ARLAB

ME/CprE/ComS 557

Computer Graphics and Geometric Modeling

Texture Mapping with OpenGL

October 15th, 2015

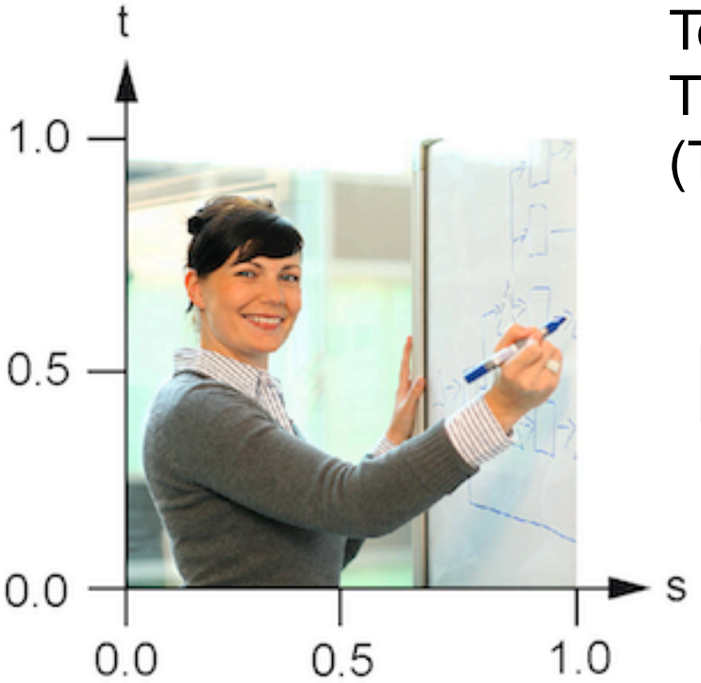
Rafael Radkowski

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Content

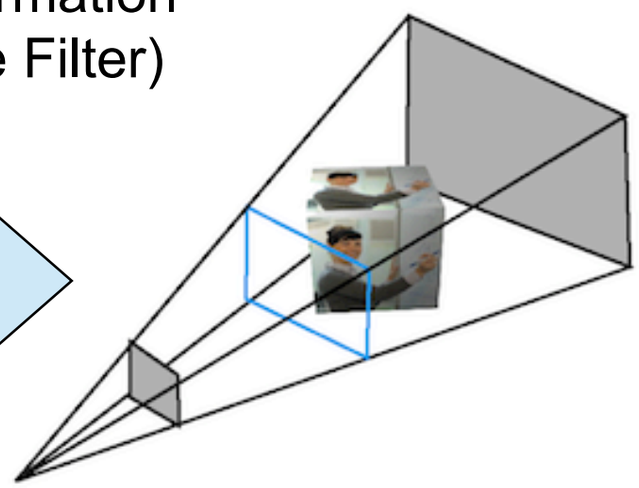
- Texture Filter Reminder
- Bilateral Filter
- Midmaps
- Trilinear Filter

The Texture Rendering Process



Texture

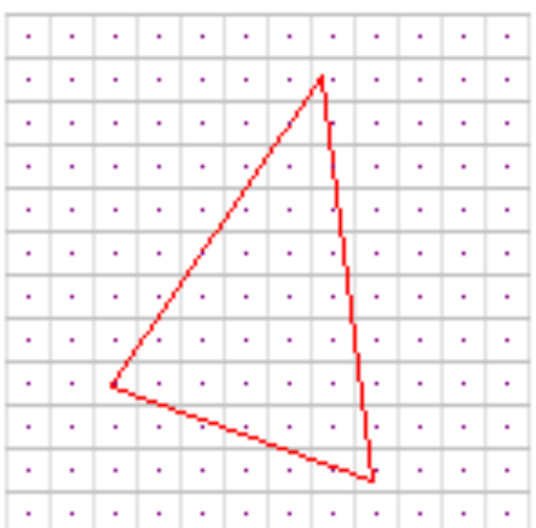
Texture Object Transformation (Texture Filter)

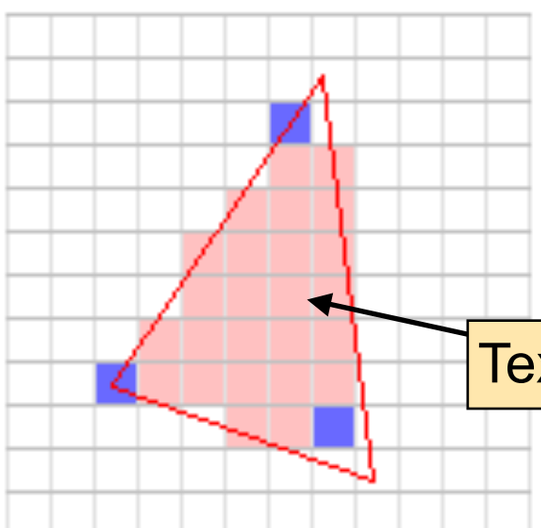


Texture mapped to a object and perspective distorted

Screen Transformation

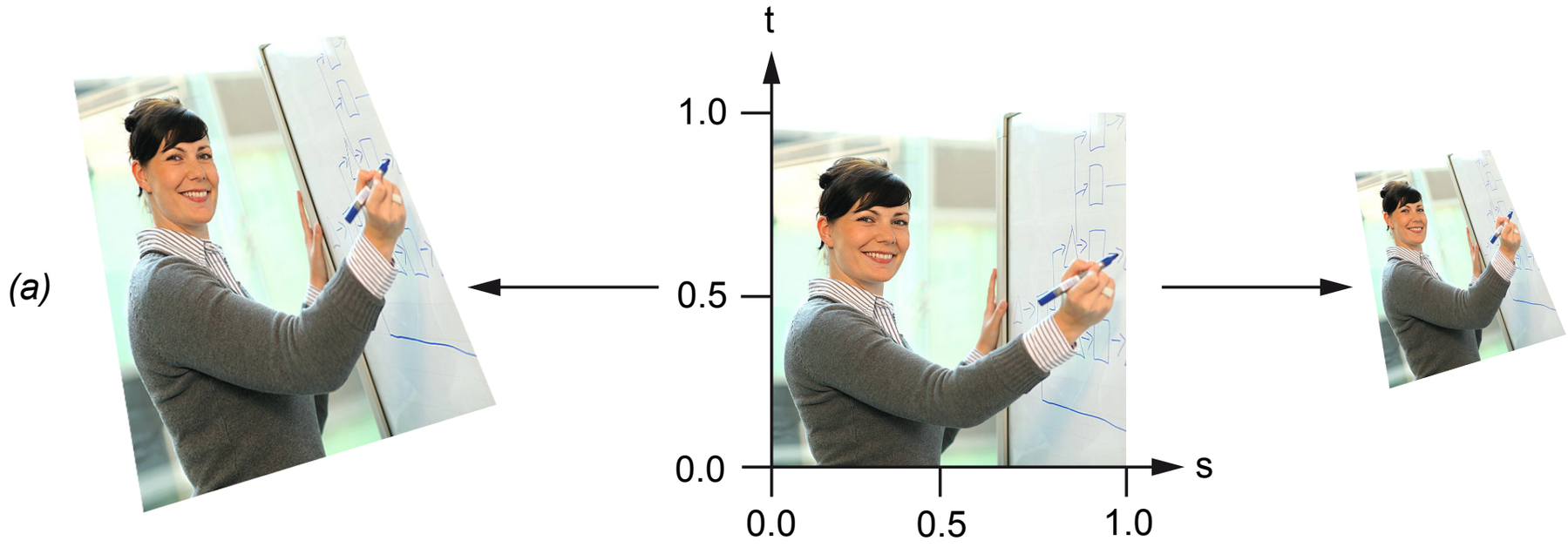
The output on screen





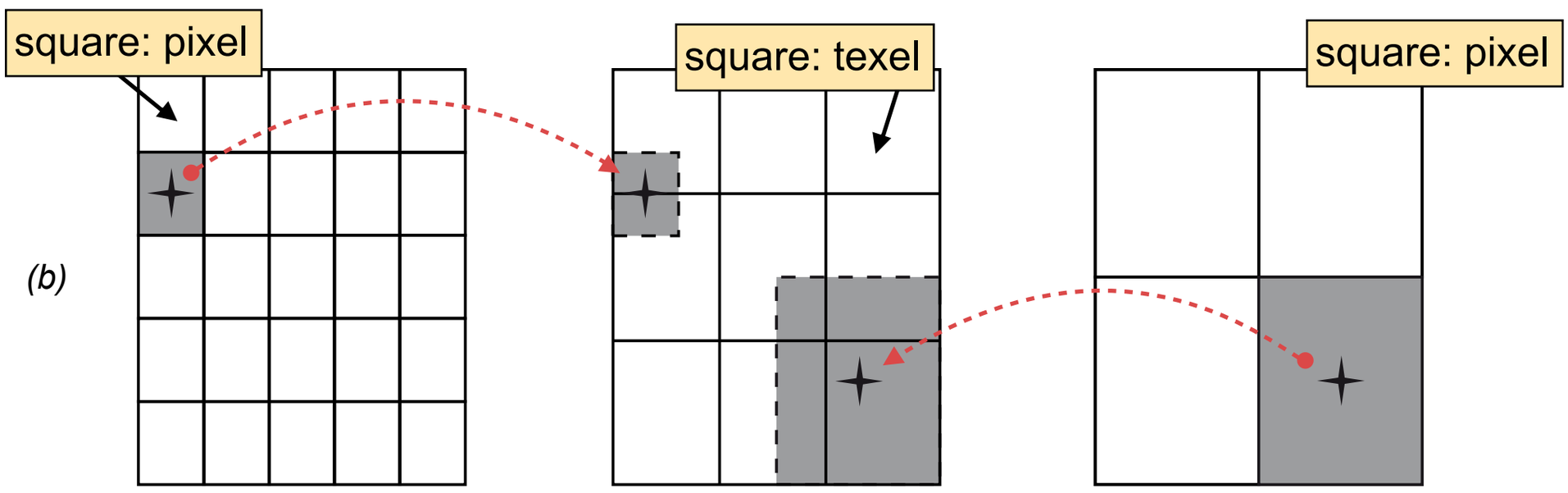
The graphics card must identify the correct color for each fragment (pixel) of a primitive.

