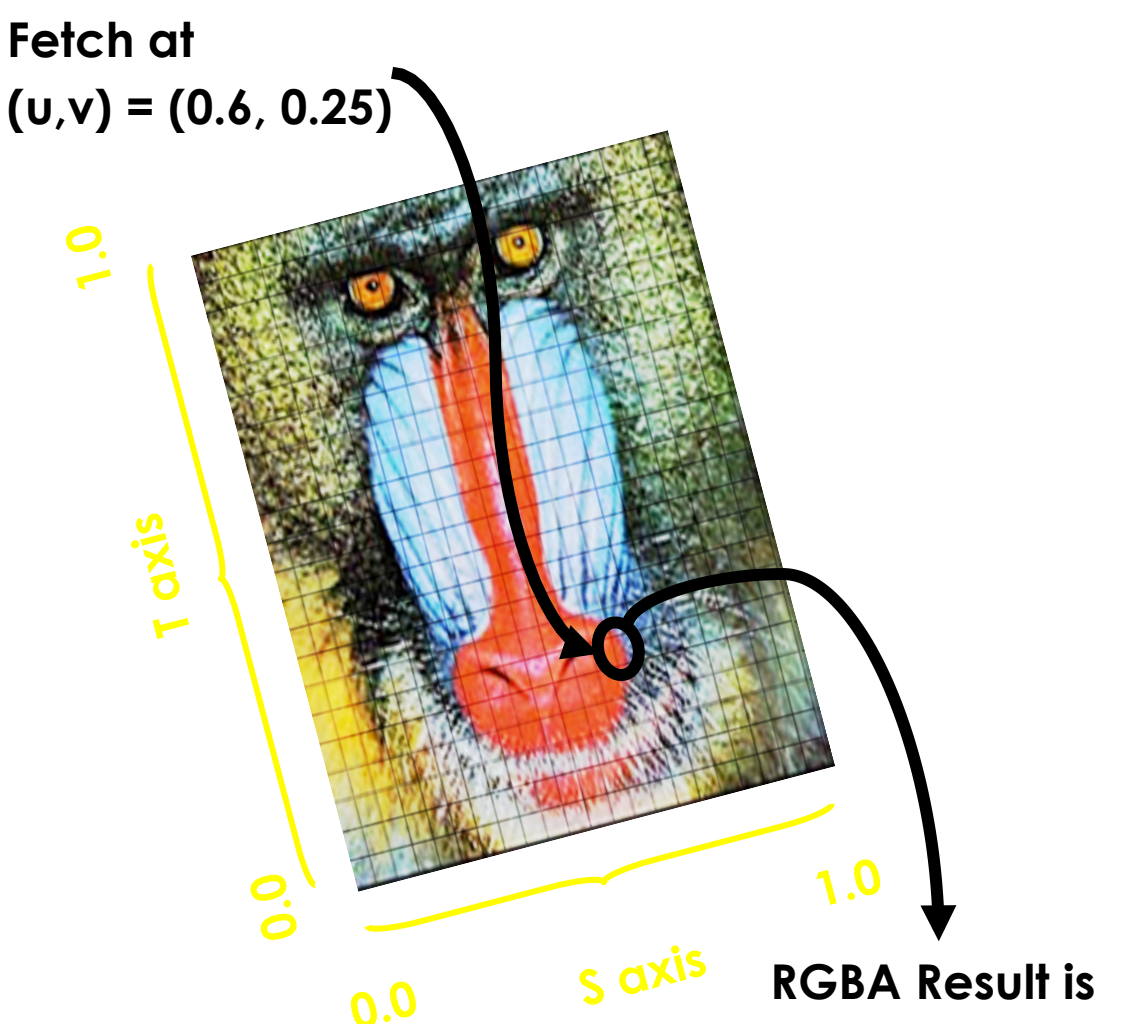
Eric Shaffer

A note about coordinates…

- We're using the following convention:
- (u,v) are the texture coordinates assigned in the parametric space with u and v in [0,1]
- (s,t) are the texel coordinates in a texture
- ….some people use (s,t) to denote the parametric coordinates…

A Texture Fetch Simplified

- *Seems pretty simple…*
- Given
  1. An image
  2. A position
- Return the color of image at position



Filtering Textures

- Magnification occurs when we have more fragments than texels
- What are two filters we can use to map texels to fragments?

- If we are magnifying a texture, what is the maximum number of texels that must be fetched per fragment?

Filtering Textures

- Magnification occurs when we have more fragments than texels
- What are two filters we can use to map texels to fragments?
  - Nearest Neighbor
  - Bilinear Filtering

- If we are magnifying a texture, what is the maximum number of texels that must be fetched per fragment?
  - Four for bilinear filtering.

