Texture Filtering

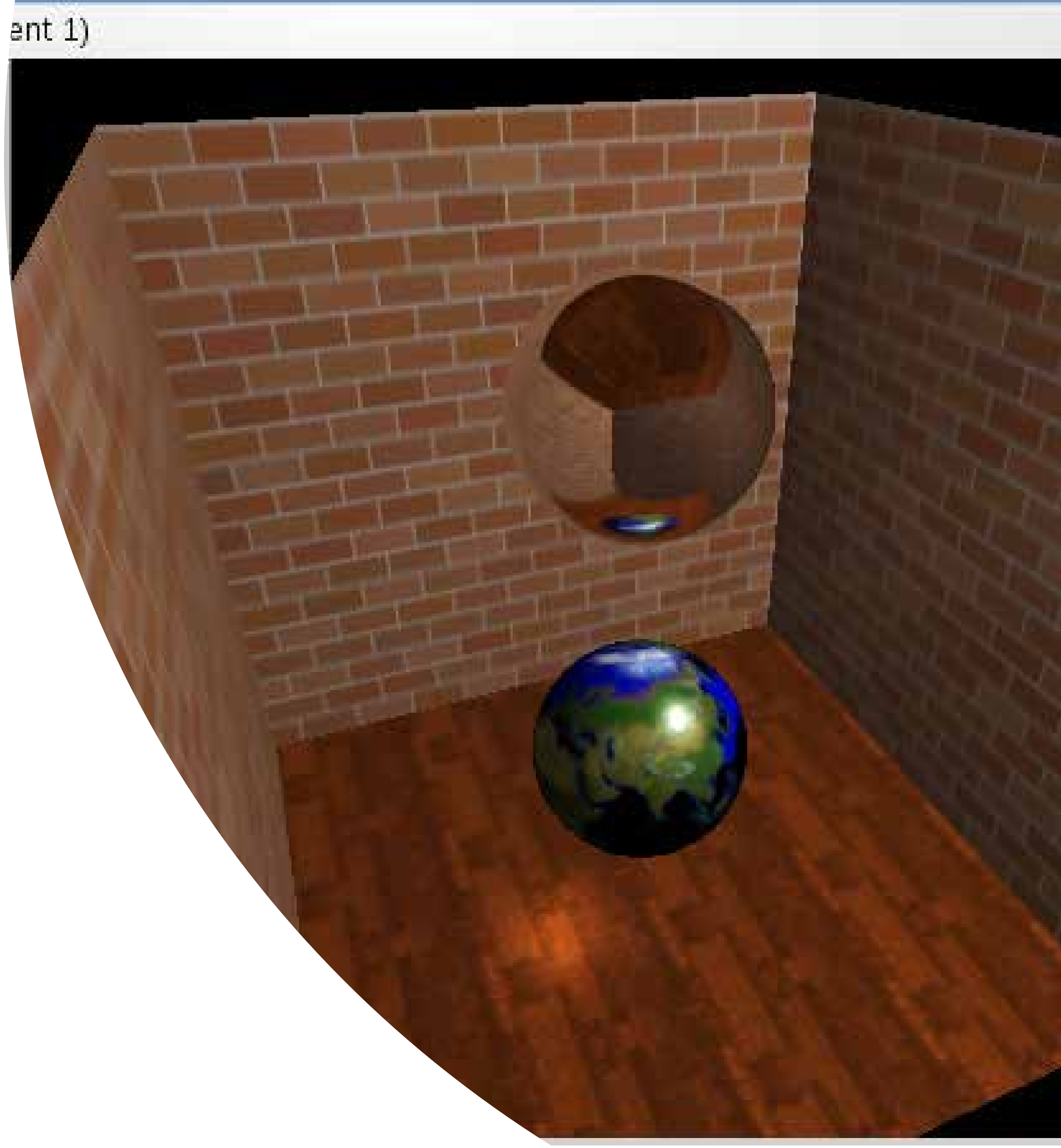
CS 418: Interactive Computer Graphics

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

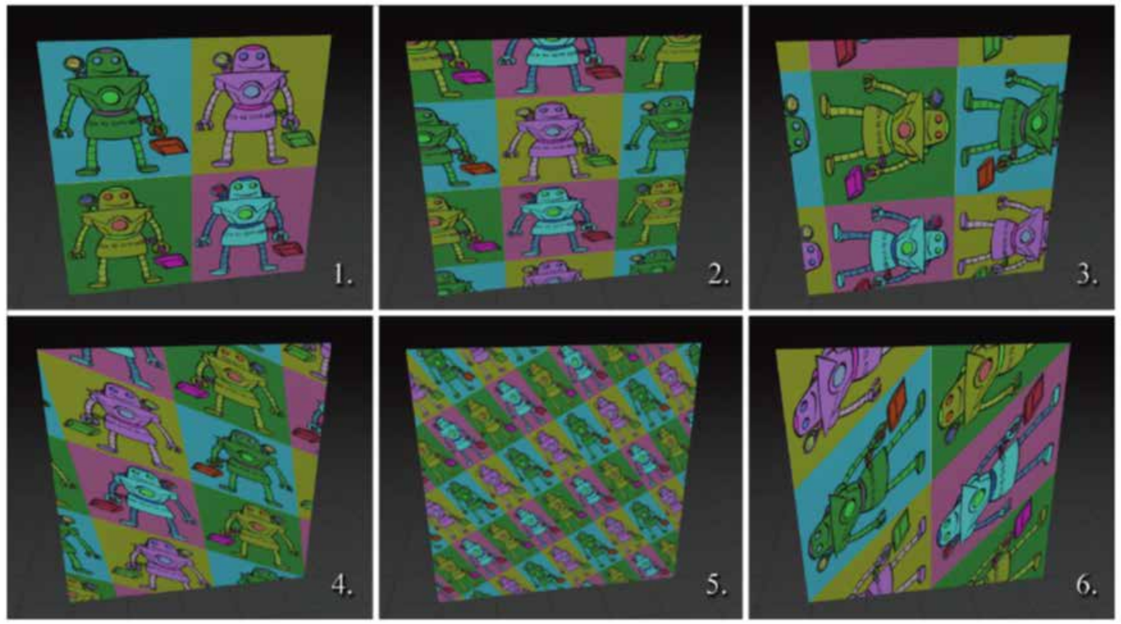
Eric Shaffer

Reviewing Texture 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...



Match the Coordinates to the Image

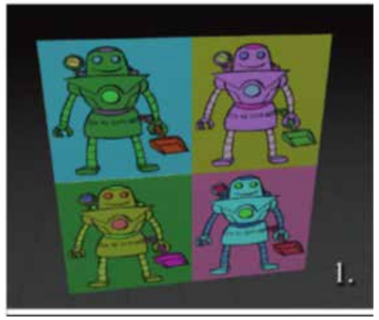


Match each textured quad with the set of texture coordinates used to generate it given in the list below.