Magnification & Minifying

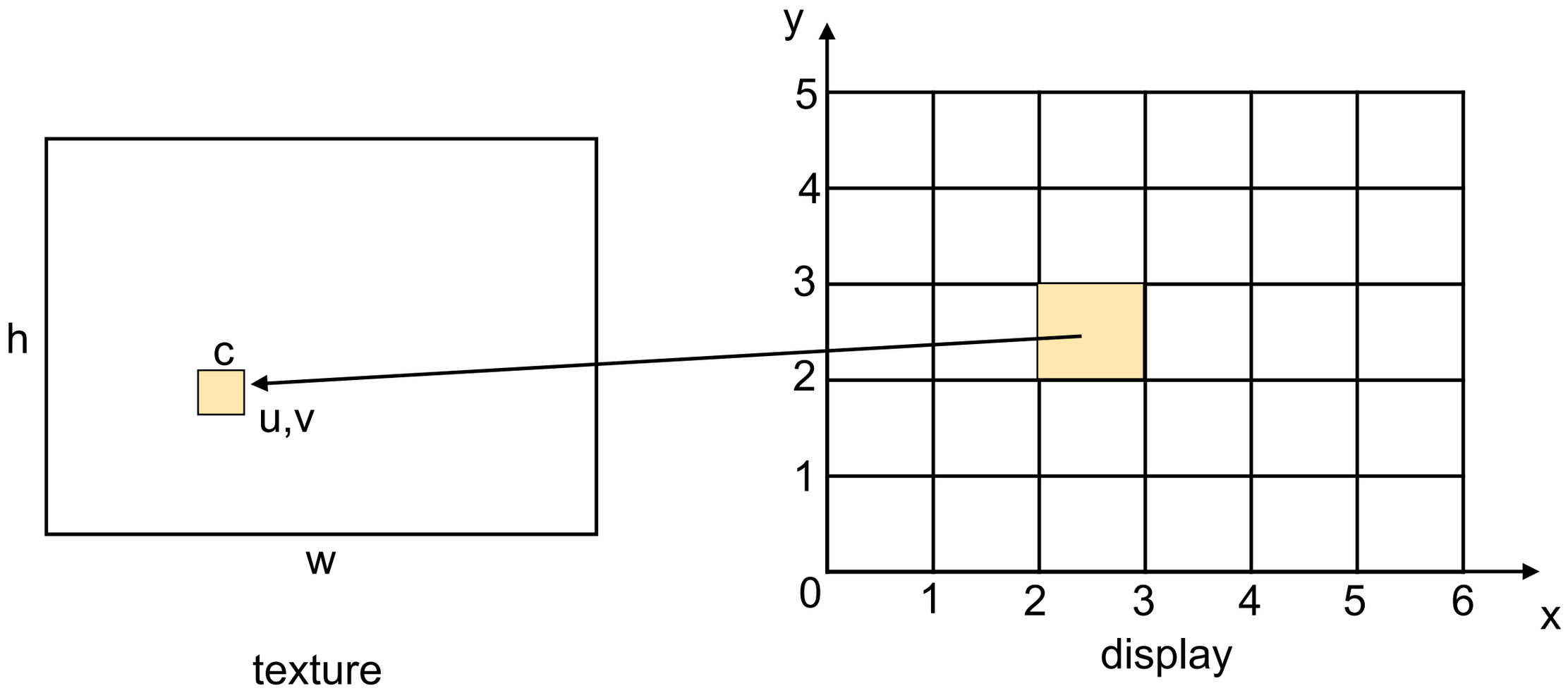


Texture Magnification Function

Texture Minifying Function



Color selection

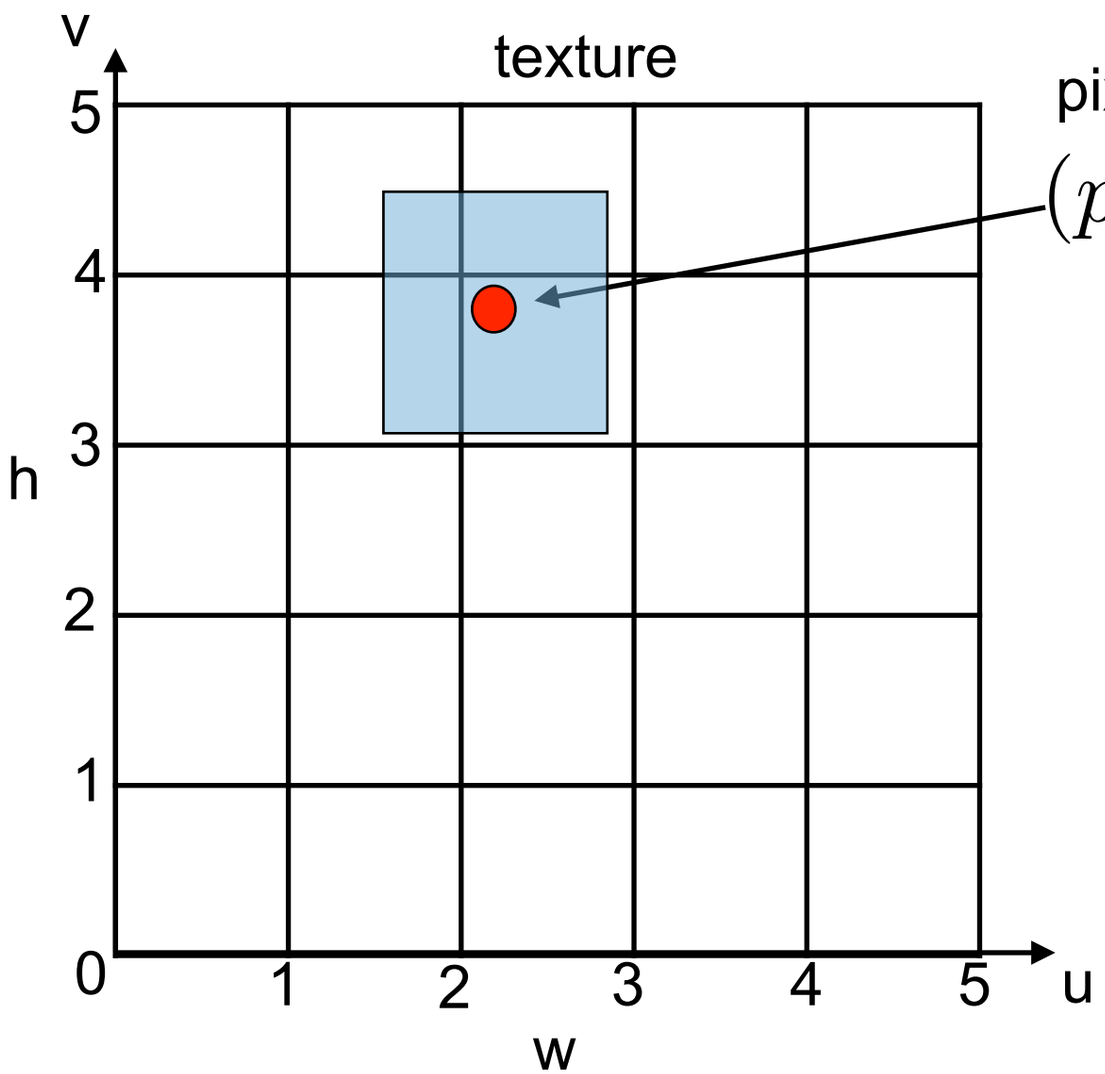


We need to find the color value c in the texture with size h and w in pixels, for fragment i,j.

$$v = \lfloor x\ w \rfloor \qquad \text{and} \qquad v = \lfloor y\ h \rfloor$$

$\lfloor x \rfloor$ is the *floor function.* it gives the highest inter value $\leq x$

Point Sampling / Nearest Neighbor



pixel center

(p_x, p_y)

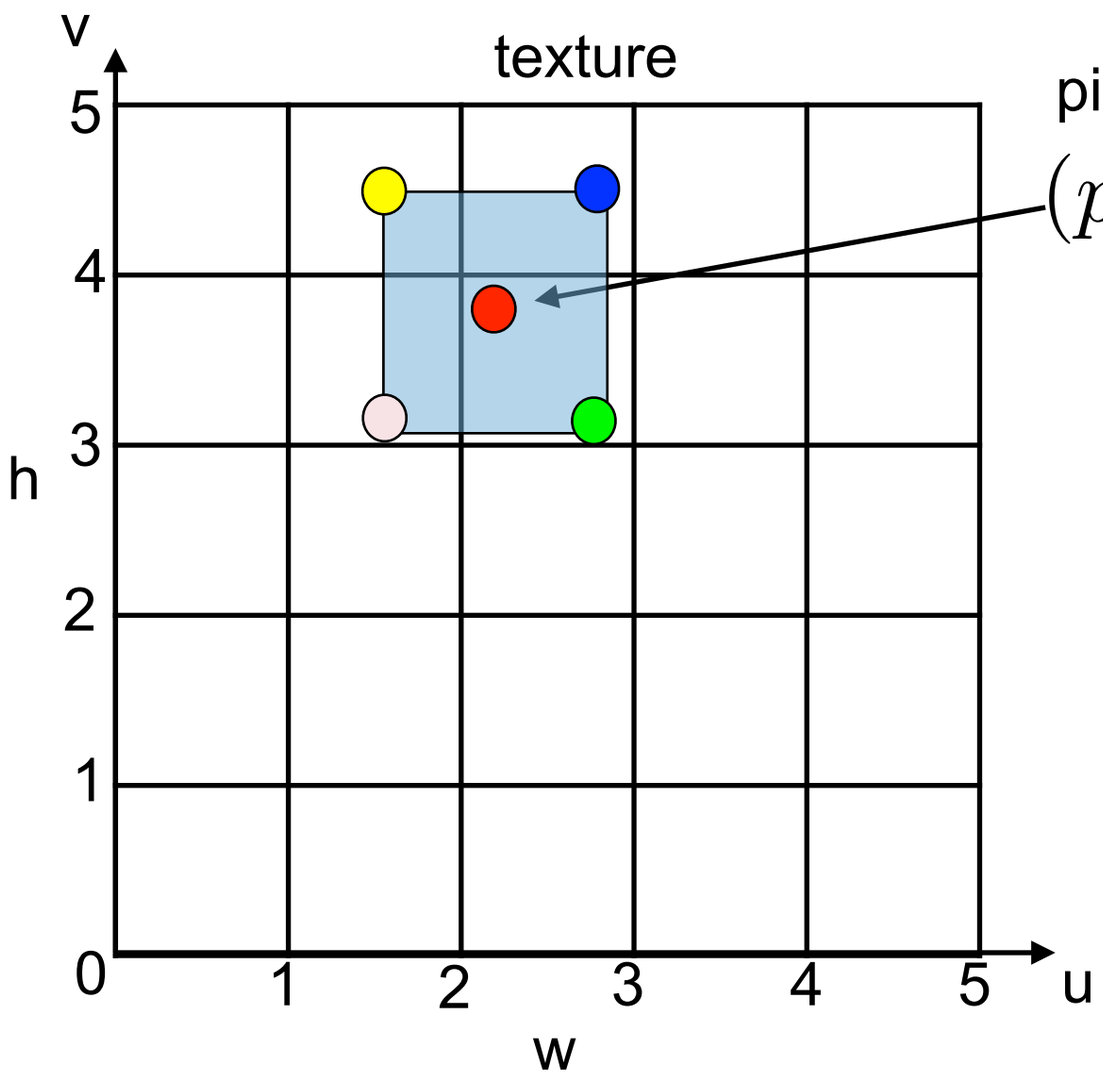
Here, we take the nearest color *c* for our vertex coordinate