Filtering Textures

- Minification occurs when we have more texels than fragments
- Using NN or Bilinear Filtering can lead to aliasing
- Why?
- What would a better strategy be?
- What is the maximum number of texels fetched per fragment?

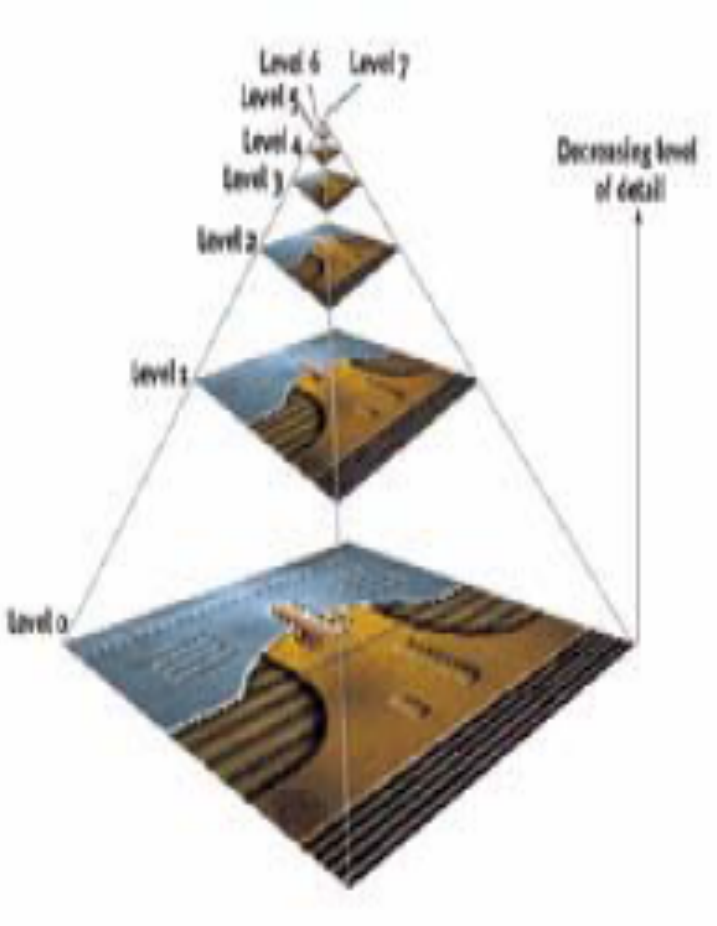
Filtering Textures

- 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 featues
  - 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)

Mipmapping


Level 6
Level 7
Level 5
Level 4
Level 3
Level 2
Level 1
Level 0
Decreasing level of detail

Pre-filtered Image Versions

- Base texture image is say 256x256
  - Then down-sample 128x128, 64x64, 32x32, all the way down to 1x1

Trick: When sampling the texture, pixel 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

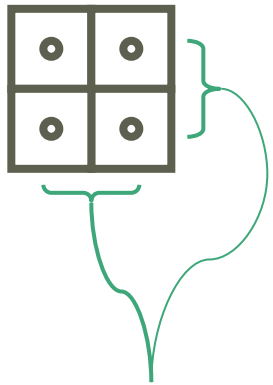
- In WebGL you can manually generate and upload a mipmap
- Or you can have WebGL generate it for you

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

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

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 $\partial u/\partial x$, $\partial u/\partial y$, $\partial v/\partial x$, $\partial v/\partial y$
    - Means 4 subtractions per quad (1 per pixel)
- Now compute approximation to gradient length
  - $p = \max(\text{sqrt}((\partial u/\partial x)^2+(\partial u/\partial y)^2), \text{sqrt}((\partial v/\partial x)^2+(\partial v/\partial y)^2))$



one-pixel separation

Level-of-detail Bias and Clamping

- Convert p length to level-of-detail and apply LOD bias
  - λ = log2(p) + lodBias
- Now clamp λ to valid LOD range
  - λ' = max(minLOD, min(maxLOD, λ))