(a)	$0 : (0.20, -0.30)$	$1 : (1.30, -0.30)$	$2 : (1.30, 1.20)$	$3 : (0.20, 1.20)$
(b)	$0 : (5.00, -1.00)$	$1 : (6.00, -1.00)$	$2 : (6.00, 0.00)$	$3 : (5.00, 0.00)$
(c)	$0 : (1.00, 0.00)$	$1 : (-0.23, -0.77)$	$2 : (0.00, 1.00)$	$3 : (1.24, 1.77)$
(d)	$0 : (2.00, 0.00)$	$1 : (1.00, 1.00)$	$2 : (0.00, 1.00)$	$3 : (1.00, 0.00)$
(e)	$0 : (-0.10, 1.10)$	$1 : (-0.10, 0.10)$	$2 : (0.90, 0.10)$	$3 : (0.90, 1.10)$
(f)	$0 : (0.00, -1.00)$	$1 : (3.35, 0.06)$	$2 : (1.00, 2.00)$	$3 : (-2.36, 0.94)$

For example

- The first image is axis aligned and doesn't repeat



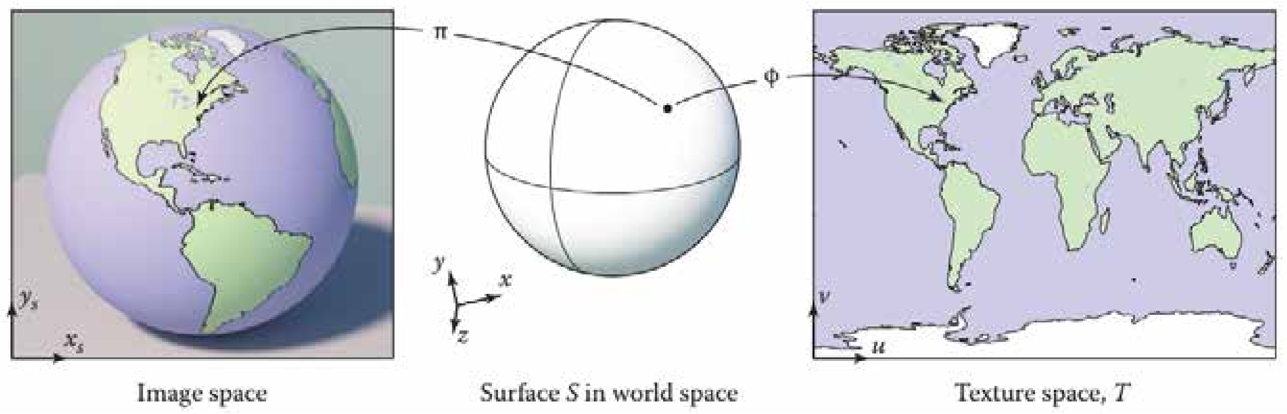
- Only possible coordinates from list are

(b) 0 : (5.00, −1.00) 1 : (6.00, −1.00) 2 : (6.00, 0.00) 3 : (5.00, 0.00)

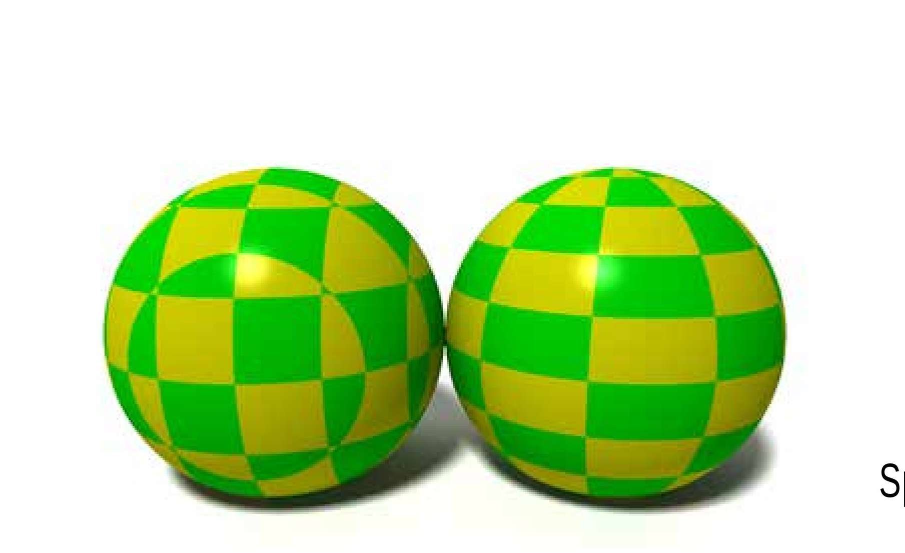
- Which give axis-aligned edges
- And parametric lengths of 1

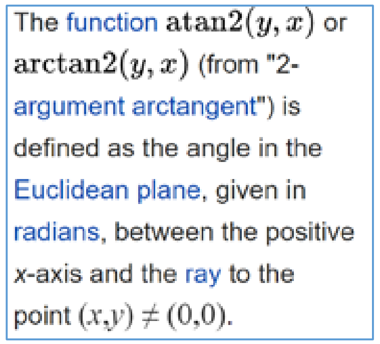
Texture Coordinates

- We've only looked at simple mappings
  - Mapping the texture onto planar surfaces
- More complicated surfaces need more complicated mappings

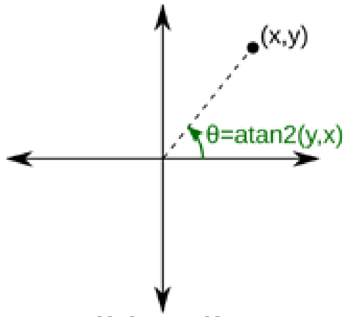


Example: Mapping onto a sphere



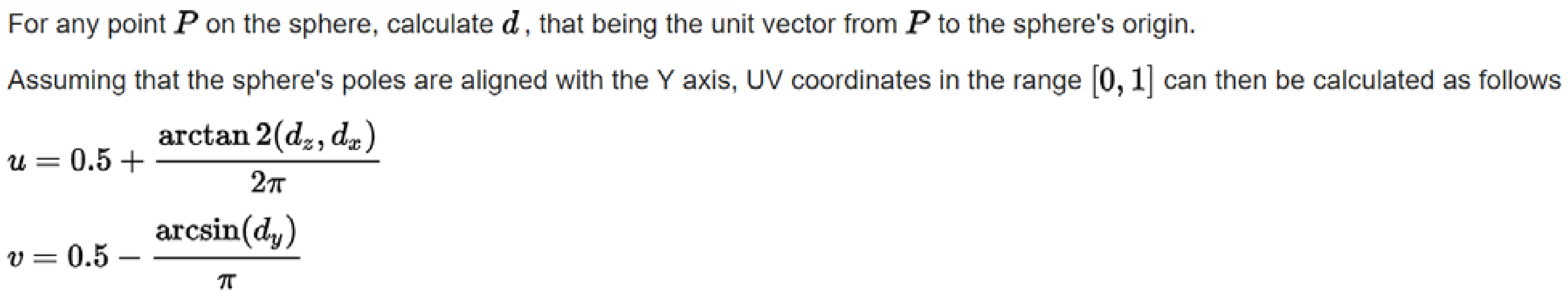


The function $\operatorname{atan2}(y, x)$ or $\operatorname{arctan2}(y, x)$ (from "2-argument arctangent") is defined as the angle in the Euclidean plane, given in radians, between the positive *x*-axis and the ray to the point $(x,y) \neq (0,0)$.



- Wikipedia

Sphere on the right uses:



For any point P on the sphere, calculate d, that being the unit vector from P to the sphere's origin.