$c(u, v) = c_{x,y}$ with

$u = \lfloor p_x \; w \rfloor$

$v = \lfloor p_y \; h \rfloor$

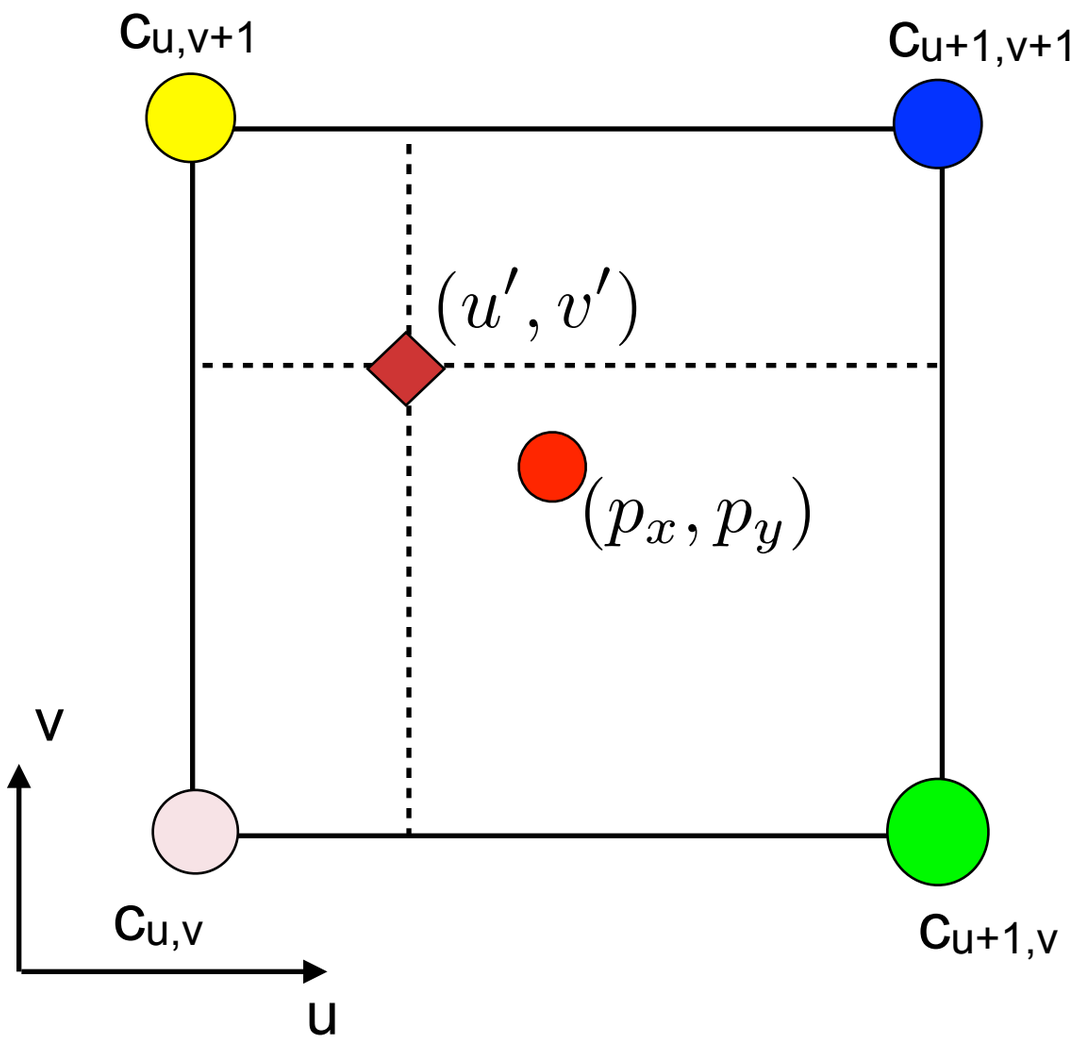
Bilinear Interpolation



Bilinear interpolation is an extension of linear interpolation.

The key idea is to perform linear interpolation first in one direction, and then again in the other direction.

Bilinear Interpolation



Relative position - range 0 to 1 - within four adjacent texel positions

$$(u', v') = (p_x - \lfloor p_x \rfloor, p_y - \lfloor p_y \rfloor)$$

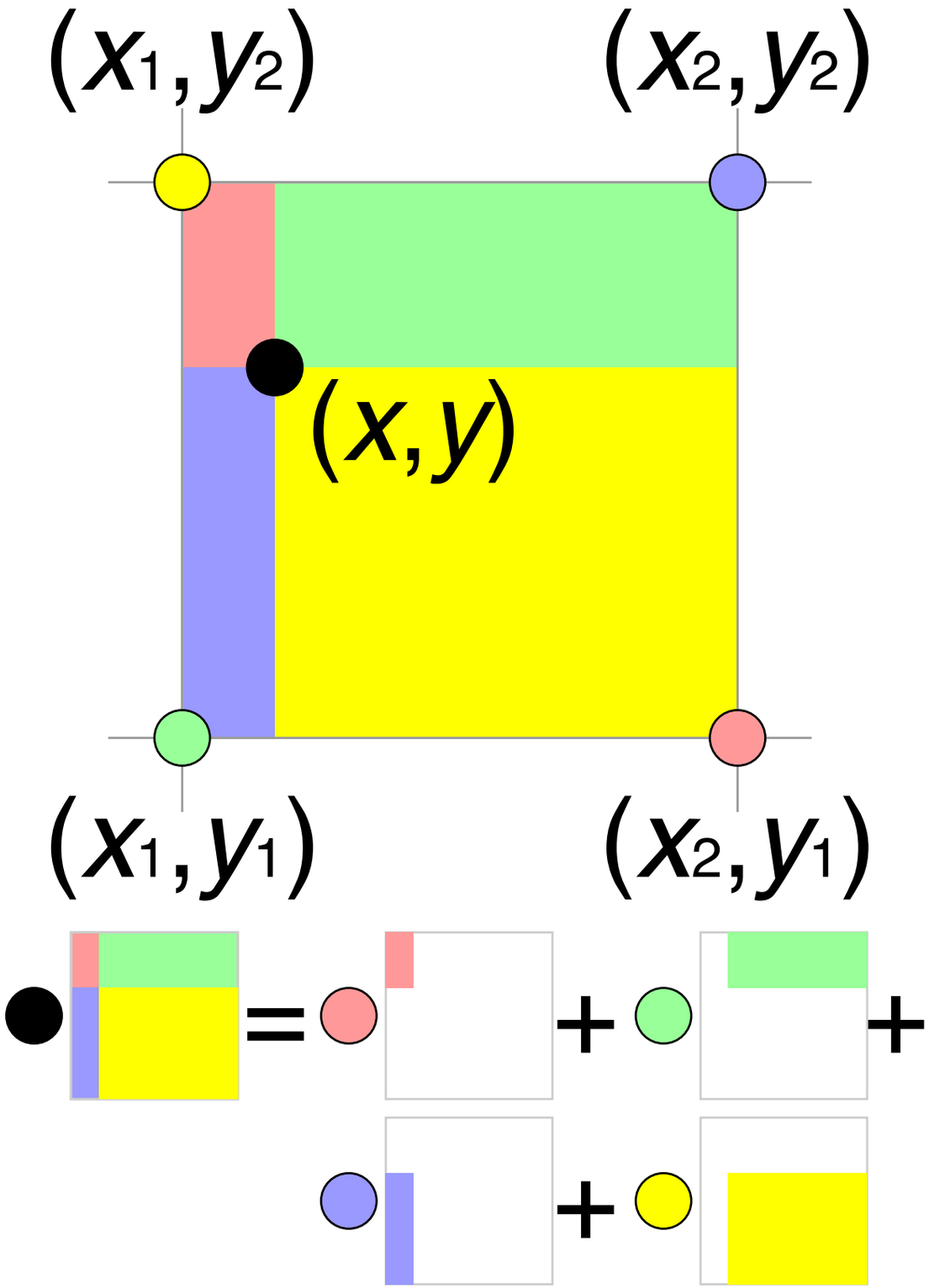
Color value

$$\begin{aligned} c_{x,y} = (1-u')(1-v')c_{u,v} \\ +u'(1-v')v_{u+1,v} \\ +(1-u')v'c_{u,v+1} \\ +u'v'c(u+1,v+1) \end{aligned}$$

- The "opposite" color has more weight
- Note, all fragments add up to 1.

ARLAB

Bilinear Interpolation



source: wikipedia

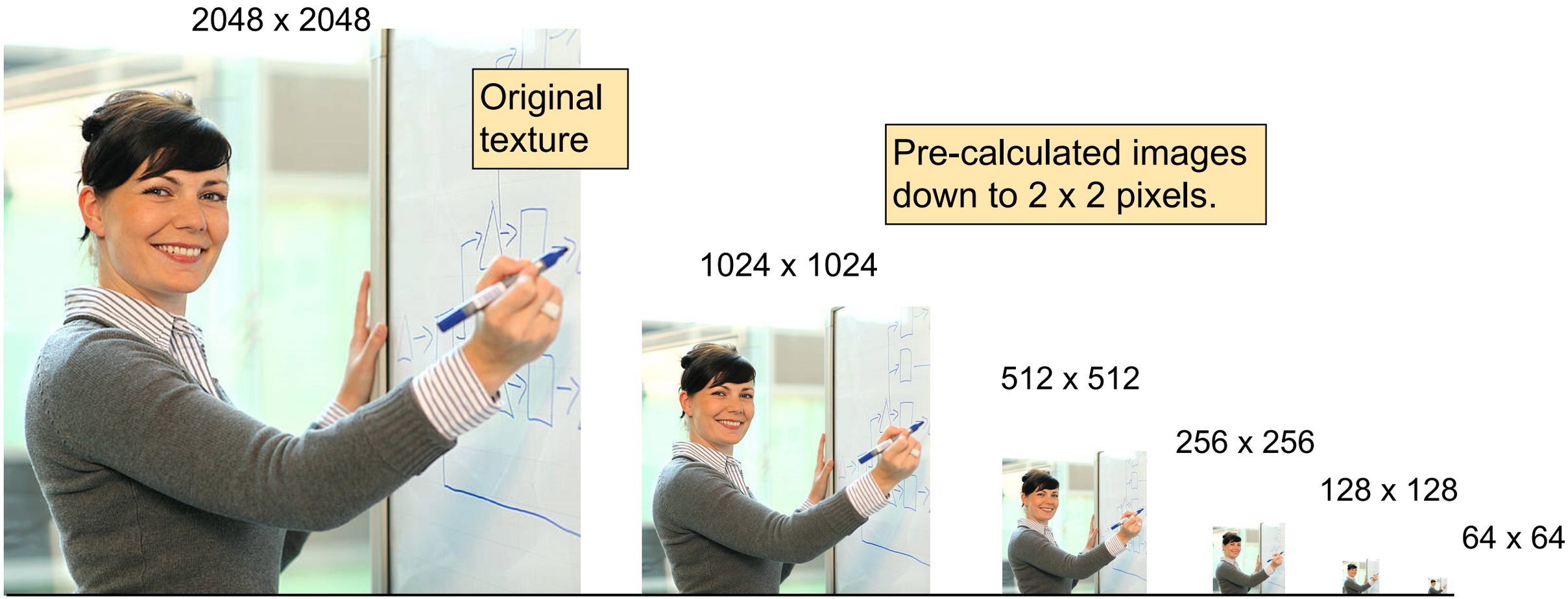
Limitations

Bilinear filtering is rather accurate until the scaling of the texture gets below half or above double the original size of the texture - that is, if the texture was 256 pixels in each direction, scaling it to below 128 or above 512 pixels can make the texture look bad, because of missing pixels or too much smoothness.