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 pixel from the biggest mip

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

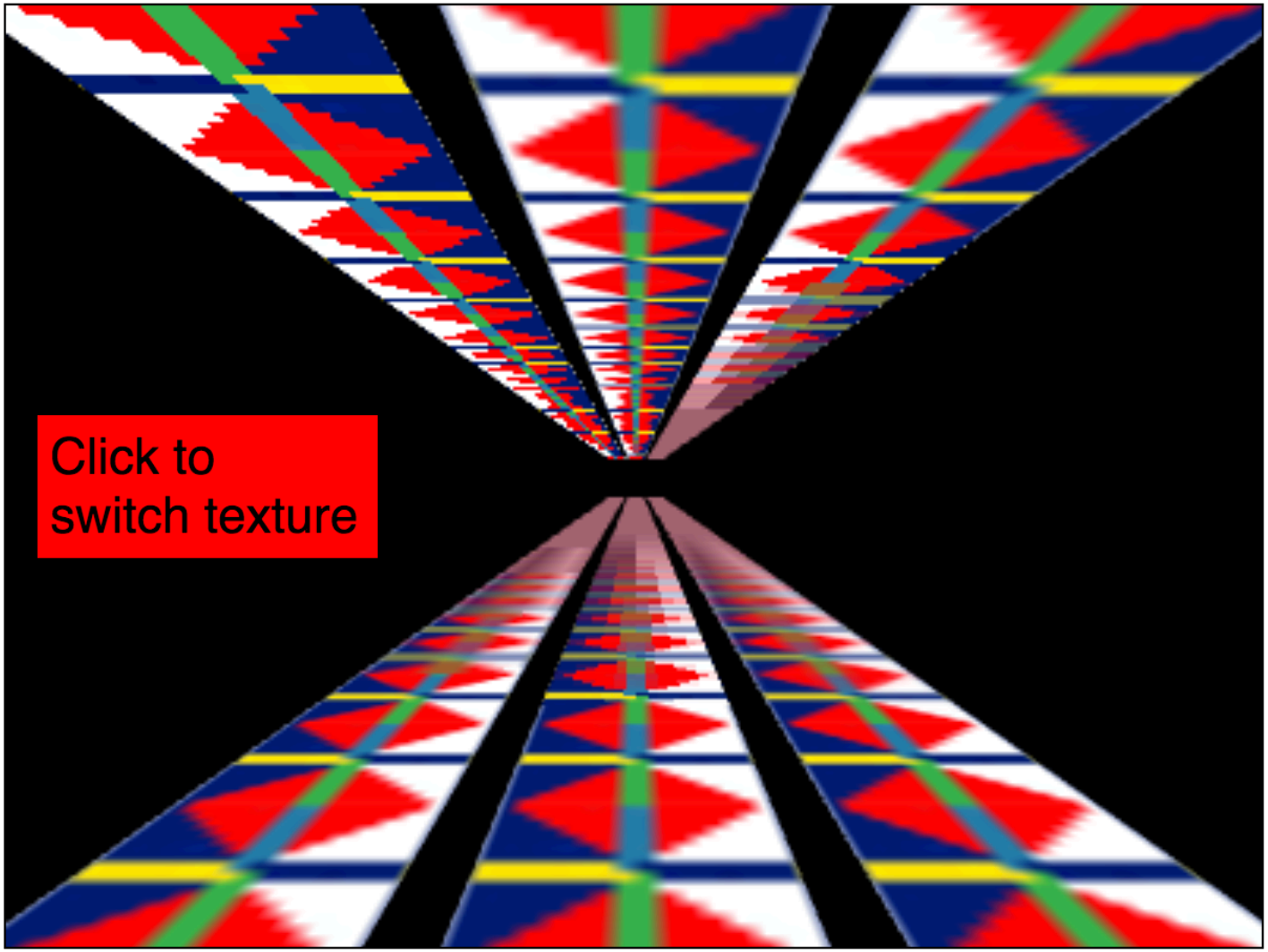
NEAREST_MIPMAP_NEAREST = choose the best mip,
then pick one pixel from that mip