Assuming that the sphere's poles are aligned with the Y axis, UV coordinates in the range $[0, 1]$ can then be calculated as follows

$$u = 0.5 + \frac{\arctan 2(d_z, d_x)}{2\pi}$$

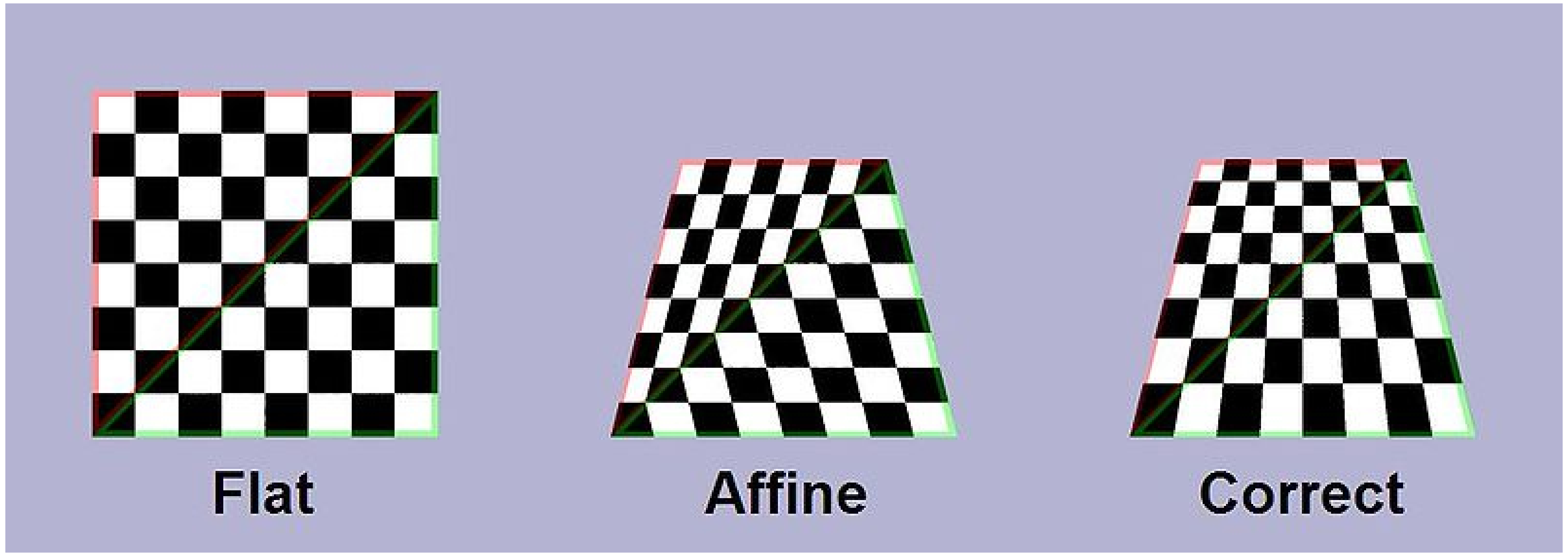
$$v = 0.5 - \frac{\arcsin(d_y)}{\pi}$$

How is the sphere on the left being textured?

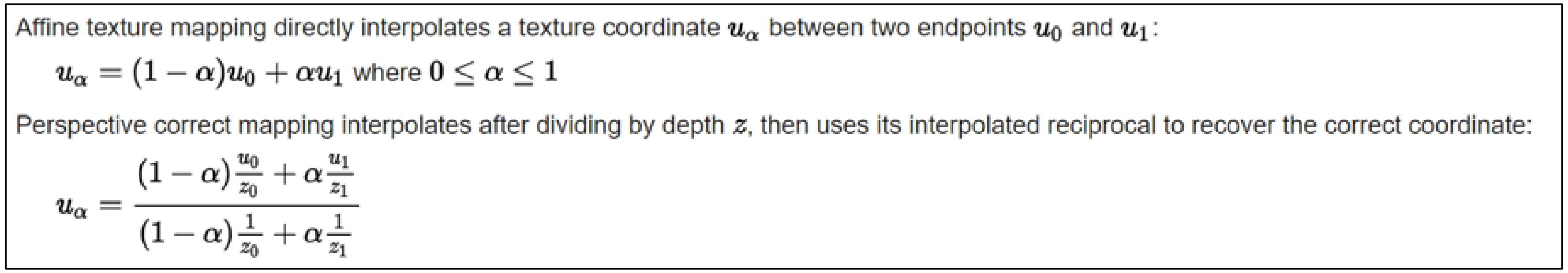
Perspective Correct Coordinates

- Linear (affine) interpolation problematic for texture coordinates

- Won’t produce perspective correct texture coordinates



Perspective Correct Coordinates



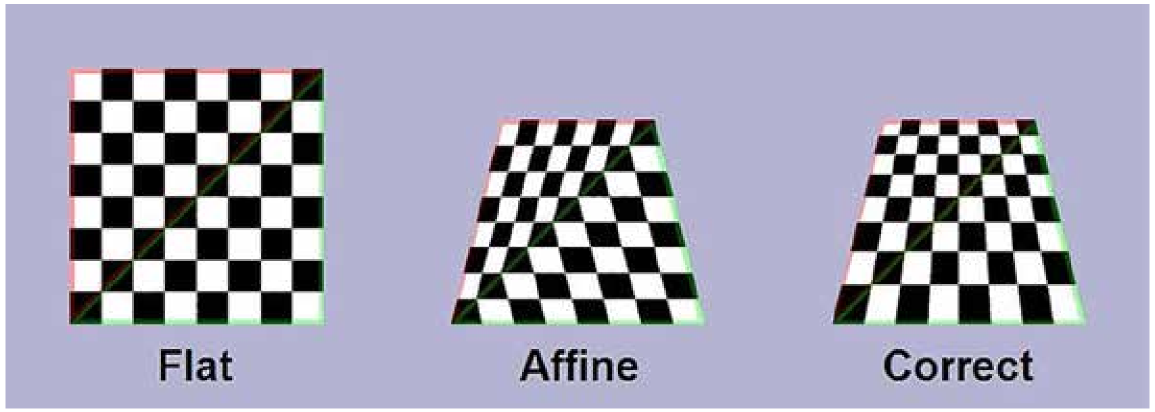
Affine texture mapping directly interpolates a texture coordinate u_α between two endpoints u_0 and u_1:

$$u_\alpha = (1-\alpha)u_0 + \alpha u_1 \text{ where } 0 \le \alpha \le 1$$

Perspective correct mapping interpolates after dividing by depth z, then uses its interpolated reciprocal to recover the correct coordinate:

$$u_\alpha = \frac{(1-\alpha)\frac{u_0}{z_0} + \alpha\frac{u_1}{z_1}}{(1-\alpha)\frac{1}{z_0} + \alpha\frac{1}{z_1}}$$