MipMap-Filter

multum in parvo, meaning "much in little"

VRAC | HCI



Filtering for minification is expensive, and different areas must be averaged depending on the amount of minification

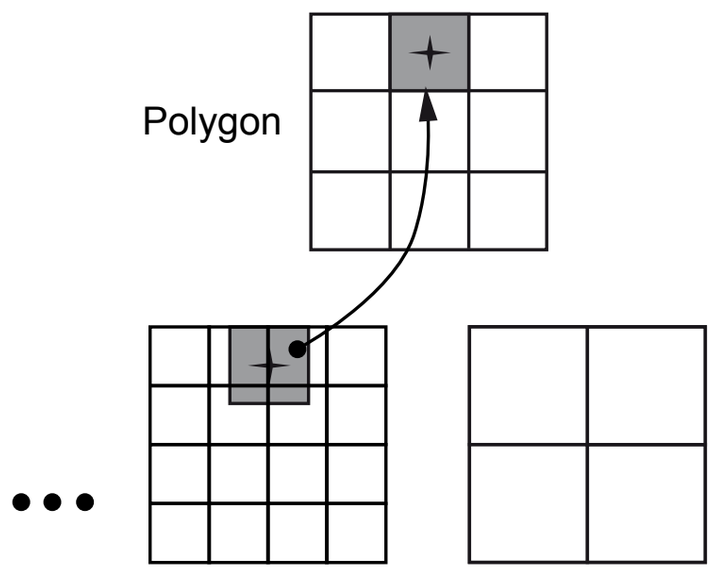
MipMap-Approach

- Prefilter different texture images at different resolutions
- For each screen pixel, choose texture that has an appropriate size

MipMap-Minifying Filter

VRAC|HCI

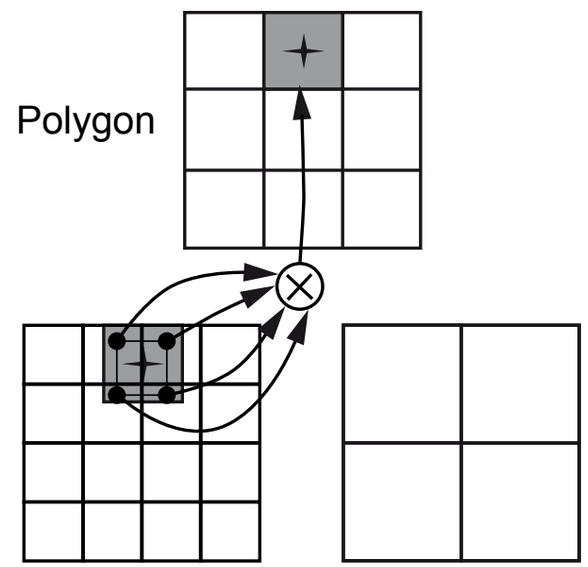
Point Sampling



MipMap-Level n ≤ λ ≤ n+1

*GL_ NEAREST_MIPMAP_NEAREST**

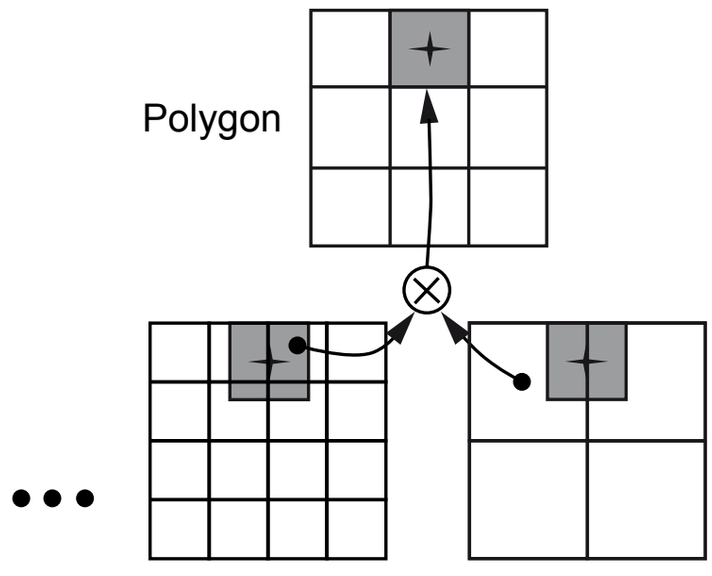
Bilineare Filter



MipMap-Level n ≤ λ ≤ n+1

GL_LINEAR_MIPMAP_NEAREST

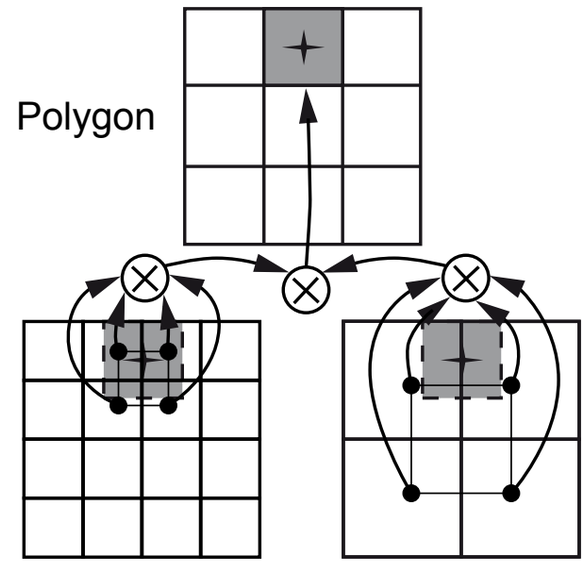
Trilineare Filter



MipMap-Level n ≤ λ ≤ n+1

GL_ NEAREST_MIPMAP_LINEAR

Trilineare Filter



MipMap-Level n ≤ λ ≤ n+1

GL_LINEAR_MIPMAP_LINEAR

Calculating the MipMap Lavel:

$$\lambda = log_2(max(\frac{1}{\rho_x}, \frac{1}{\rho_y}))$$

ρ_x, ρ_y :Scaling factors, provided by the graphics card.

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Calculating the MidMap Level

The level depends on a scale factor ρ(x, y) and the level-of-detail parameter λ(x, y), defined as

$$\lambda_{base}(x,y) = \log_2[\rho(x,y)]$$

with

$$\rho = \max\left\{\sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial w}{\partial x}\right)^2}, \sqrt{\left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial w}{\partial y}\right)^2}\right\}$$

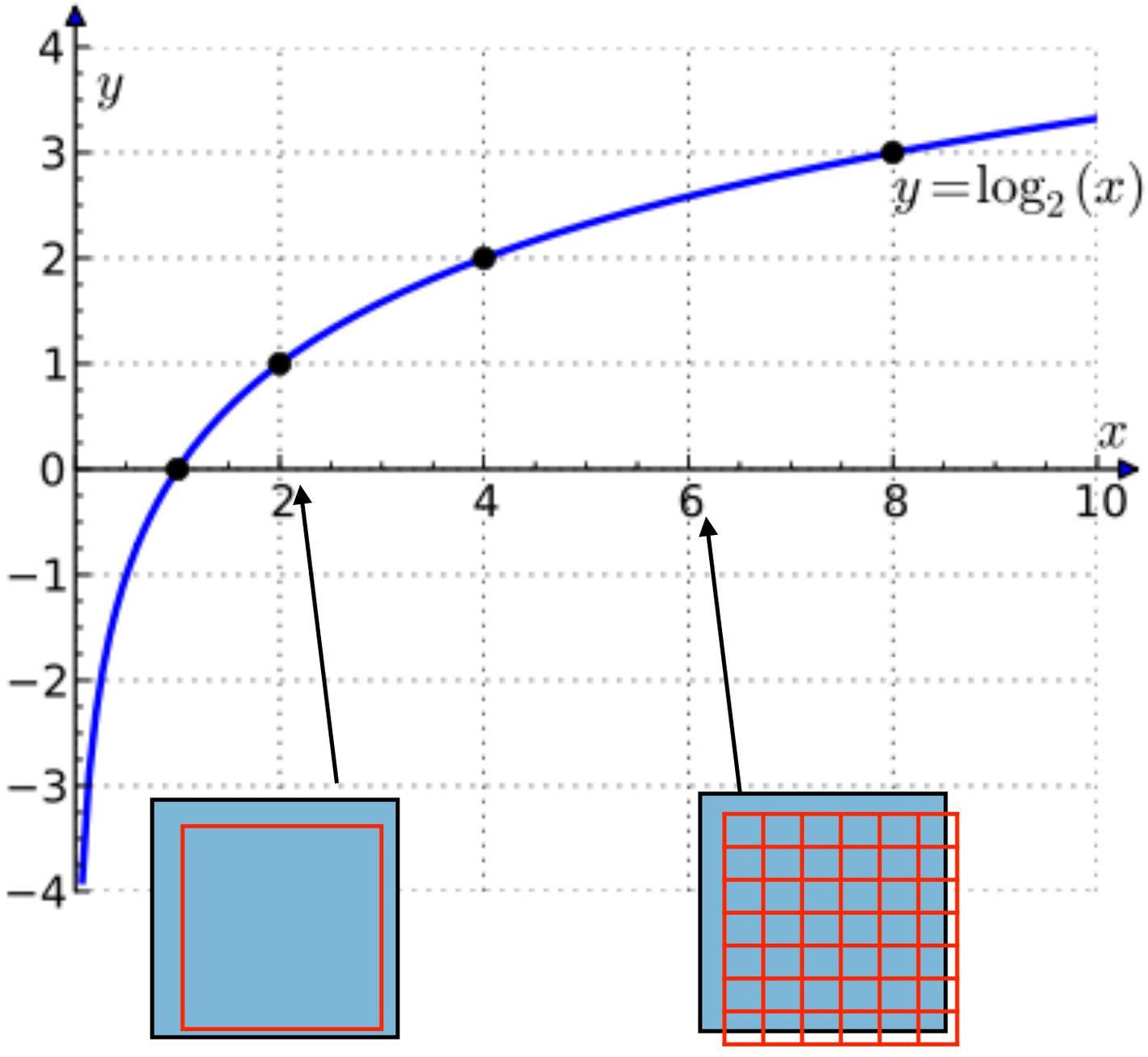
where ∂u/∂x indicates the derivative of *u* with respect to window *x*, and similarly for the other derivatives. For a line, the formula is

$$\rho = \sqrt{\left(\frac{\partial u}{\partial x}\Delta x + \frac{\partial u}{\partial y}\Delta y\right)^2 + \left(\frac{\partial v}{\partial x}\Delta x + \frac{\partial v}{\partial y}\Delta y\right)^2 + \left(\frac{\partial w}{\partial x}\Delta x + \frac{\partial w}{\partial y}\Delta y\right)^2} \Big/ l,$$

where $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$ with (x_1,y_1) and (x_2,y_2) being the segment's window coordinate endpoints and $l = \mathrm{sqrt}(\Delta x^2 + \Delta y^2)$.