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

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

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

Mipmap Texture Filtering



WebGL: Highest Quality Filtering

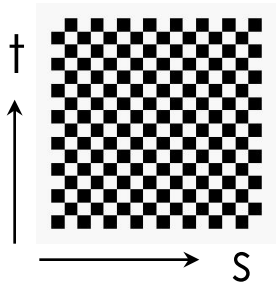
```
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_LINEAR);

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

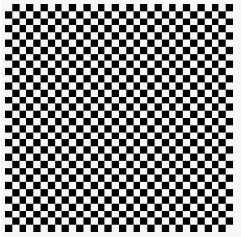
Although some WebGL implementations may now support anisotropic texture filtering…which is even better

Wrap Modes

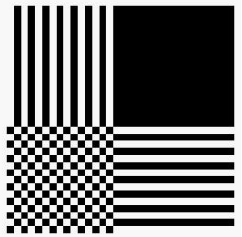
- Texture image is defined in [0..1]x[0..1] region
  - What happens outside that region?
  - Texture wrap modes say



texture



GL_REPEAT
wrapping



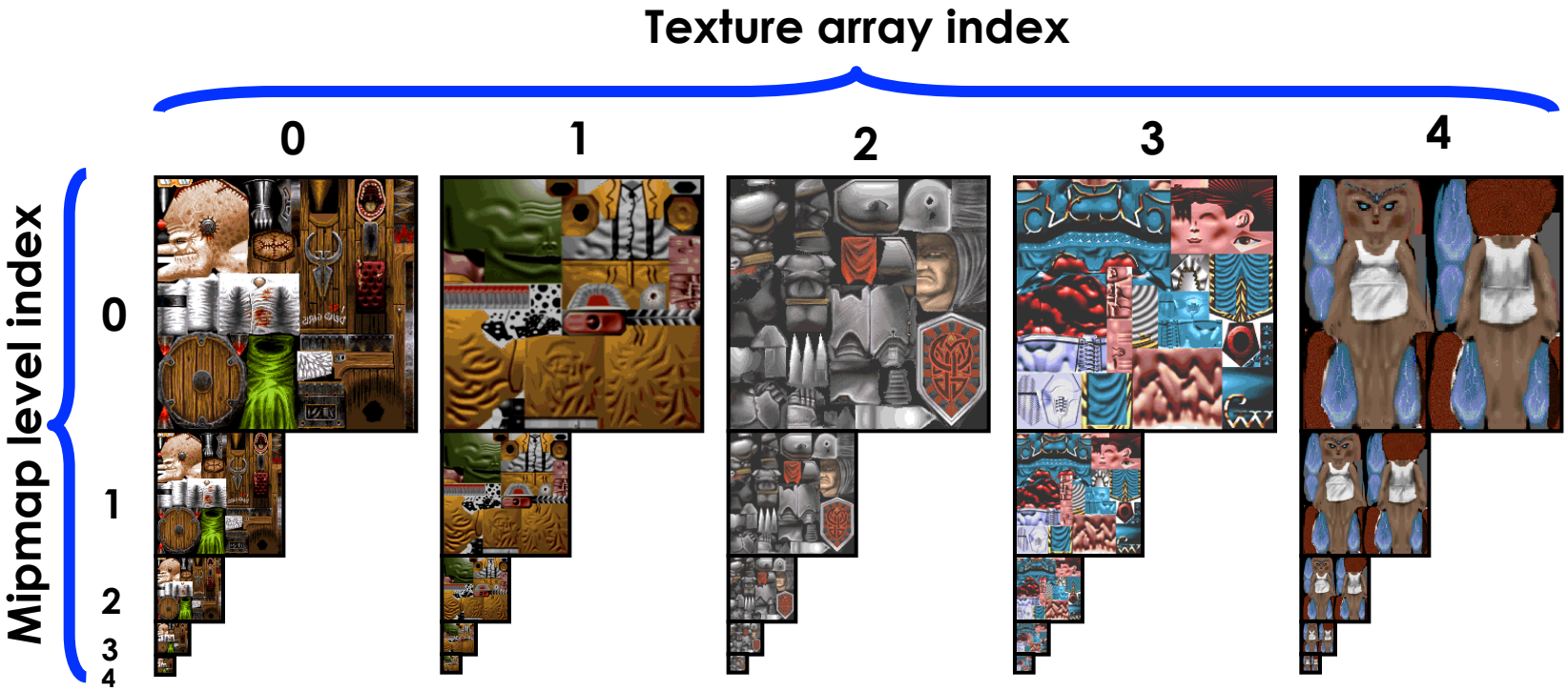
GL_CLAMP
wrapping

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



Anisotropic Texture Filtering

- Standard (isotropic) mipmap LOD selection
  - Uses magnitude of texture coordinate gradient (not direction)
  - Tends to spread blurring at shallow viewing angles
- Anisotropic texture filtering considers gradients direction
  - Minimizes blurring

Isotropic

Anisotropic

Texturing in WebGL: Vertex Shader

Need to alter the vertex shader to pass-through texture coordinates

```
attribute vec4 a_position;
attribute vec2 a_texcoord;
uniform mat4 uMVmatrix;
uniform mat4 uPMatrix;
varying vec2 v_texcoord;

void main() {
  gl_Position = uPMatrix* uMVmatrix * a_position;
  // Pass the texcoord to the fragment shader.
  v_texcoord = a_texcoord;
}
```

Texturing in WebGL: Fragment Shader

Need to alter the fragment shader to fetch colors from textures

```
precision mediump float;

// Passed in from the vertex shader.
varying vec2 v_texcoord;

// The texture.
uniform sampler2D u_texture;

void main() {
   gl_FragColor = texture2D(u_texture, v_texcoord);
}
```