-- Wikipedia

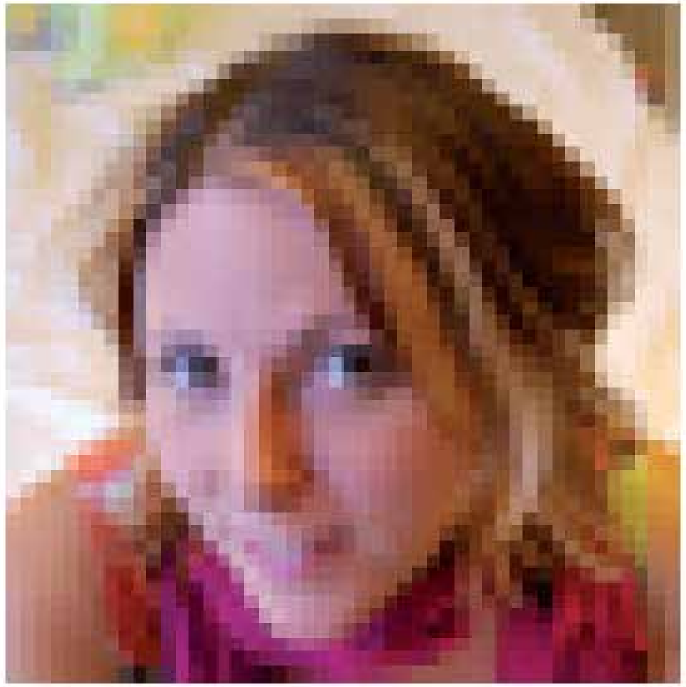


Texture Filtering

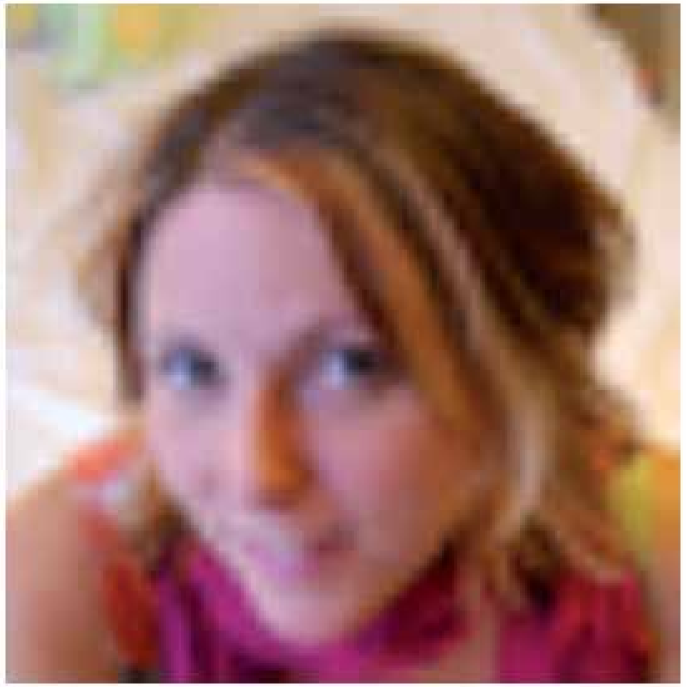
- We often have a mismatch between texture size and number of fragments
  - Requires us to adjust how the texture is sampled...
  - This more complicated sampling process is called ***texture filtering***
- ***Magnification*** occurs when we have more fragments than texels
- ***Minification*** occurs when we have more texels than fragments

- Two common mag filters
  - Nearest Neighbor
  - Bilinear
- Most common min filter
  - Mipmapping