Binary Logarithm

ARLAB

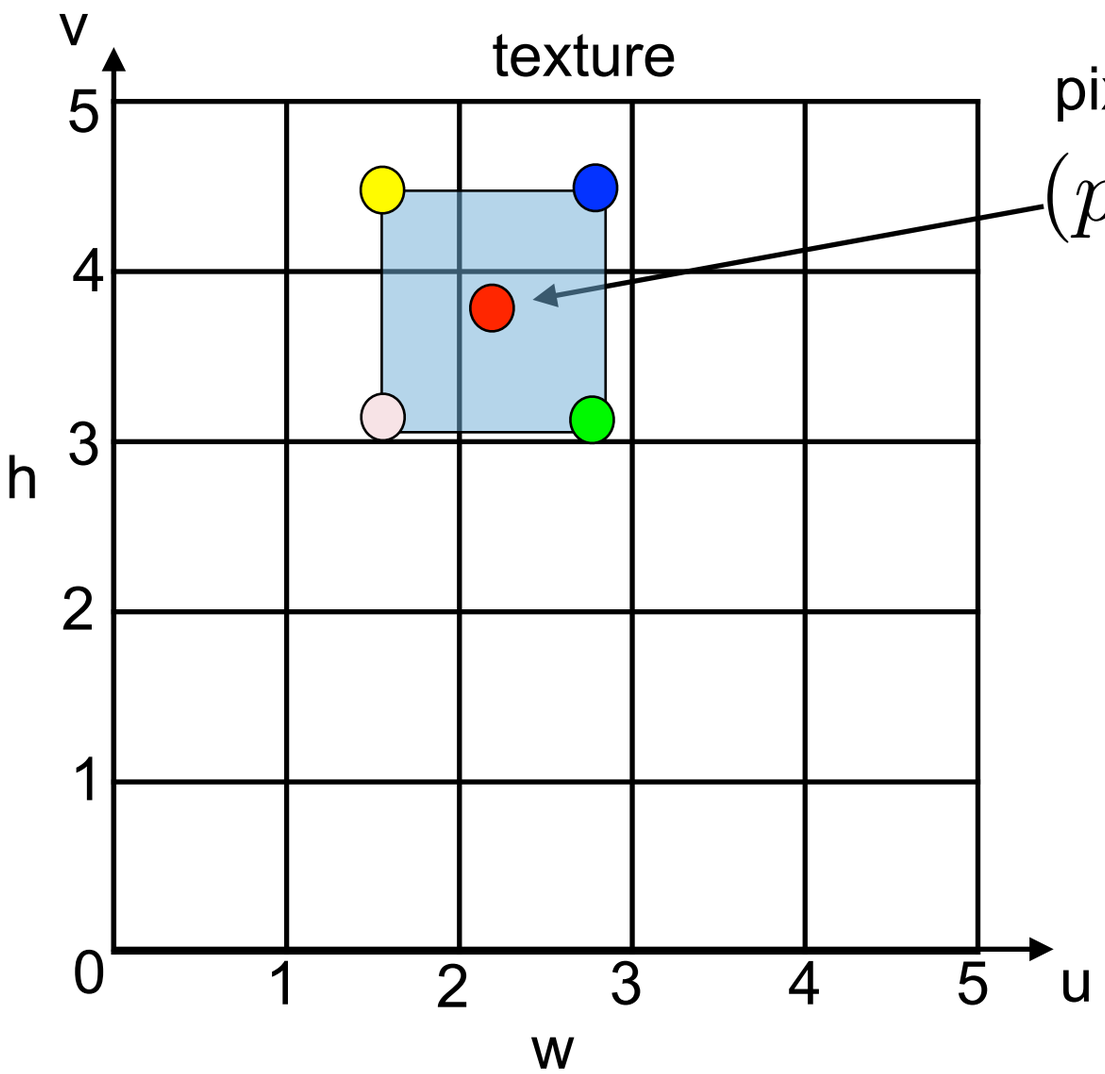


x: how often does my texture texel fit into my current fragment / pixel?

red: texel

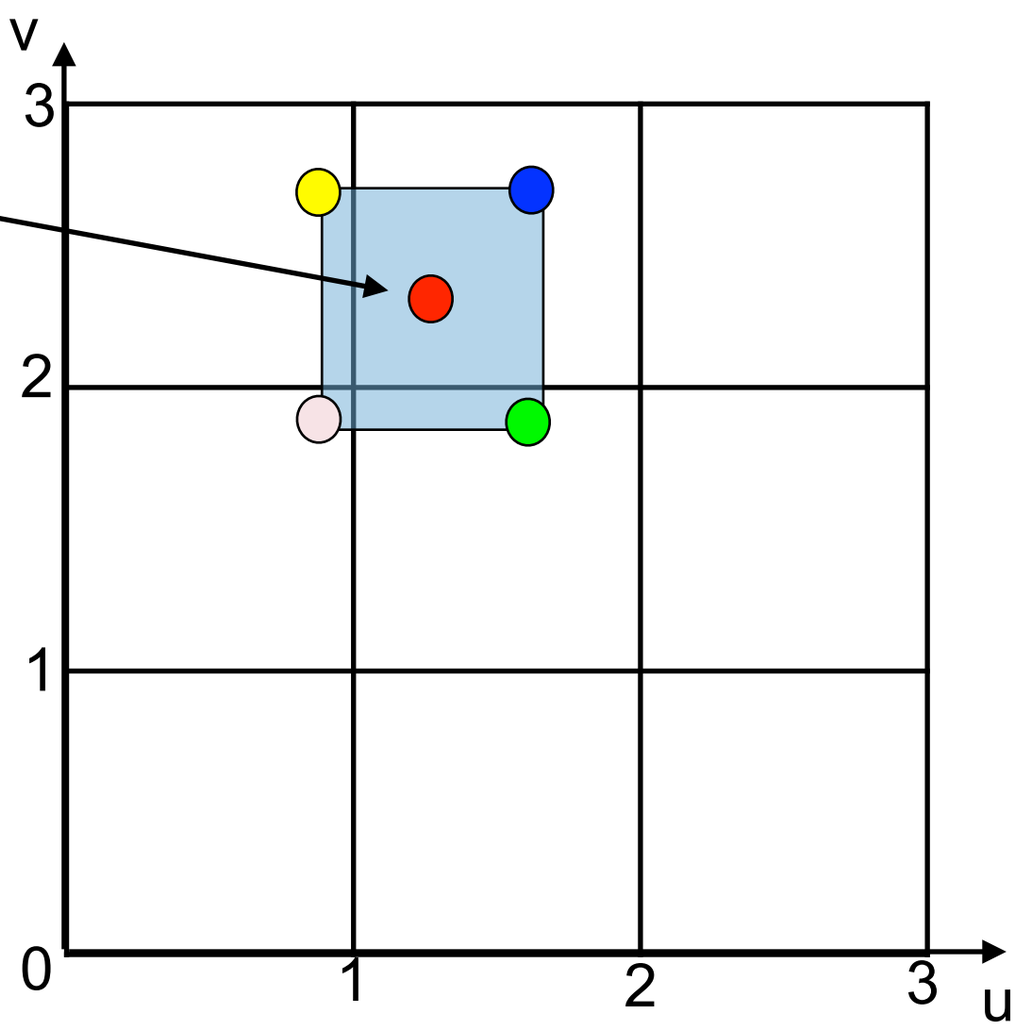
blue: fragment

Trilinear Interpolation (Bicubic Interpolation)