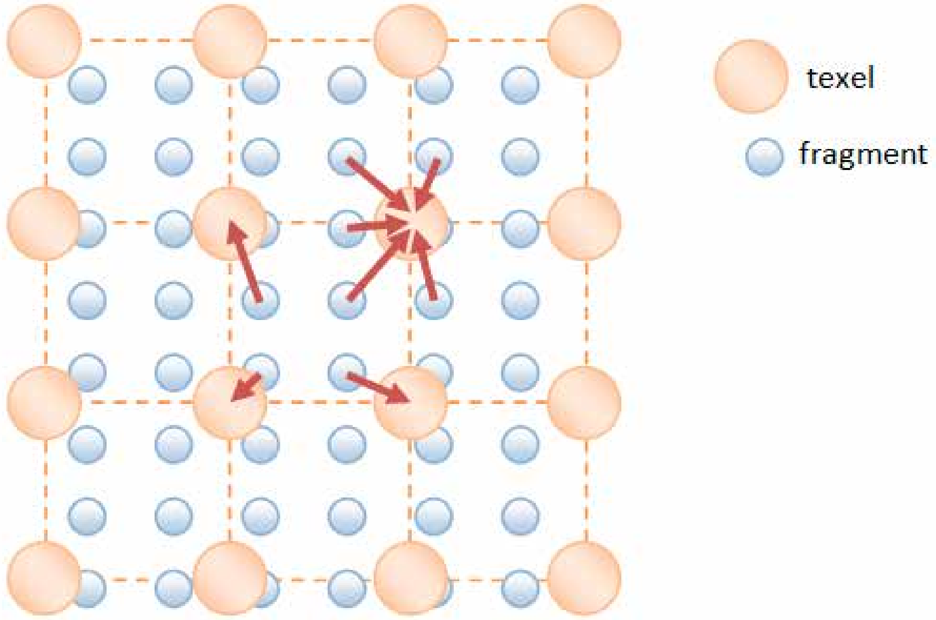
Magnification



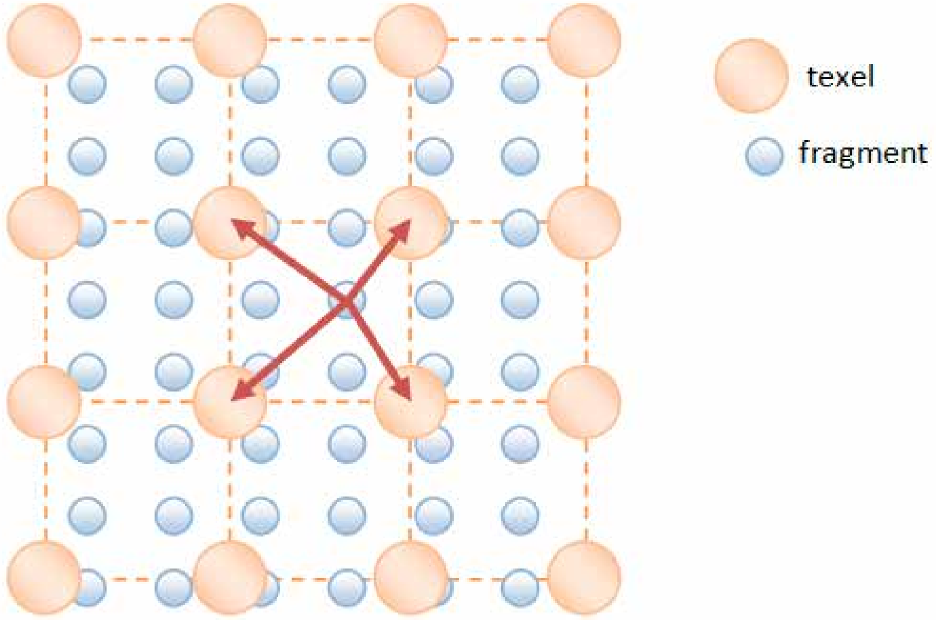
Nearest neighbor filtering



Bilinear Interpolation

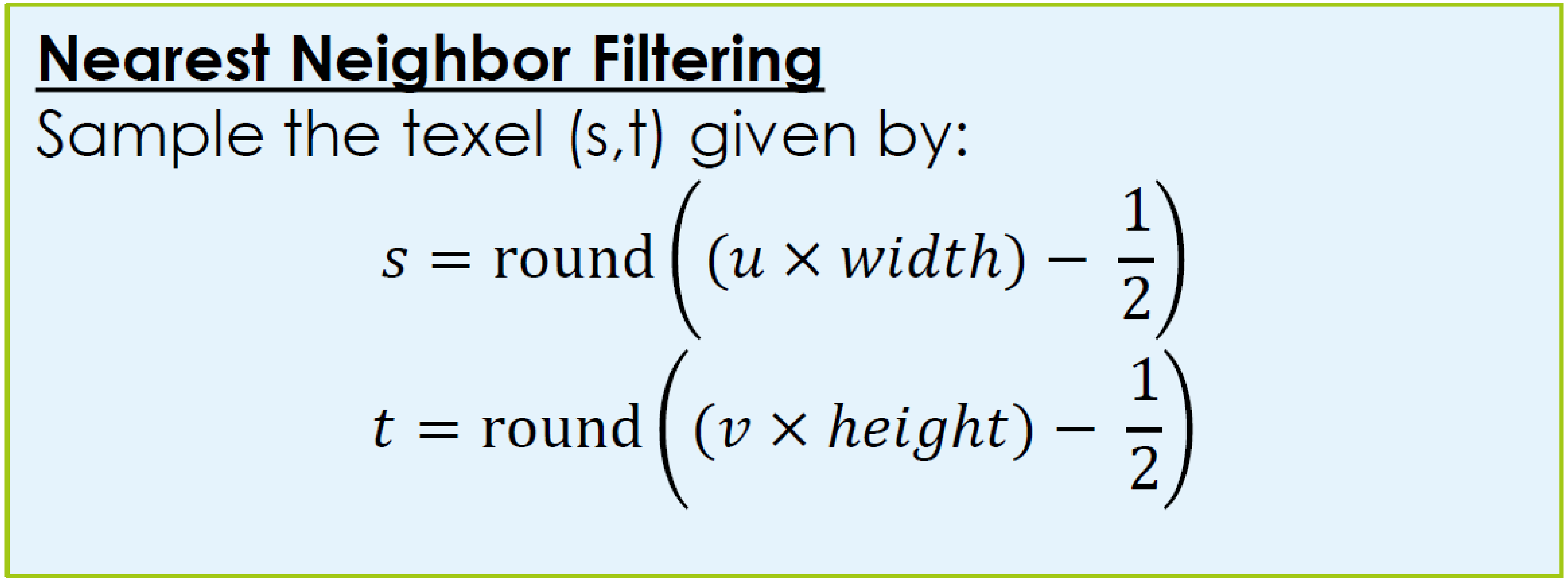


Magnification – Nearest Point Sampling



Magnification – Bilinear Interpolation

Magnification: Nearest Neighbor



Nearest Neighbor Filtering

Sample the texel (s,t) given by:

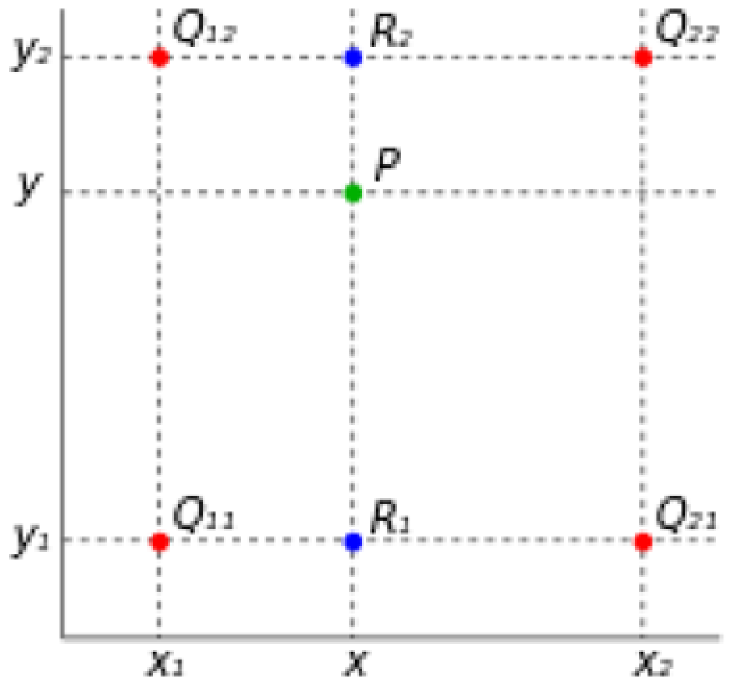
$$s = \text{round}\left((u \times width) - \frac{1}{2}\right)$$

$$t = \text{round}\left((v \times height) - \frac{1}{2}\right)$$

Magnification: Bilinear Interpolation

- In bilinear interpolation, we estimate a value for a function
  - On a 2D grid…with function samples at the grid vertices
- We interpolate first in one direction (e.g. the x direction)
  - Interpolate using linear interpolation twice
  - Find 2 points…one on each edge
- Then interpolate in the other direction (e.g. the y direction)
  - Linear interpolation again
  - Between the two points from the first round of interpolation

Magnification: Bilinear Interpolation



$$f(x, y_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}),$$
$$f(x, y_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}).$$

$$(x, y) \approx \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2)$$

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?

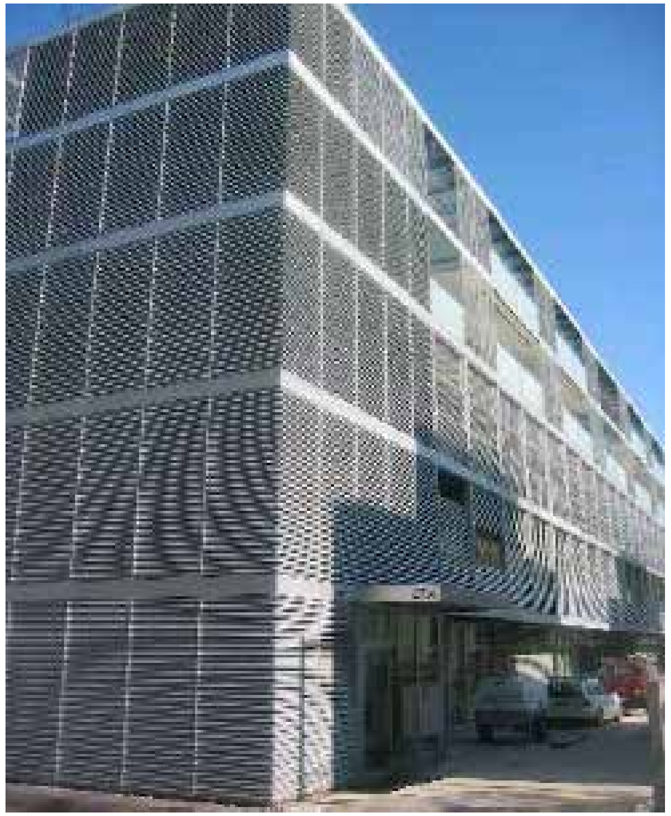
Filtering Textures

- 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 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



Minification

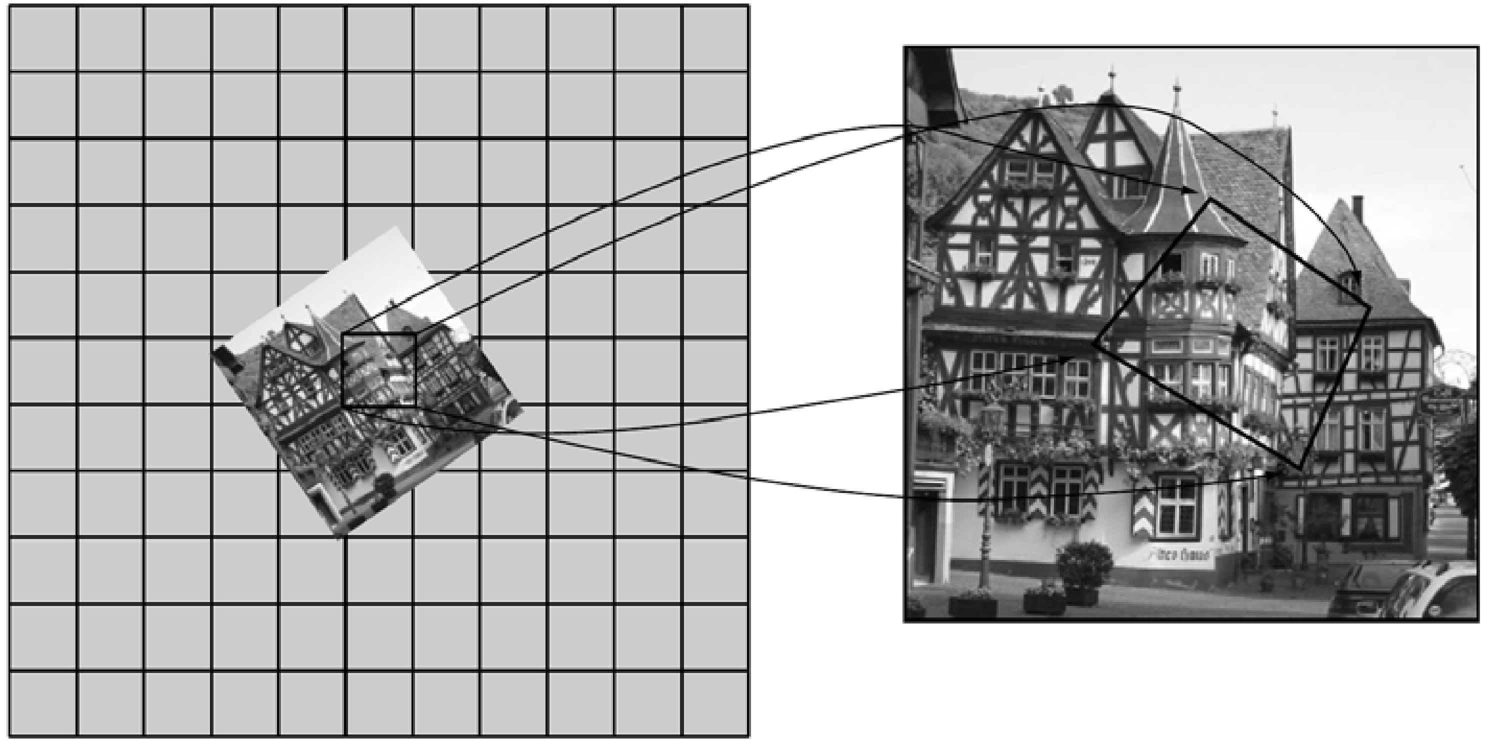


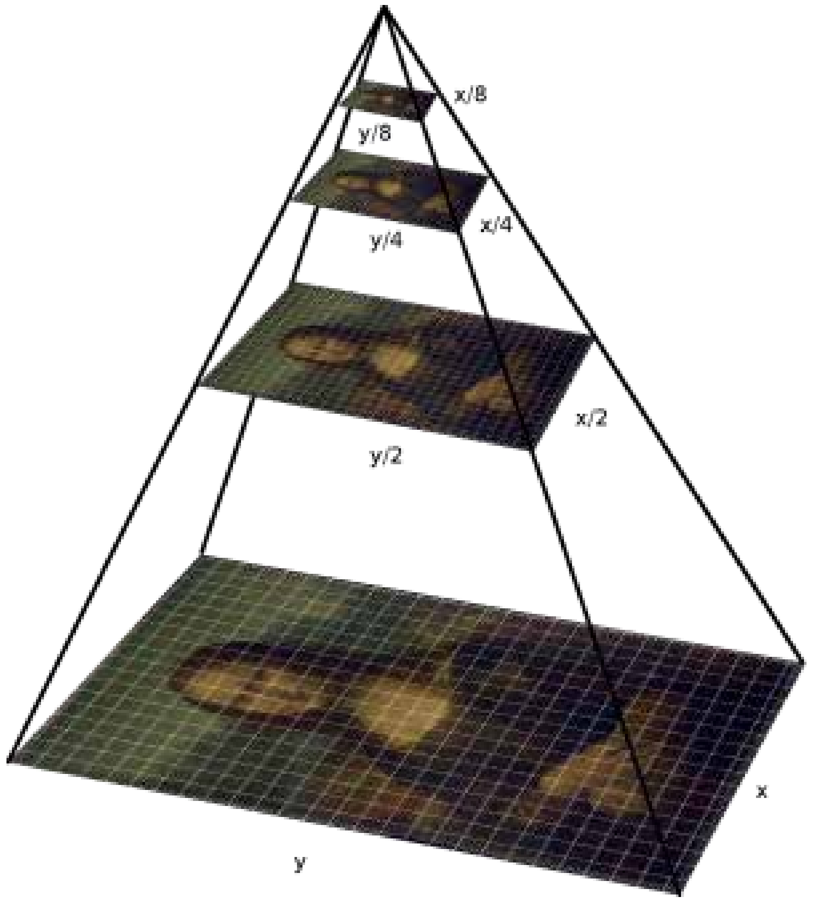
Figure 9.18 Mapping the square screen-space area of a pixel back into texel space: (a) screen space with pixel of interest highlighted and (b) texel-space back-projection of pixel area.