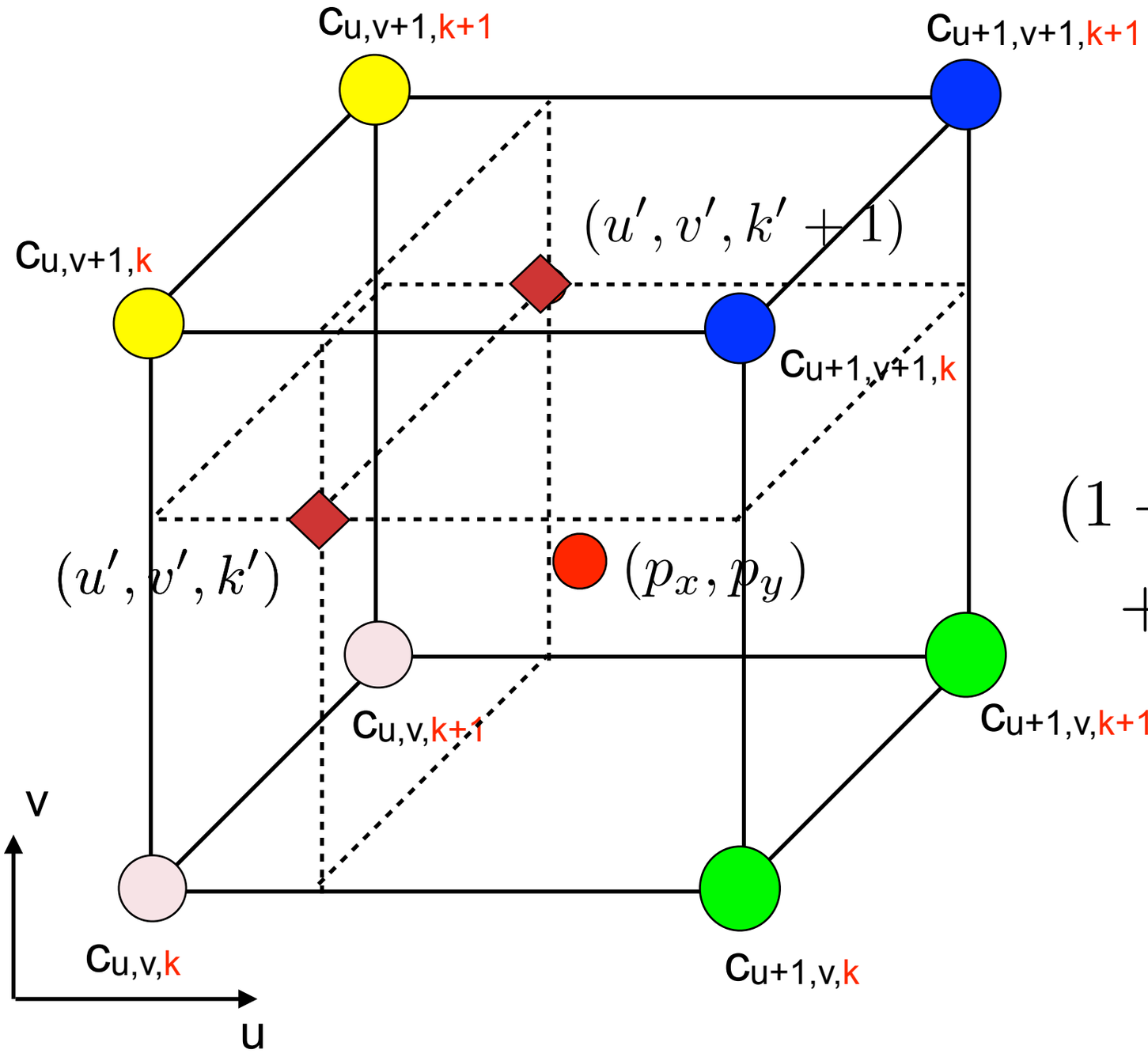


pixel center

(p_x, p_y)



Trilinear Interpolation (Bicubic Interpolation)

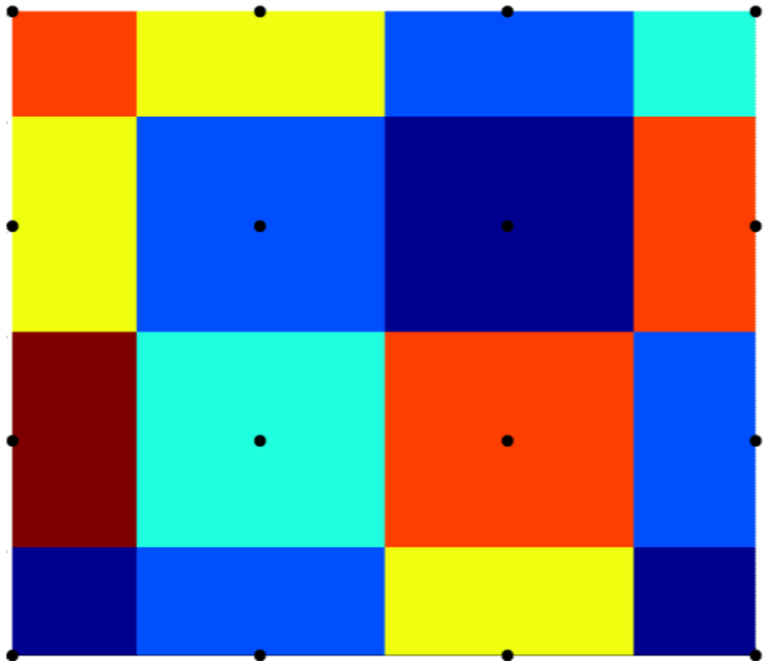


Mathematical representation: we add an additional component k.

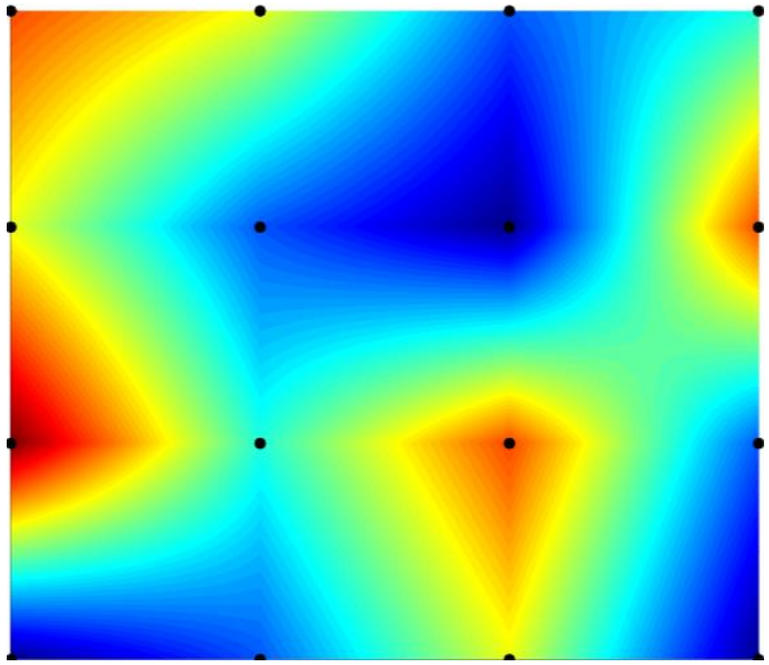
$$c_{x,y} = c(u, v, k) =$$
$$(1-u')(1-v')(1-k')c_{u,v,k}$$
$$+u'(1-v')(1-k)v_{u+1,v,k}$$
$$+\ldots..$$

Example

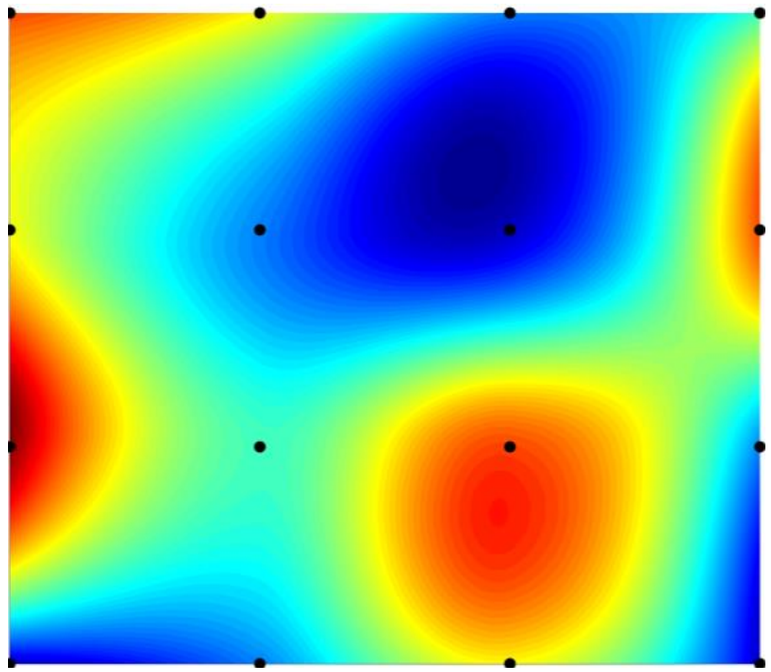
ARLAB



nearest texel

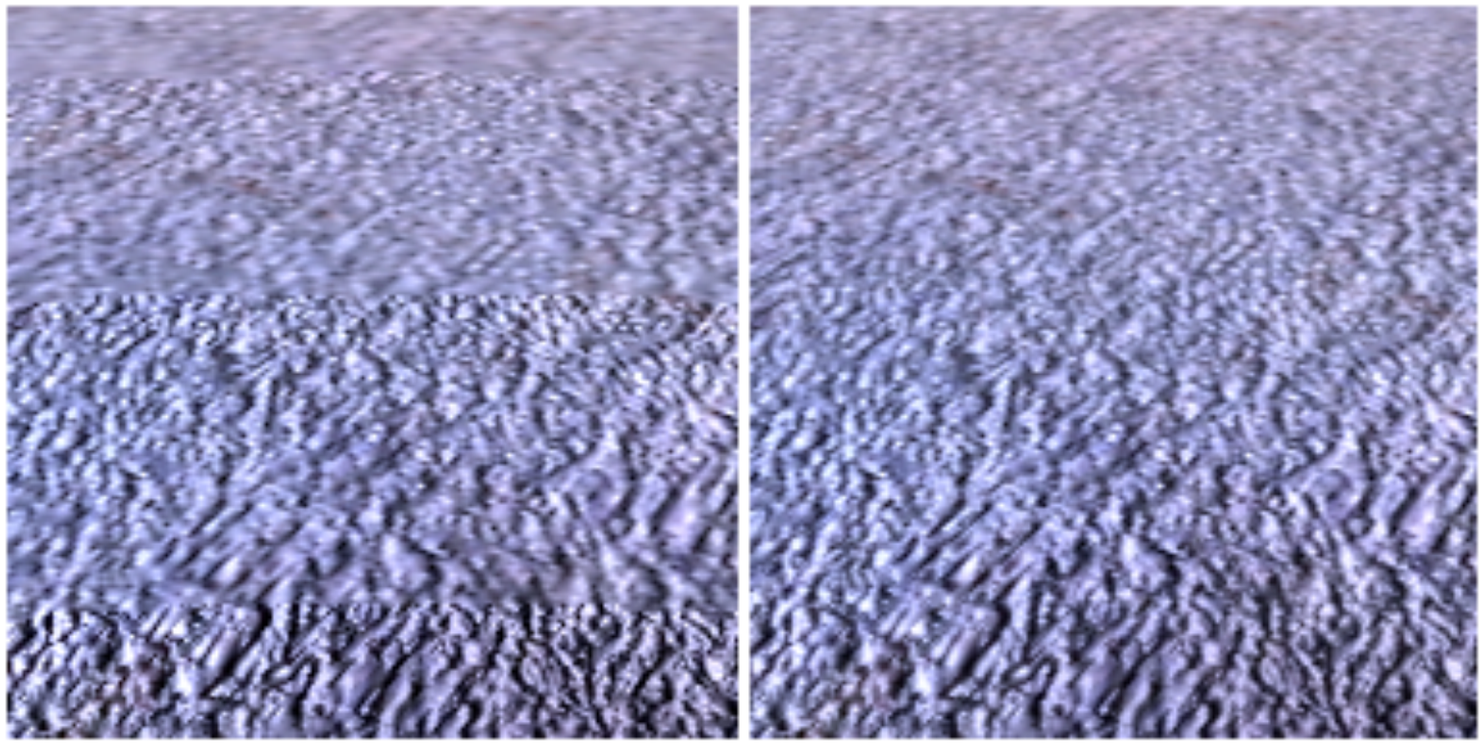


bilinear interpolation (weighted average of 2x2 texels)