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

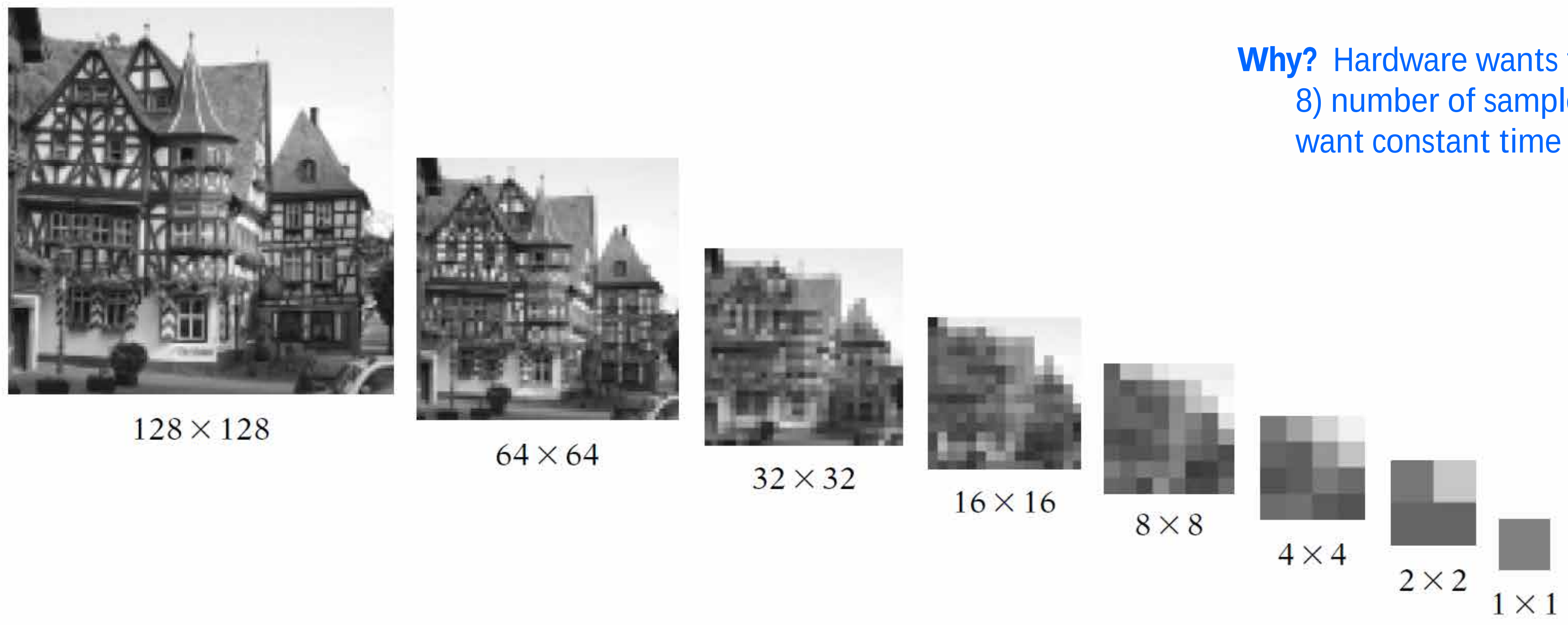


Pre-filtered Image Versions

- 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

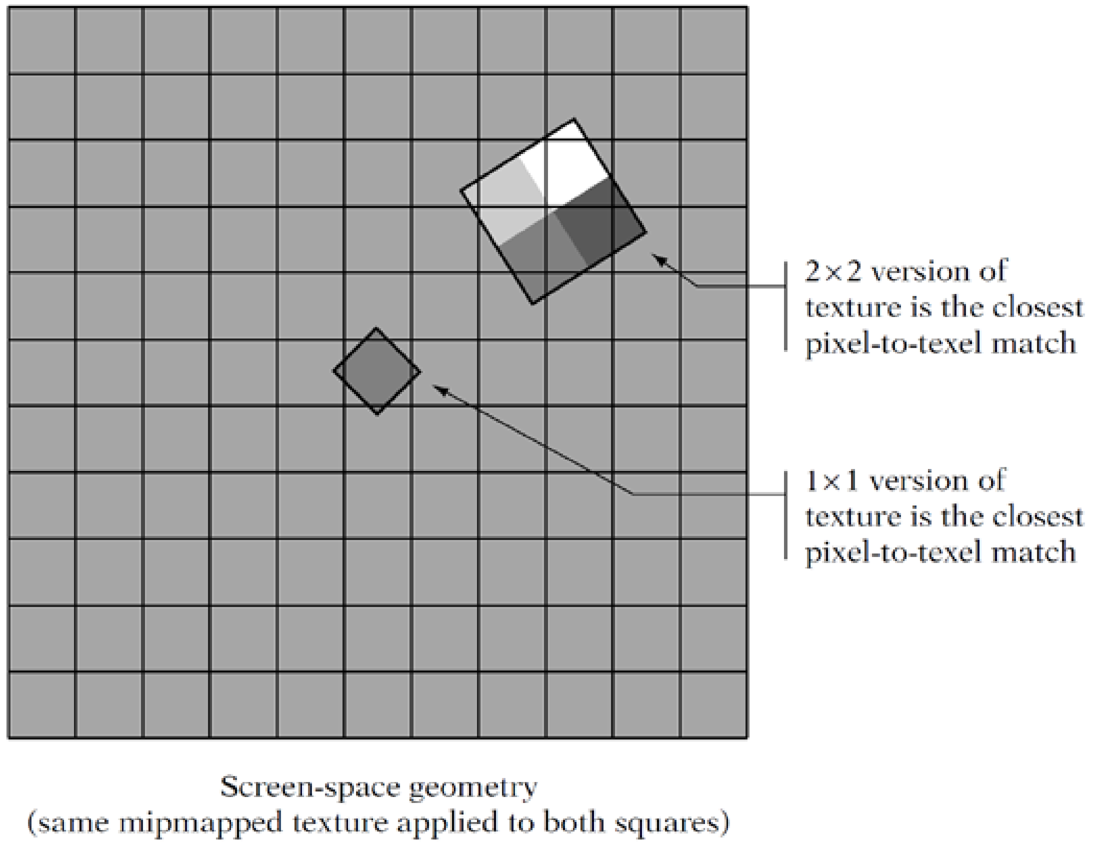

2×2 version of texture is the closest pixel-to-texel match
1×1 version of texture is the closest pixel-to-texel match
Screen-space geometry (same mipmapped texture applied to both squares)

FIGURE 9.19 Choosing between two sizes of a texture.

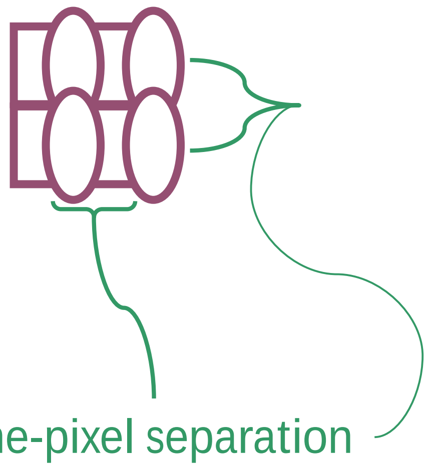
Mipmap Level-of-detail Selection

- 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
    -

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
    -
    - Means 4 subtractions per quad (1 per pixel)

- Now compute approximation to gradient length
  - p = max(sqrt ^{2} ^{2}), sqrt ^{2} ^{2}))



Level-of-detail Bias and Clamping

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

- Now clamp to valid LOD range
  - = max(minLOD, min(maxLOD,))

Determine Mipmap Levels

- Determine lower and upper mipmap levels
  - b = floor(   )) is bottom mipmap level
  - t = floor(+1) is top mipmap level

- Determine filter weight between levels
  - w = frac(   ) 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

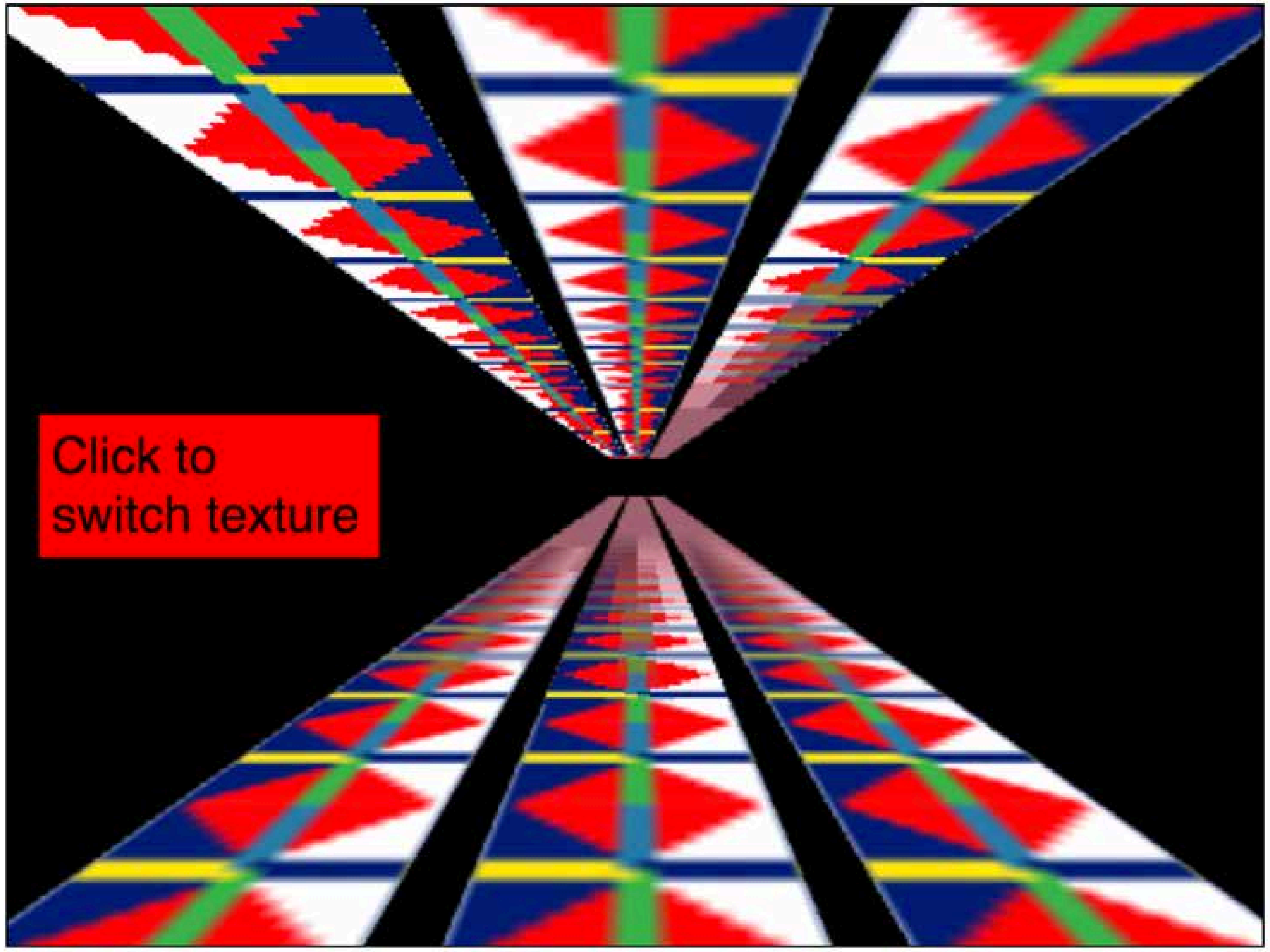
NEAREST_MIPMAP_NEAREST = choose the best mip,
then pick one pixel from that mip

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

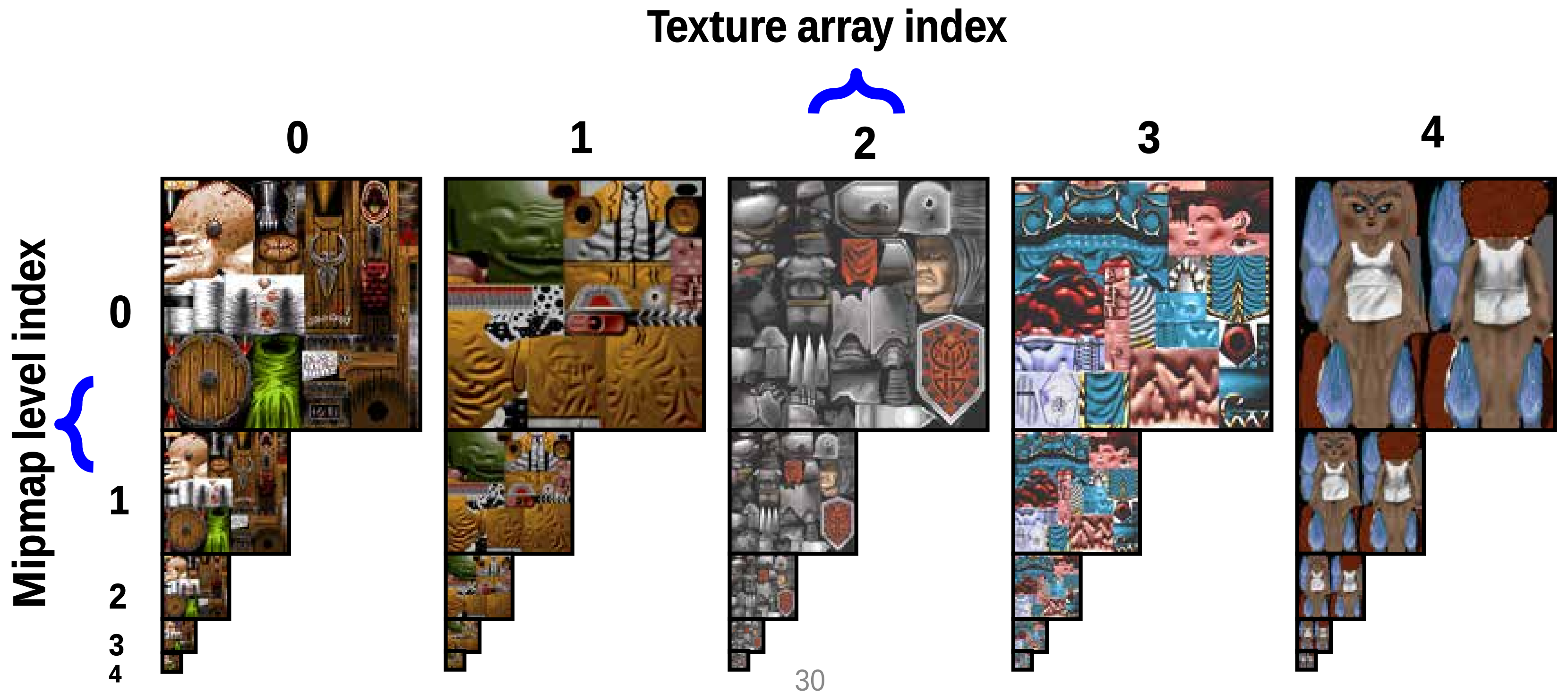
WebGL: Non-power of 2 textures

- 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


Texture array index
0
1
2
3
4
Mipmap level index
0
1
2
3
4