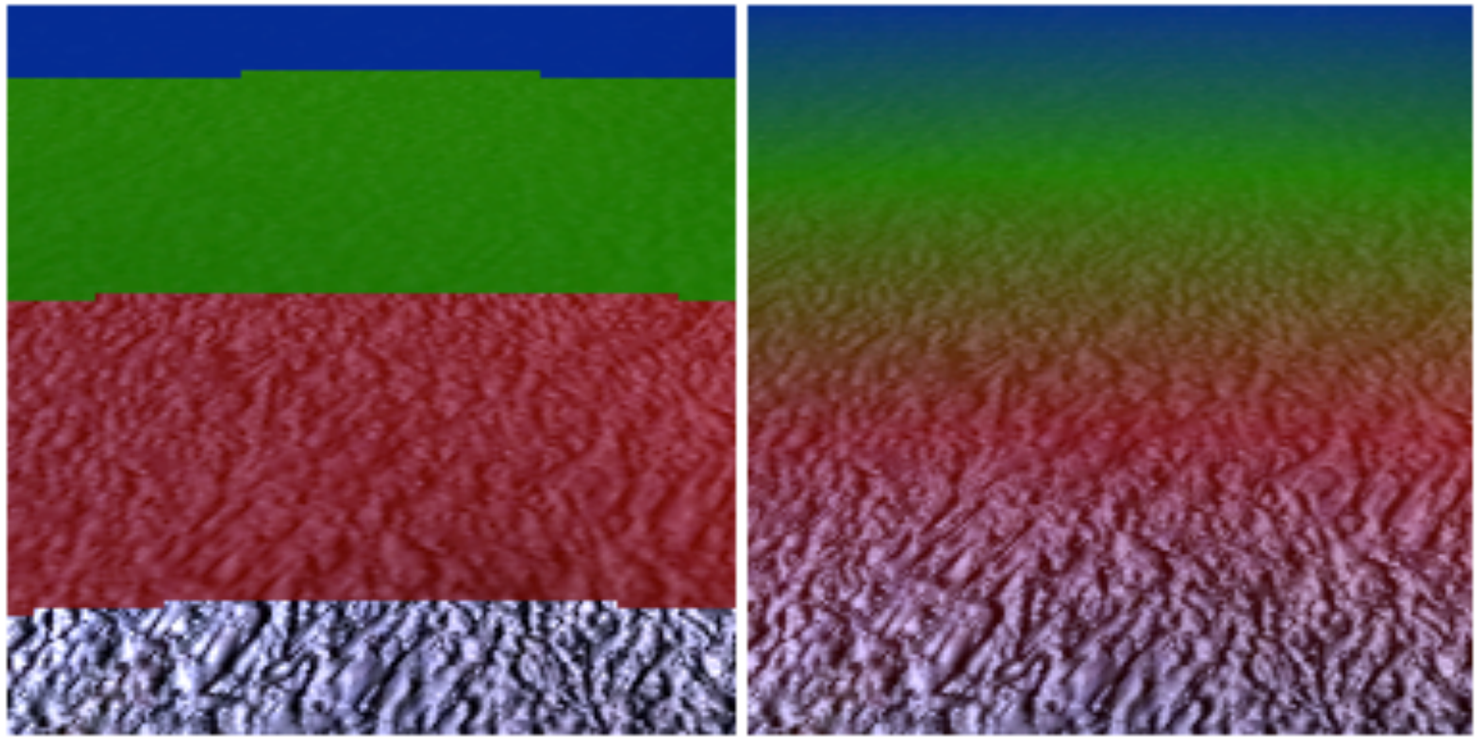


trilinear or bicubic interpolation (weighted average of 2x2 texels)

Bilinear vs. Trilinear



Difference between a Bilinear (left) and a Trilinear (right) filter



Difference between a Bilinear (left) and a Trilinear (right) filter visualized in false colors. Each color represents a different MipMap level

ARLAB

Filter with OpenGL and GLSL

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Fragment Shader Program
Nearest Neighbor

ARLAB

This is the regular texture shader code

```
#version 410 core

uniform sampler2D tex; //this is the texture

in vec2 pass_TexCoord; //this is the texture coord
in vec4 pass_Color;
out vec4 color;

void main(void)
{
    // This function finds the color component for each texture coordinate.
    vec4 tex_color =  texture(tex, pass_TexCoord);

    // This mixes the background color with the texture color.
    // The GLSL shader code replaces the former environment. It is now up to us
    // to figure out how we like to blend a texture with the background color.
    color = pass_Color + tex_color;
}
```

Fragment Shader Program
Bilateral Filter

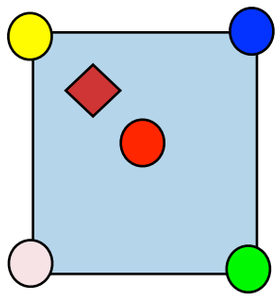
ARLAB

```
#version 410 core

uniform sampler2D tex; //this is the texture

in vec2 pass_TexCoord; //this is the texture coord
in vec4 pass_Color;
out vec4 color;
```

```
void main(void)
{
    // This function finds the color component for each texture coordinate.
    vec4 tex_color =  texture2DBilinear(tex, pass_TexCoord);

    // This mixes the background color with the texture color.
    // The GLSL shader code replaces the former environment. It is now up to us
    // to figure out how we like to blend a texture with the background color.
    color = pass_Color + tex_color;
}
```

Fragment Shader Program
Bilateral Filter

ARLAB



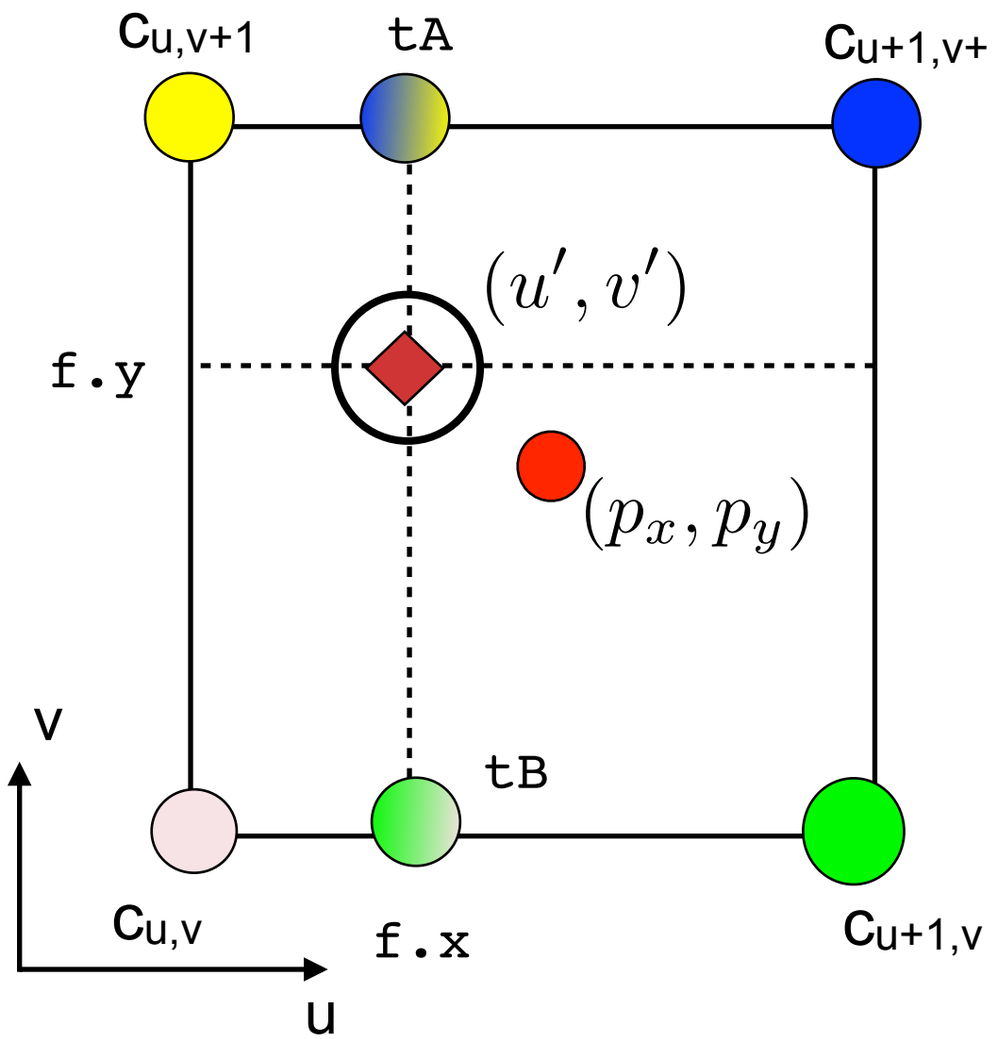
Texture is a square texture

```
const float textureSize = 512.0; //size of the texture
const float texelSize = 1.0 / textureSize; //size of one texel


vec4 texture2DBilinear( sampler2D textureSampler, vec2 uv )
{
   vec4 tl = texture(textureSampler, uv);
   vec4 tr = texture(textureSampler, uv + vec2(texelSize, 0));
   vec4 bl = texture(textureSampler, uv + vec2(0, texelSize));
   vec4 br = texture(textureSampler, uv + vec2(texelSize , texelSize));

   vec2 f = fract( uv.xy * textureSize ); //  the decimal part
   vec4 tA = mix( tl, tr, f.x ); //  interpolate the red dot in the image
   vec4 tB = mix( bl, br, f.x ); //  interpolate the blue dot in the image
   return mix( tA, tB, f.y ); //  interpolate the green dot in the image
}
```

Fragment Shader Program
Bilateral Filter

ARLAB

The mathematical theory differs a little from the computational process. However, the output is equal




```
vec2 f = fract( uv.xy * textureSize );
```

```
vec4 tA = mix( tl, tr, f.x );
```

```
vec4 tB = mix( bl, br, f.x );
```

```
return mix( tA, tB, f.y );
```

fract / mix

ARLAB

compute the fractional part of the argument

```
genType fract( genType x);
```

Parameters:

- x - Specify the value to evaluate.

linearly interpolate between two values

```
genType mix( genType x,genType y, float a);
```

Parameters

- x - Specify the start of the range in which to interpolate.
- y - Specify the end of the range in which to interpolate.
- a - Specify the value to use to interpolate between x and y.

Midmaps in GLSL

Calculate the Midmap level in your fragment shader code.

Two steps:

1. We need to find the correct midmap level
2. Fetch the nearest texture color

This only works with GLSL > 4.0

```
#version 400

uniform sampler2D myTexture;
in vec2 textureCoord; // in normalized units
out vec4 fragColor;

void main()
{
    float mipmapLevel = textureQueryLod(myTexture, textureCoord).x;

    fragColor = textureLod(myTexture, textureCoord, mipmapLevel);
}
```

textureQueryLod

ARLAB

compute the level-of-detail that would be used to sample from a texture

```
vec2 textureQueryLod( gsampler2D sampler, vec2 P);
```

Parameters:

sampler - Specifies the sampler to which the texture whose level-of-detail will be queried is bound.

P - Specifies the texture coordinates at which the level-of-detail will be queried.

textureLod

Perform a texture lookup with explicit level-of-detail

```
gvec4 textureLod( gsampler2D sampler, vec2 P, float lod);
```

Parameters:

- sampler - Specifies the sampler to which the texture from which texels will be retrieved is bound.
- P - Specifies the texture coordinates at which texture will be sampled.
- lod - Specifies the explicit level-of-detail

Midmaps in GLSL

Version 3: we need to calculate the midmap level on our own.

```
#version 330

uniform sampler2D myTexture;
in vec2 textureCoord; // in normalized units
out vec4 fragColor;

float mip_map_level(in vec2 texture_coordinate) // in texel units
{
    vec2  dx_vtc        = dFdx(texture_coordinate);
    vec2  dy_vtc        = dFdy(texture_coordinate);
    float delta_max_sqr = max(dot(dx_vtc, dx_vtc), dot(dy_vtc, dy_vtc));
    float mml = 0.5 * log2(delta_max_sqr);
    return max( 0, mml );
}

void main()
{
    // convert texture coordinates to texel units before calling mip_map_level
    float mipmapLevel = mip_map_level(textureCoord * textureSize(myTexture, 0));

    fragColor = textureLod(myTexture, textureCoord, mipmapLevel);
}
```

dFdx, dFdy

Return the partial derivative of an argument with respect to x or y

```
genType dFdx( genType p);
genType dFdy( genType p);
```

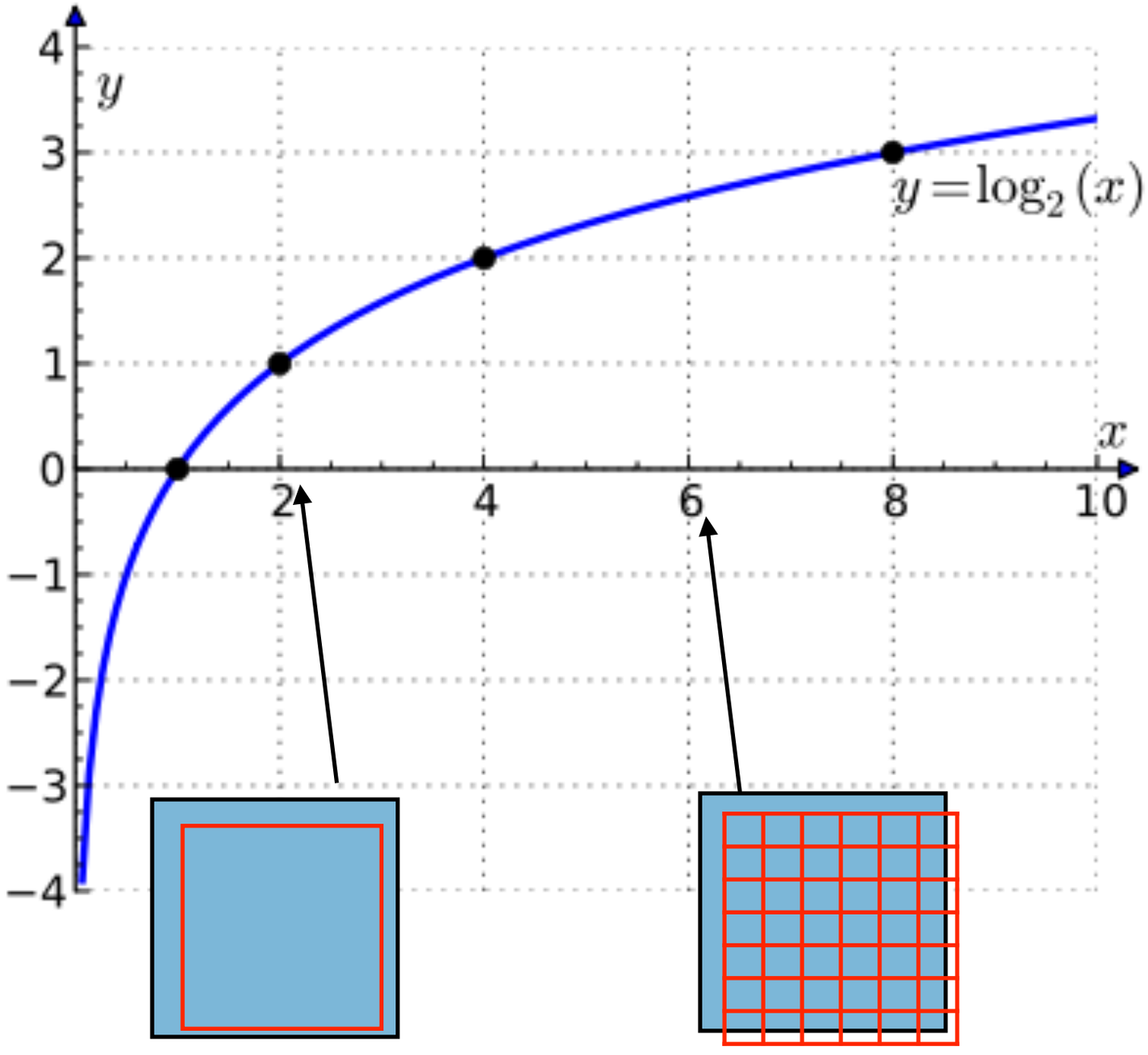
Parameters

- p - Specifies the expression of which to take the partial derivative.

Available only in the fragment shader

$$p_{ux} = \frac{\partial u}{\partial x}, \quad p_{vx} = \frac{\partial v}{\partial x}$$
$$p_{uy} = \frac{\partial u}{\partial y}, \quad p_{vy} = \frac{\partial v}{\partial y}$$

Binary Logarithm



x: how often does my texture texel fit into my current fragment / pixel?

red: texel

blue: fragment

Trilinear / Cubic Filter

ARLAB

```
vec4 TrilinearFilter(sampler2D texture, vec2 texcoord, vec2 texscale)
{
    float fx = fract(texcoord.x);
    float fy = fract(texcoord.y);
    texcoord.x -= fx;
    texcoord.y -= fy;

    vec4 xcubic = cubic(fx);
    vec4 ycubic = cubic(fy);

    vec4 c = vec4(texcoord.x - 0.5, texcoord.x + 1.5, texcoord.y -
             0.5, texcoord.y + 1.5);
    vec4 s = vec4(xcubic.x + xcubic.y, xcubic.z + xcubic.w, ycubic.x +
             ycubic.y, ycubic.z + ycubic.w);
    vec4 offset = c + vec4(xcubic.y, xcubic.w, ycubic.y, ycubic.w) / s;

    vec4 sample0 = texture2D(texture, vec2(offset.x, offset.z) * texscale);
    vec4 sample1 = texture2D(texture, vec2(offset.y, offset.z) * texscale);
    vec4 sample2 = texture2D(texture, vec2(offset.x, offset.w) * texscale);
    vec4 sample3 = texture2D(texture, vec2(offset.y, offset.w) * texscale);

    float sx = s.x / (s.x + s.y);
    float sy = s.z / (s.z + s.w);

    return mix(
        mix(sample3, sample2, sx),
        mix(sample1, sample0, sx), sy);
}
```

Building Midmaps

ARLAB

```
// Change the parameters of your texture units.
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,GL_NEAREST );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,GL_LINEAR );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,GL_REPEAT );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,GL_REPEAT );

// Create a texture and load it to your graphics hardware. This
texture is automatically associated with texture 0 and the texture
variable "texture" / the active texture.
if(channels == 3)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_BGR,
    GL_UNSIGNED_BYTE, data);
else if(channels == 4)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_BGRA,
    GL_UNSIGNED_BYTE, data);
```

Midmap

Building Midmaps

ARLAB

```
// Change the parameters of your texture units.
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,GL_NEAREST );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,GL_LINEAR );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,GL_
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,GL_

// Create a texture and load it to your graphics hardware. This
texture is automatically associated with texture 0 and the texture
variable "texture" / the active texture.
if(channels == 3){
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_BGR,
    GL_UNSIGNED_BYTE, data_0);
    glTexImage2D(GL_TEXTURE_2D, 1, GL_RGB, width, height, 0, GL_BGR,
    GL_UNSIGNED_BYTE, data_1);
}
else if(channels == 4)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_BGRA,
    GL_UNSIGNED_BYTE, data);
```

You do not need these filters anymore

Midmap level: load images with different resolutions and add your midmap levels manually

Building Midmaps

```
//*************************************************************
//Create a midmap texture pyramid and load it to the graphics
hardware.
gluBuild2DMipmaps( GL_TEXTURE_2D, 3, width, height,GL_RGB,
GL_UNSIGNED_BYTE, data );
```

Load a big image

2048 x 2048

1024 x 1024

512 x 512

256 x 256

128 x 128

64 x 64

1x1

(11) (10) (9) (8) (7) (6) (0)

The highest level is

$$\log_2(\max(width, height))$$

It takes half of the size for all other textures until the size is 1x1

MidMaps in OpenGL

gluBuild2DMipmaps builds a series of prefiltered two-dimensional texture maps of decreasing resolutions called a mipmap

GLint gluBuild2DMipmaps(GLenum target, GLint components, GLsiz
GLsizei height, GLenum format, GLenum type, const void * data);

This functions replaces glTexture2D

Parameters:

- target: Specifies the target texture. Must be GL_TEXTURE_2D.
- internalFormat: Specifies the number of color components in the texture. Must be 1, 2, 3, or 4.
- width, height: Specifies the width and height, respectively, of the texture image.
- format: Specifies the format of the pixel data. Must be one of: GL_COLOR_INDEX, GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA, GL_RGB, GL_RGBA, GL_LUMINANCE, and GL_LUMINANCE_ALPHA.
- type: Specifies the data type for data. Must be one of: GL_UNSIGNED_BYTE, GL_BYTE, GL_BITMAP, GL_UNSIGNED_SHORT, GL_SHORT, GL_UNSIGNED_INT, GL_INT, or GL_FLOAT.
- data: Specifies a pointer to the image data in memory.

Thank you!

Questions

Rafael Radkowski, Ph.D.
Iowa State University
Virtual Reality Applications Center
1620 Howe Hall
Ames, Iowa 5001, USA

+1 515.294.5580
+1 515.294.5530(fax)

rafael@iastate.edu

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY