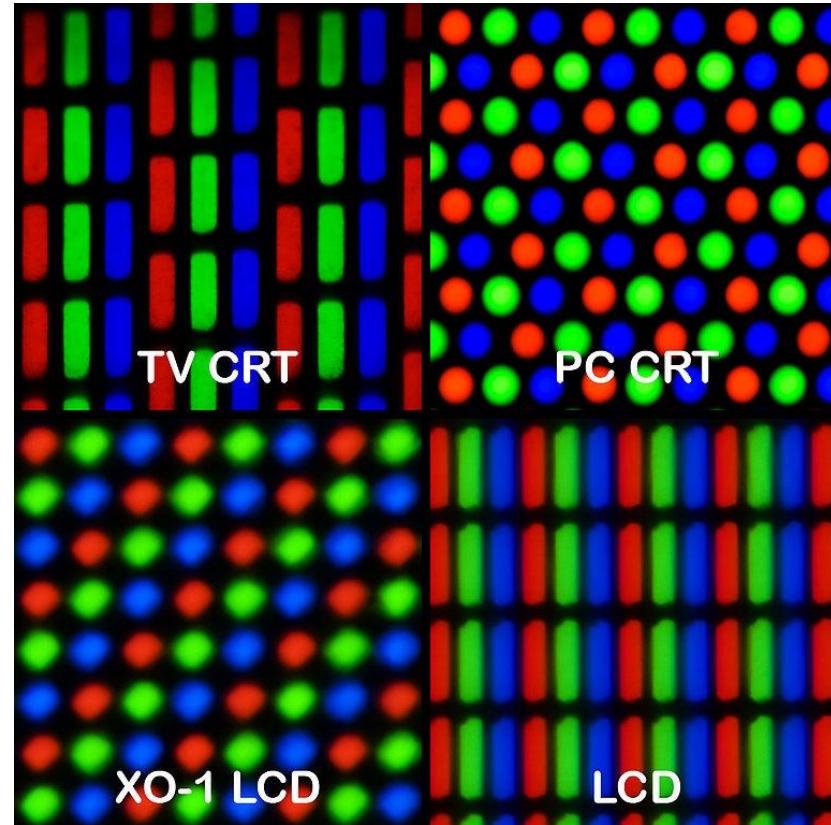
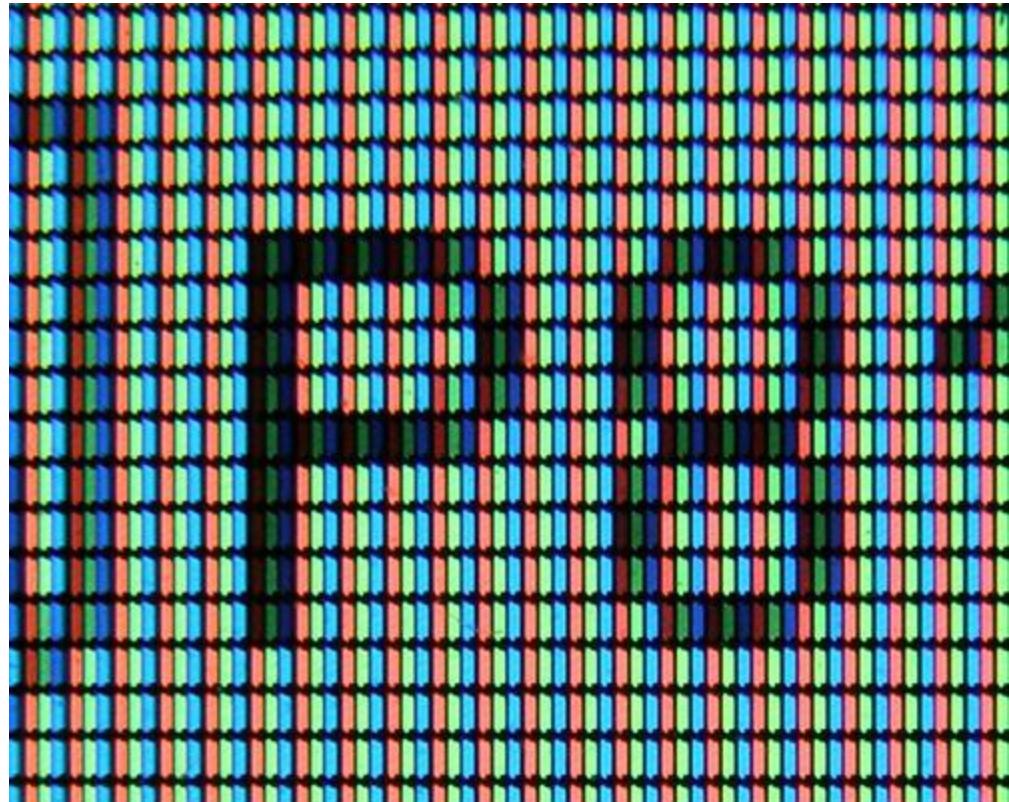

Computer Graphics labs

Lab 4 - Textures





XO-1 LCD

LCD

TV CRT

PC CRT



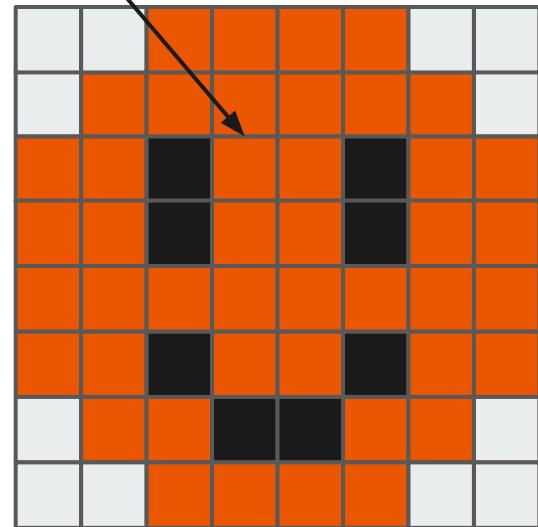
Textures

It's a 1D, 2D or 3D array of Pixels.

Each pixel holds a vector.

Usually the vector represents a color but
not always (it is up to you to interpret it).

(235, 86, 0, 255)



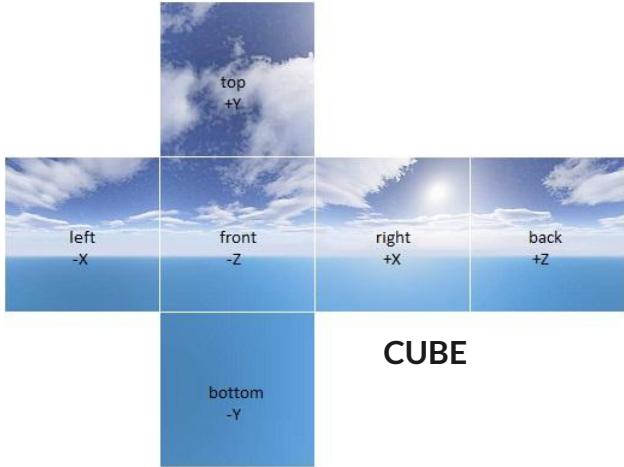
TYPES OF TEXTURES



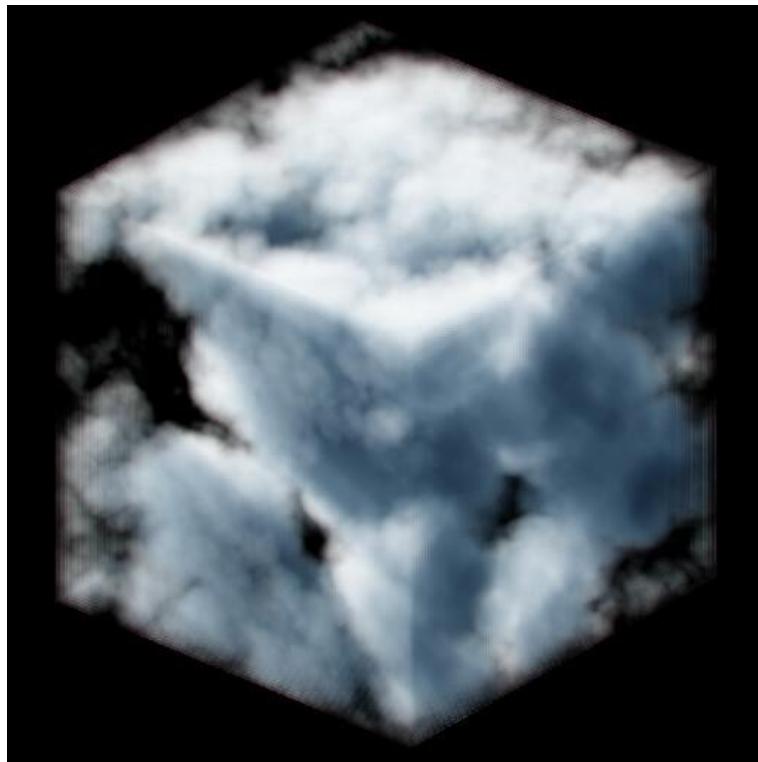
1D

H1	H2	H3	H4	H5	H6	H7	H8
G1	G2	G3	G4	G5	G6	G7	G8
F1	F2	F3	F4	F5	F6	F7	F8
E1	E2	E3	E4	E5	E6	E7	E8
D1	D2	D3	D4	D5	D6	D7	D8
C1	C2	C3	C4	C5	C6	C7	C8
B1	B2	B3	B4	B5	B6	B7	B8
A1	A2	A3	A4	A5	A6	A7	A8

2D



ARRAY



3D

AND OTHERS

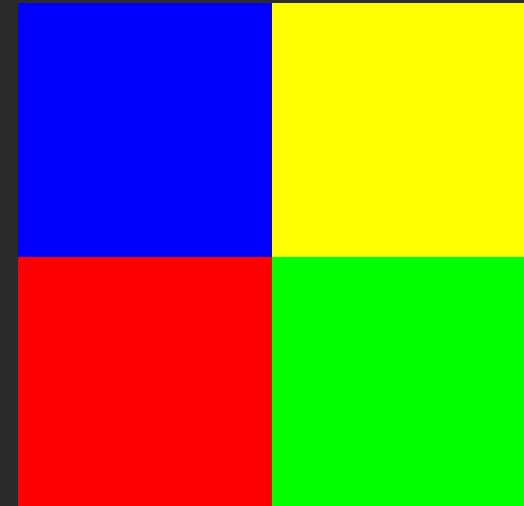
```
uint8_t pixel_data[] = {  
    255, 0, 0, 255,  
    0, 255, 0, 255,  
    0, 0, 255, 255,  
    255, 255, 0, 255,  
};
```

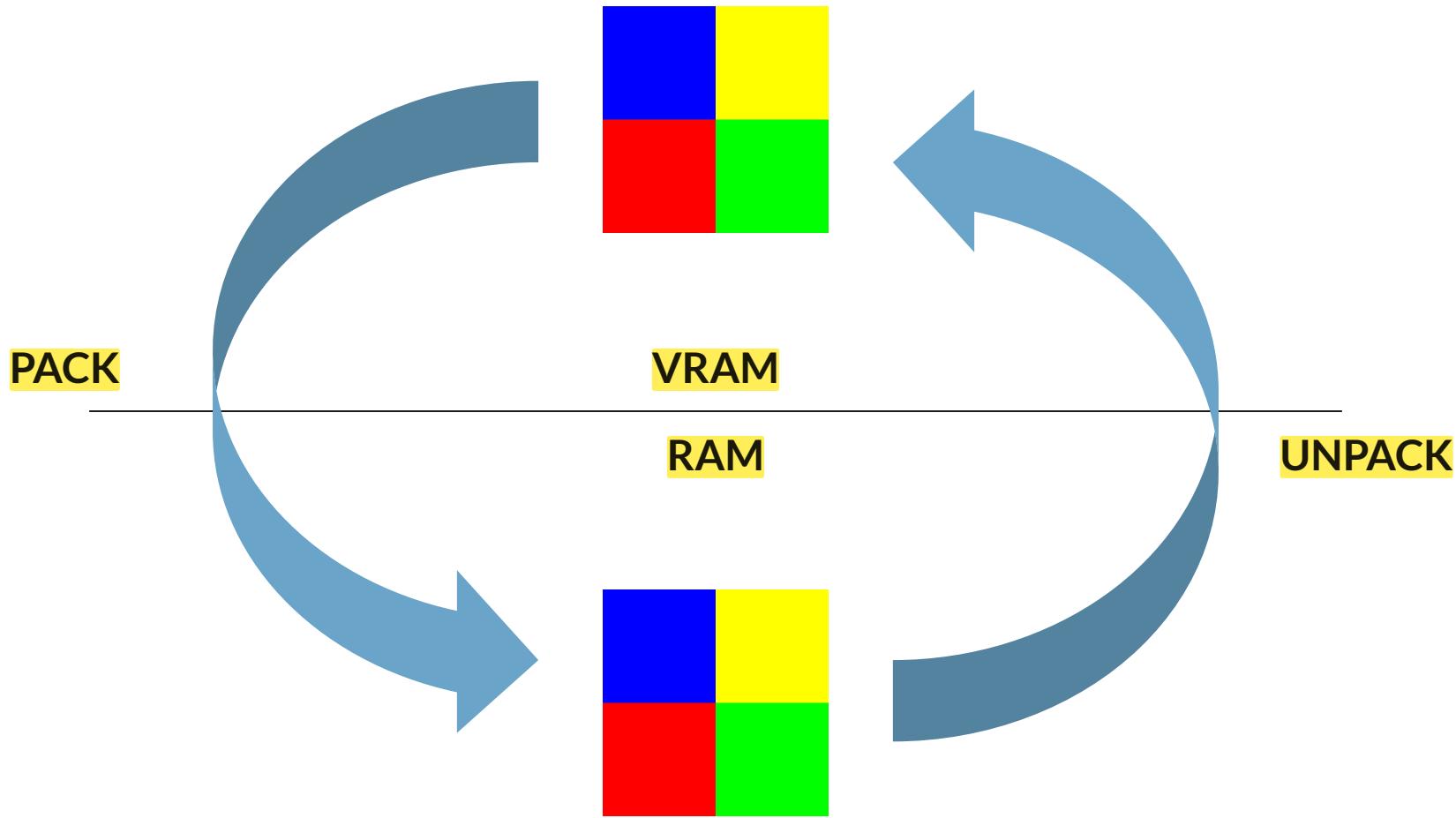
```
glGenTextures(1, &texture);  
 glBindTexture(GL_TEXTURE_2D, texture);
```

```
glPixelStorei(GL_UNPACK_ALIGNMENT, 4);
```

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, 2, 2, 0, GL_RGBA,  
GL_UNSIGNED_BYTE, pixel_data);
```

```
glGenerateMipmap(GL_TEXTURE_2D);
```





```
glPixelStorei(GL_UNPACK_ALIGNMENT, 4);
```

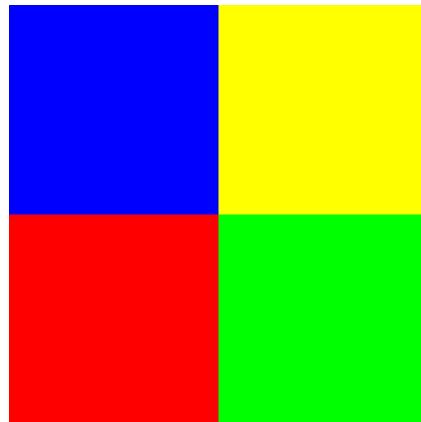
This tells OpenGL that each row starts at an offset that is multiple of 4 bytes.

The possible values are 1, 2, 4 and 8.

1 should work for every texture as long as the data is tightly packed.

```
glGenerateMipmap(GL_TEXTURE_2D);
```

A Mip level is a smaller version of the texture.



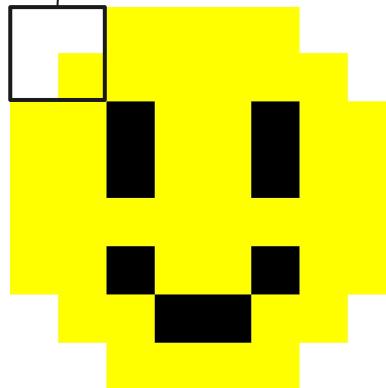
Level 0 (2x2)



Level 1 (1x1)

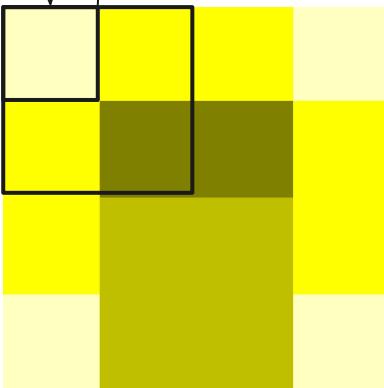


AVERAGE



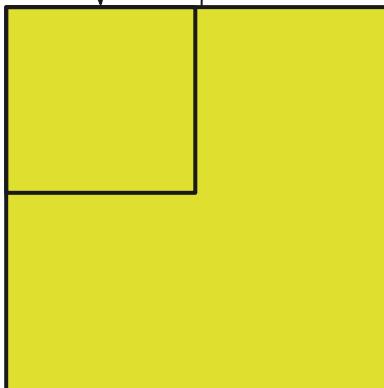
Level 0 (8x8)

AVERAGE



Level 1 (4x4)

AVERAGE



Level 2 (2x2)

Level 3 (1x1)



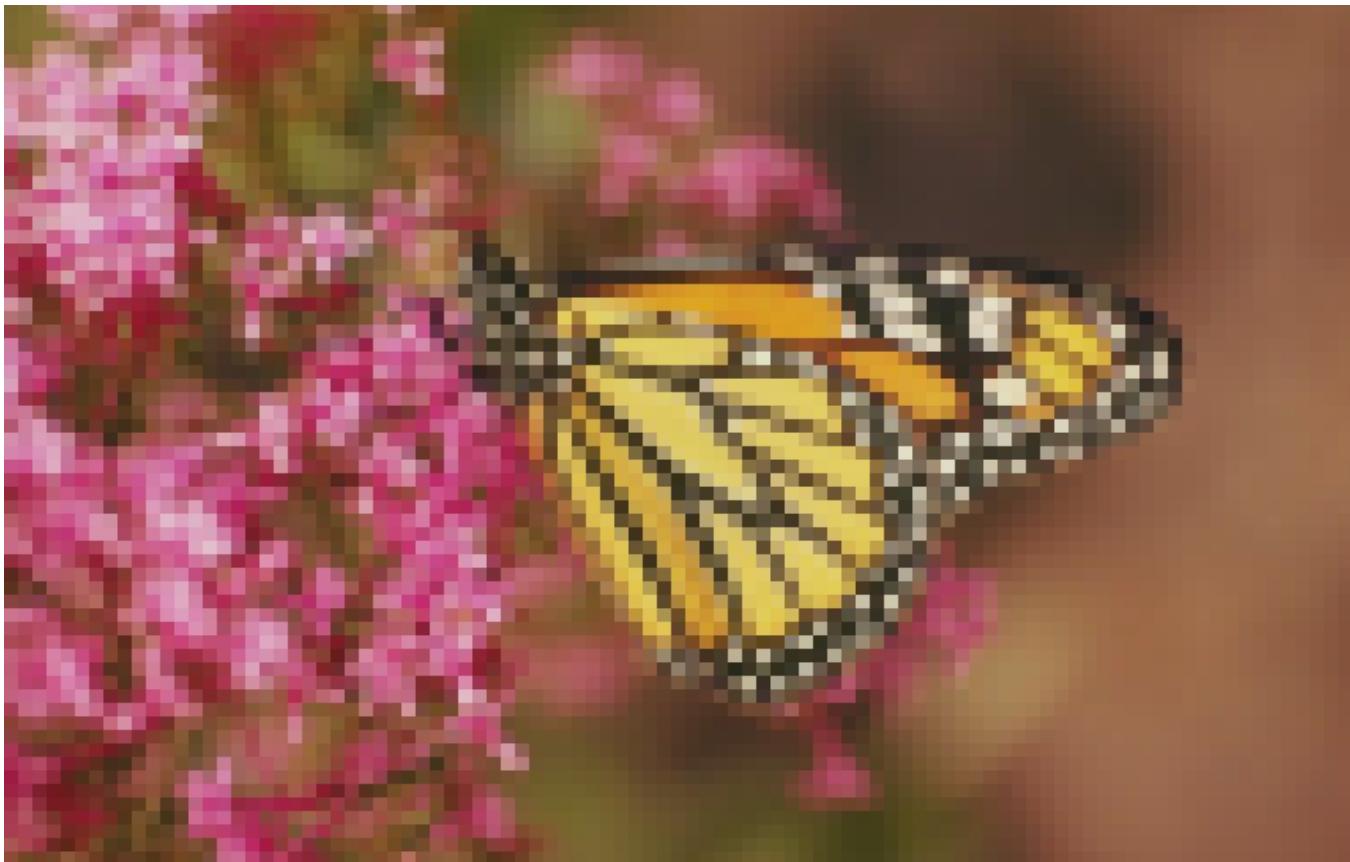
Level 0 (768x512)



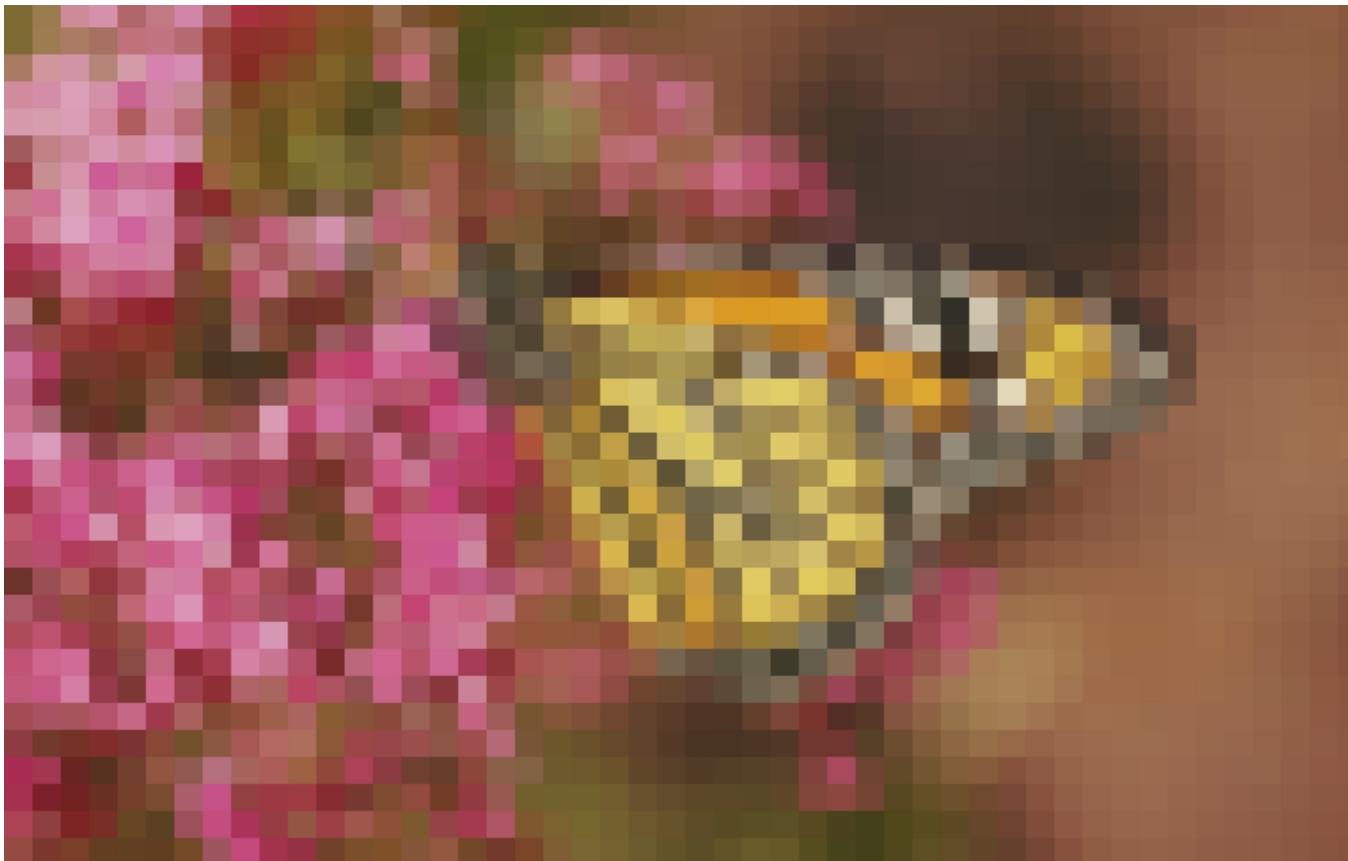
Level 1 (384x256)



Level 2 (192x128)



Level 3 (96x64)



Level 4 (48x32)



Level 5 (24x16)



Level 6 (12x8)



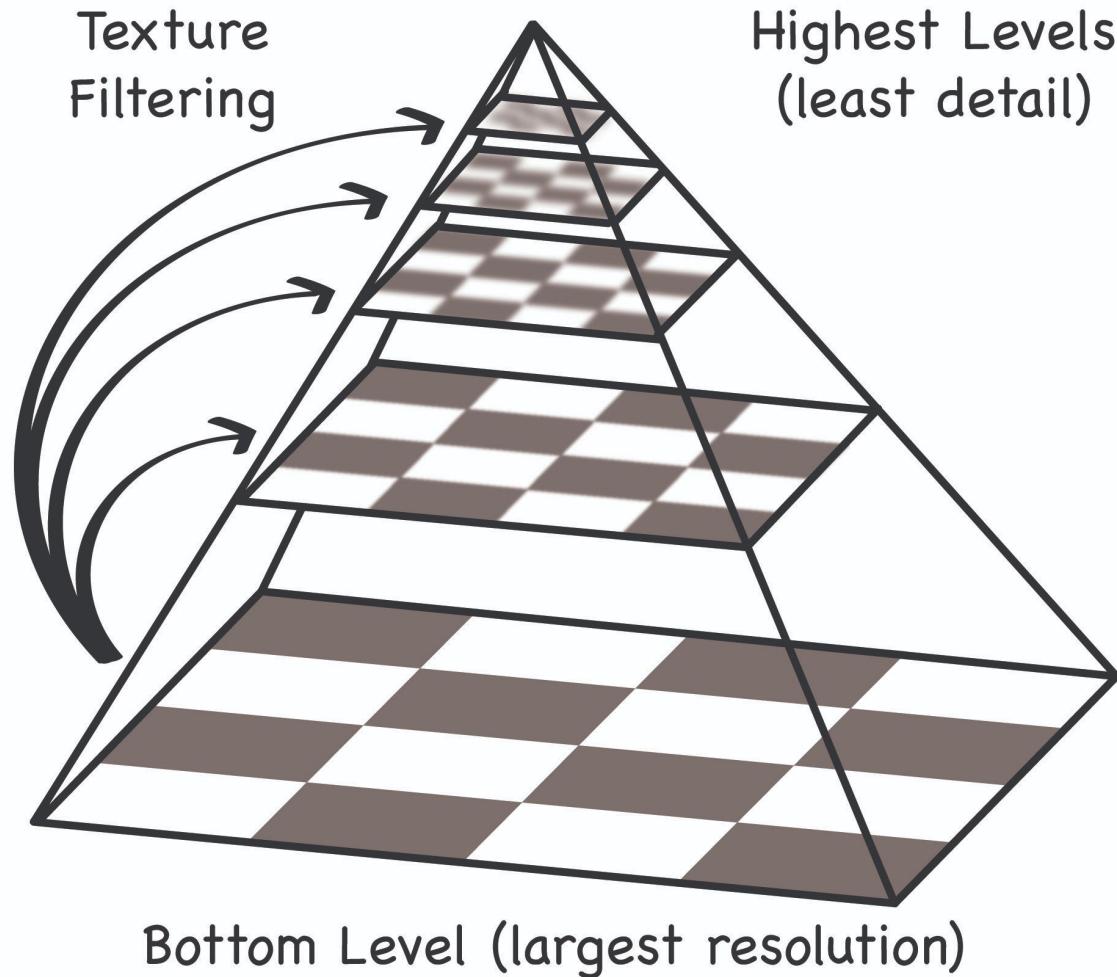
Level 7 (6x4)



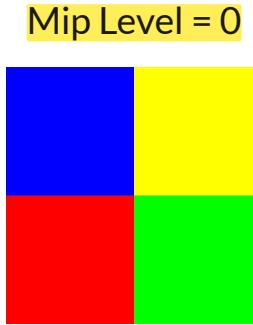
Level 8 (3x2)



Level 9 (1x1)



```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, 2, 2, 0, GL_RGBA, GL_UNSIGNED_BYTE, data);
```



Internal Format

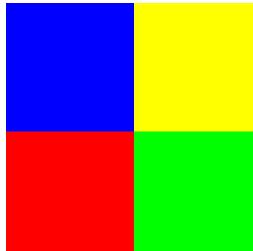
Width & Height

Border
(must be 0)

Format & Data type of Data pointer

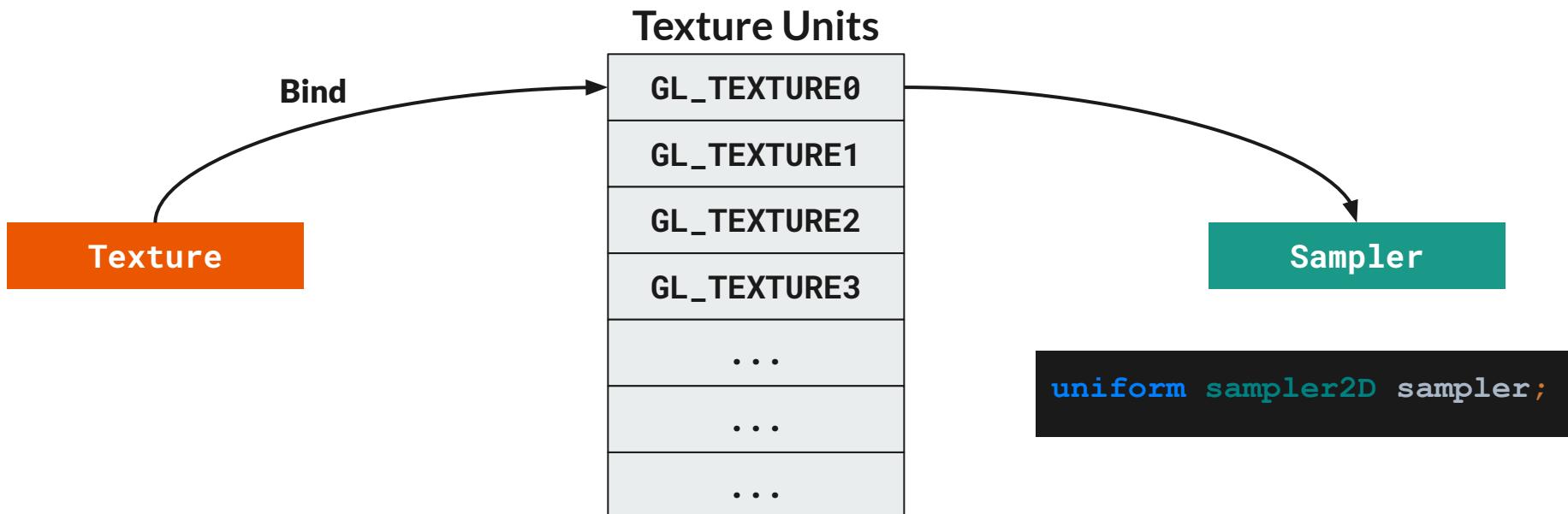
VRAM

RAM



```
uint8_t data[] = {  
    255, 0, 0, 255,  
    0, 255, 0, 255,  
    0, 0, 255, 255,  
    255, 255, 0, 255,  
};
```

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture);
GLuint sampler_loc = glGetUniformLocation(program, "sampler");
 glUniform1i(sampler_loc, 0);
```



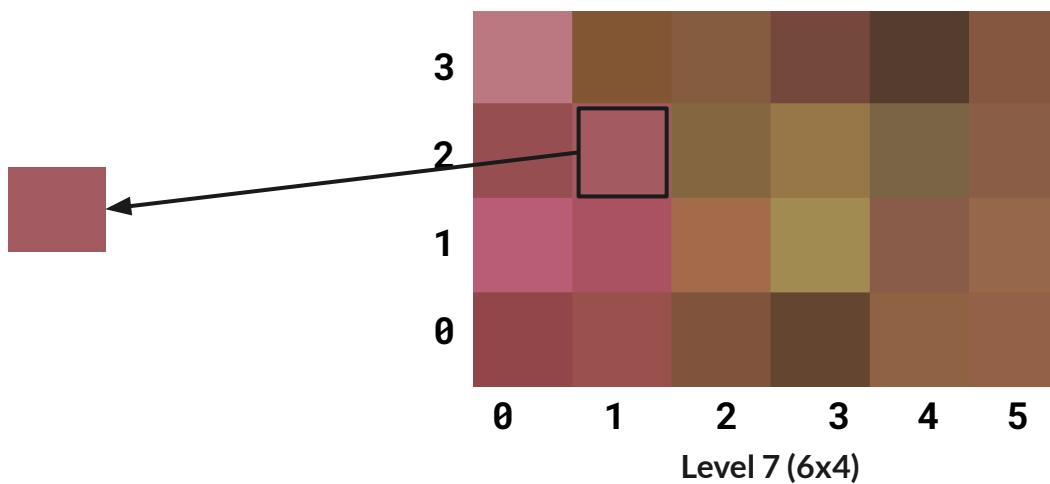
```
vec4 frag_color = texelFetch(sampler, pixel_coord, lod);
```

Texture Sampler

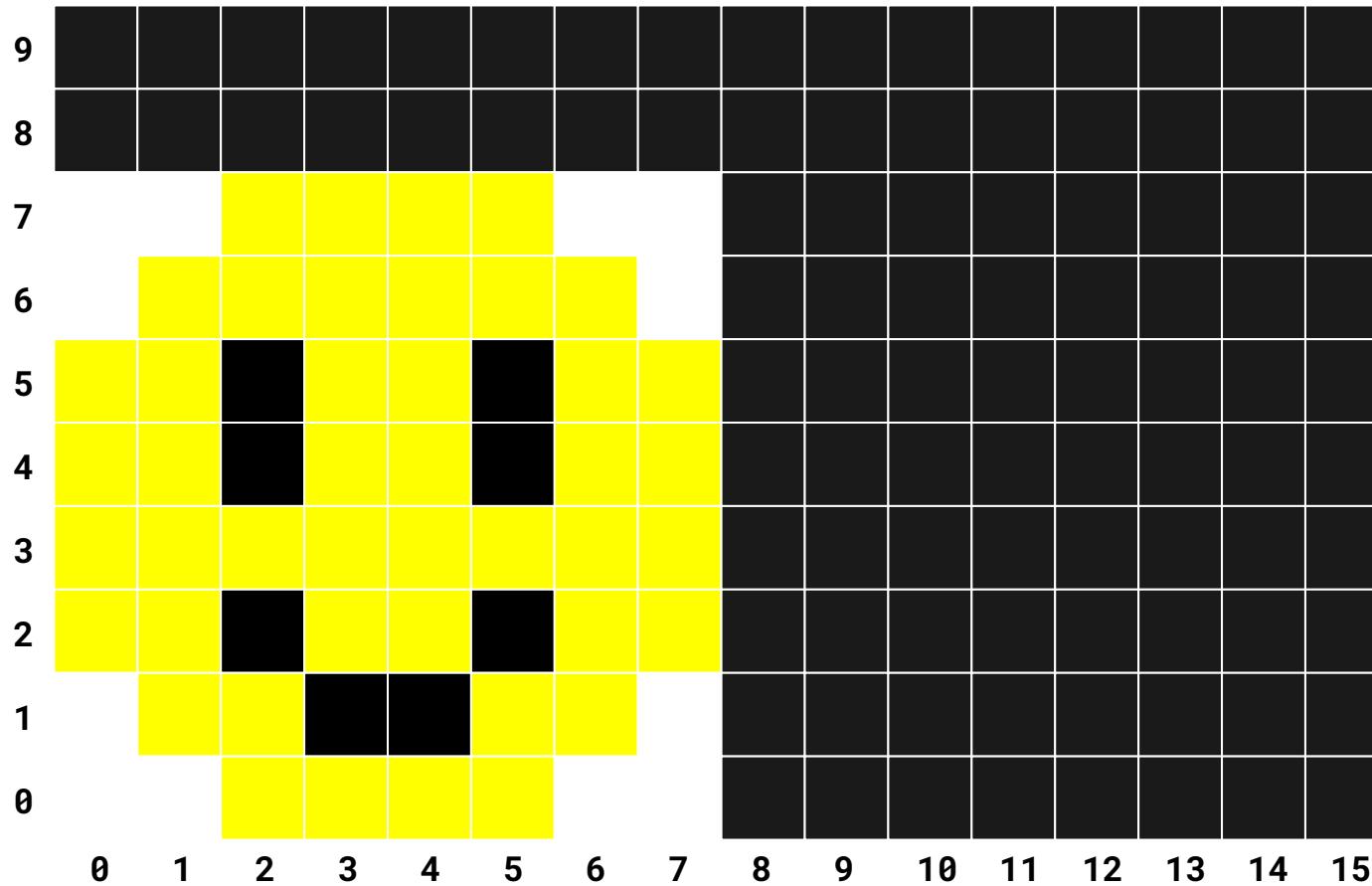
2D Index of Pixel

Mip level

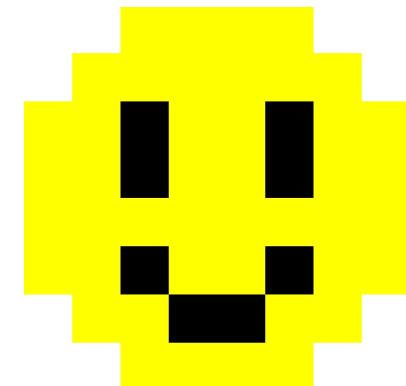
```
vec4 frag_color = texelFetch(sampler, ivec2(1,2), 7);
```



WINDOW

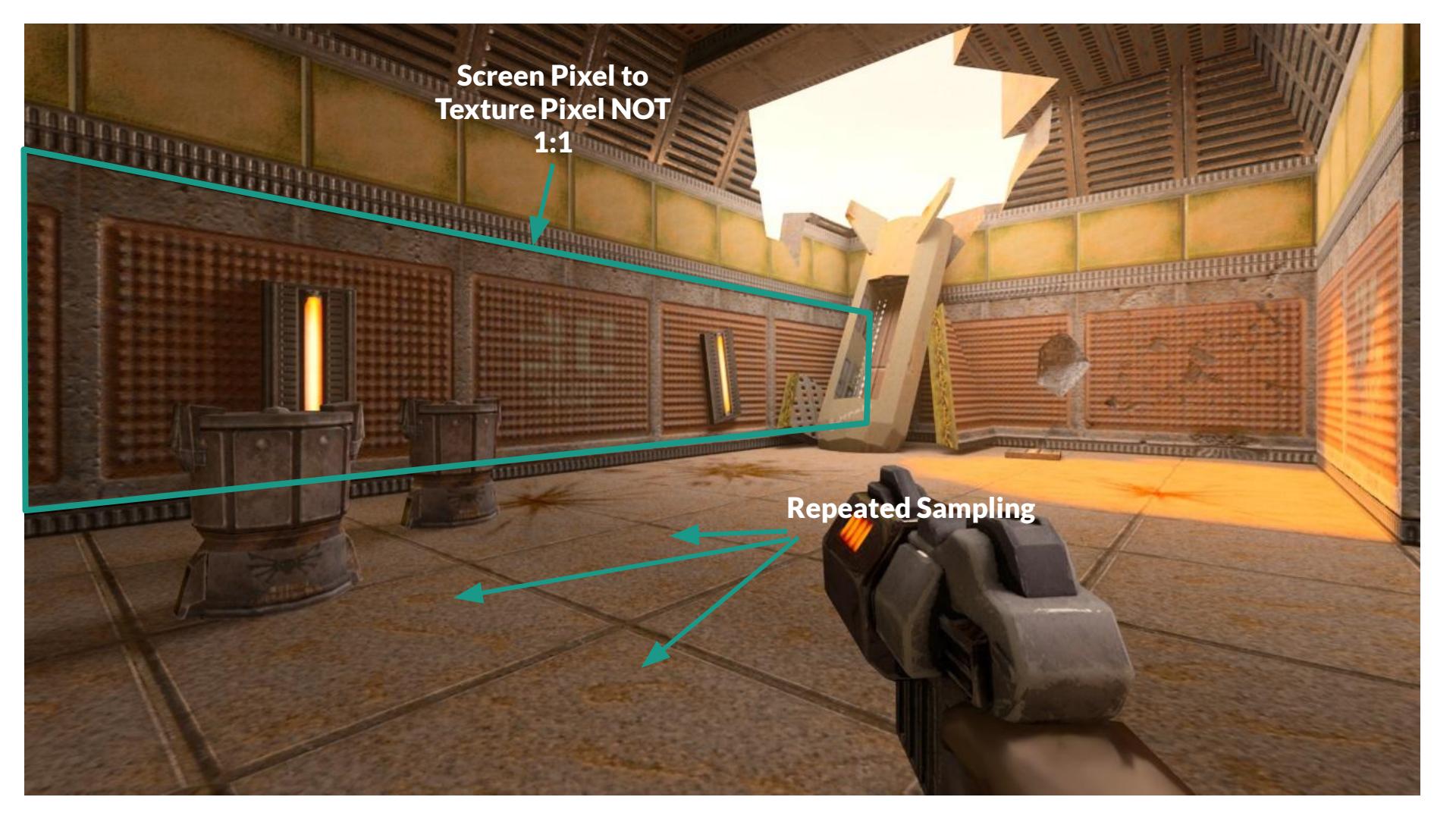


```
texelFetch(sampler,  
gl_FragCoord.xy,  
0);
```



Sampler
(Texture 2D)

**1:1 Screen Pixel to
Texture Pixel**



**Screen Pixel to
Texture Pixel NOT**

1:1

Repeated Sampling

```
vec4 frag_color = texture(sampler, texture_coord);
```

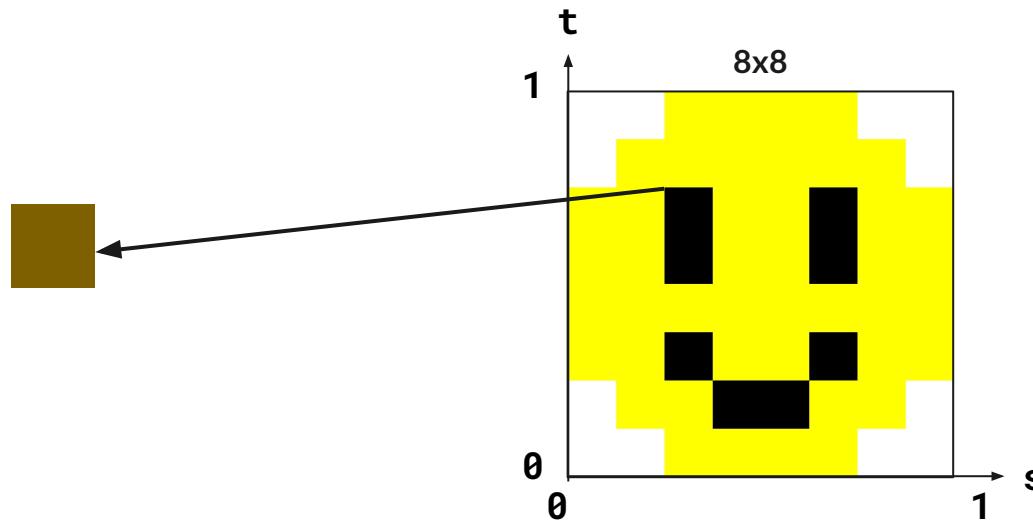


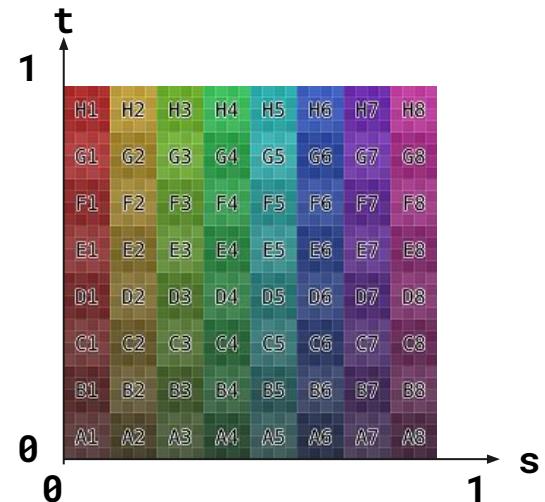
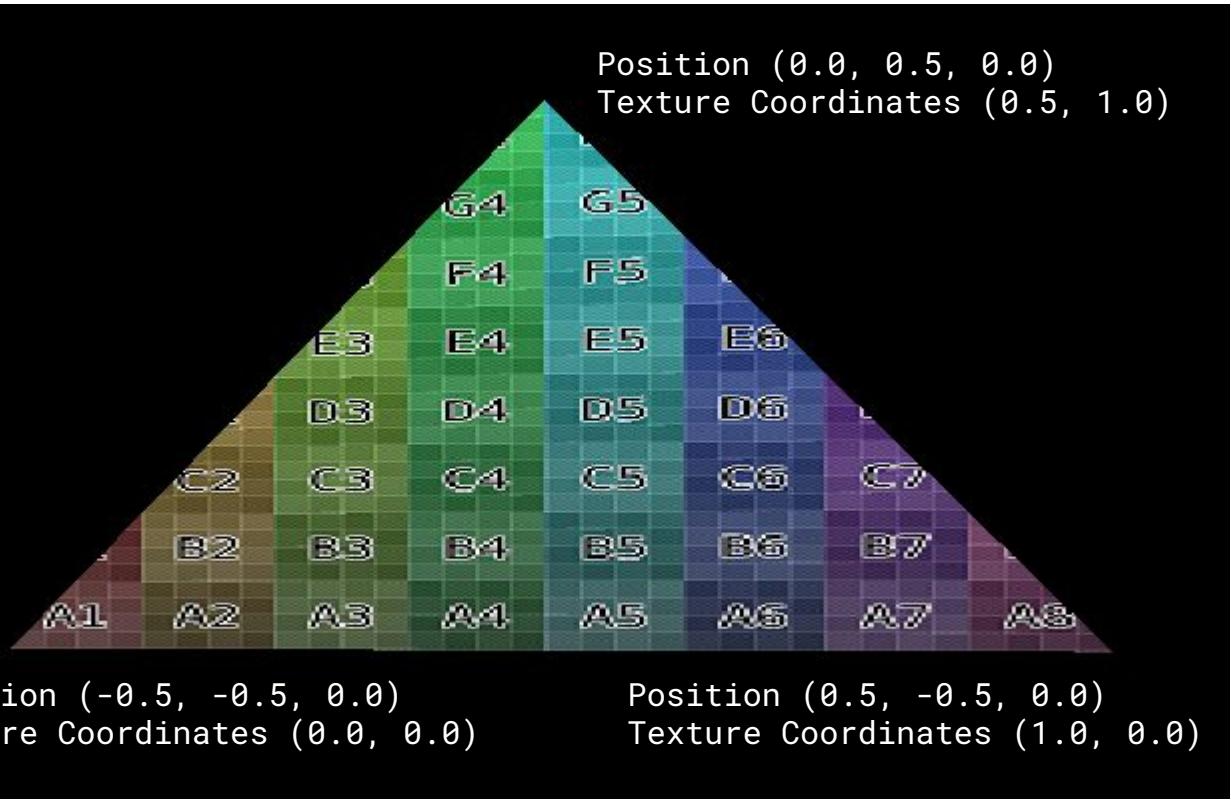
Texture Sampler



Texture Coordinate

```
vec4 frag_color = texture(sampler, vec2(0.25,0.75));
```

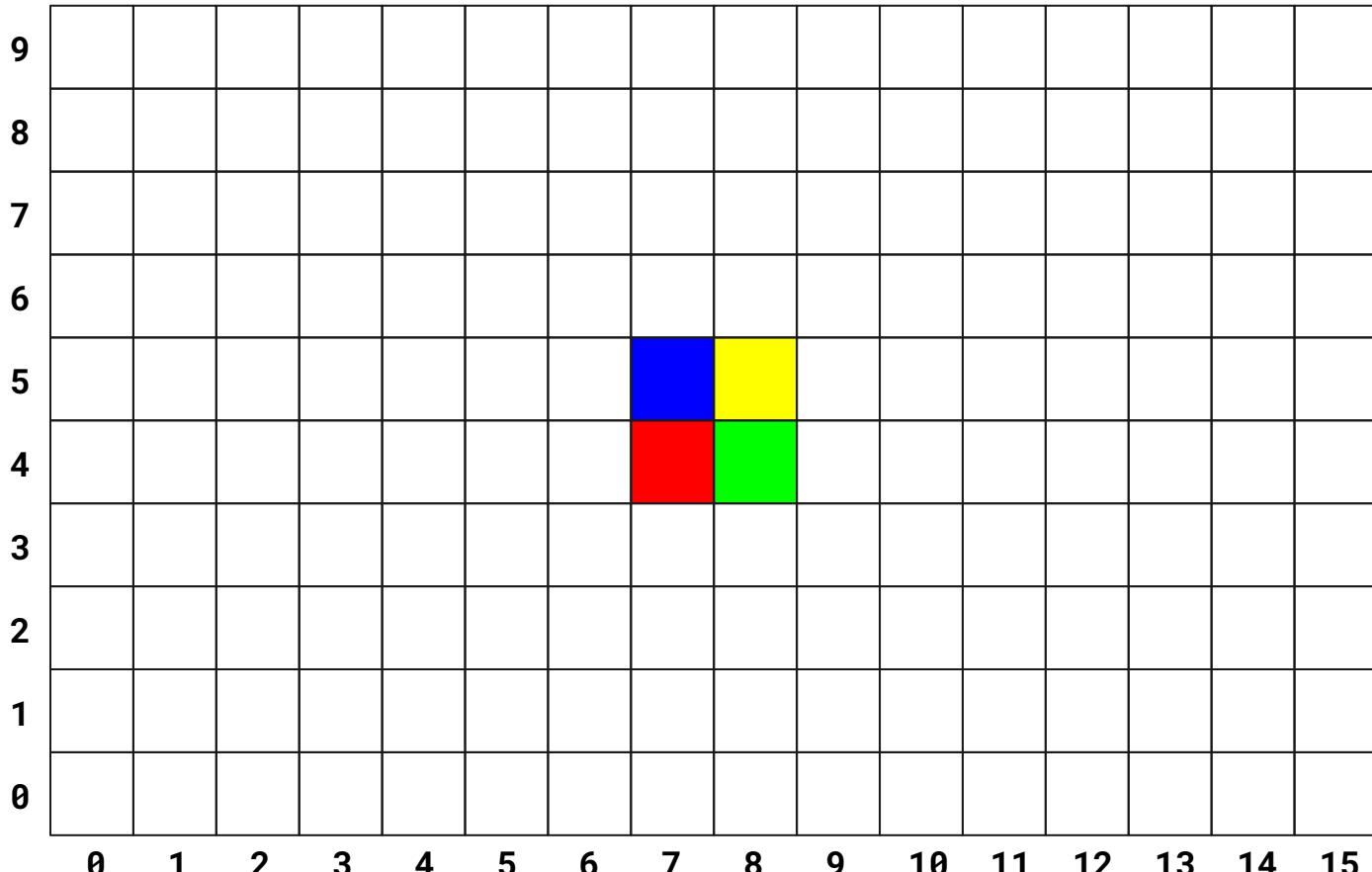




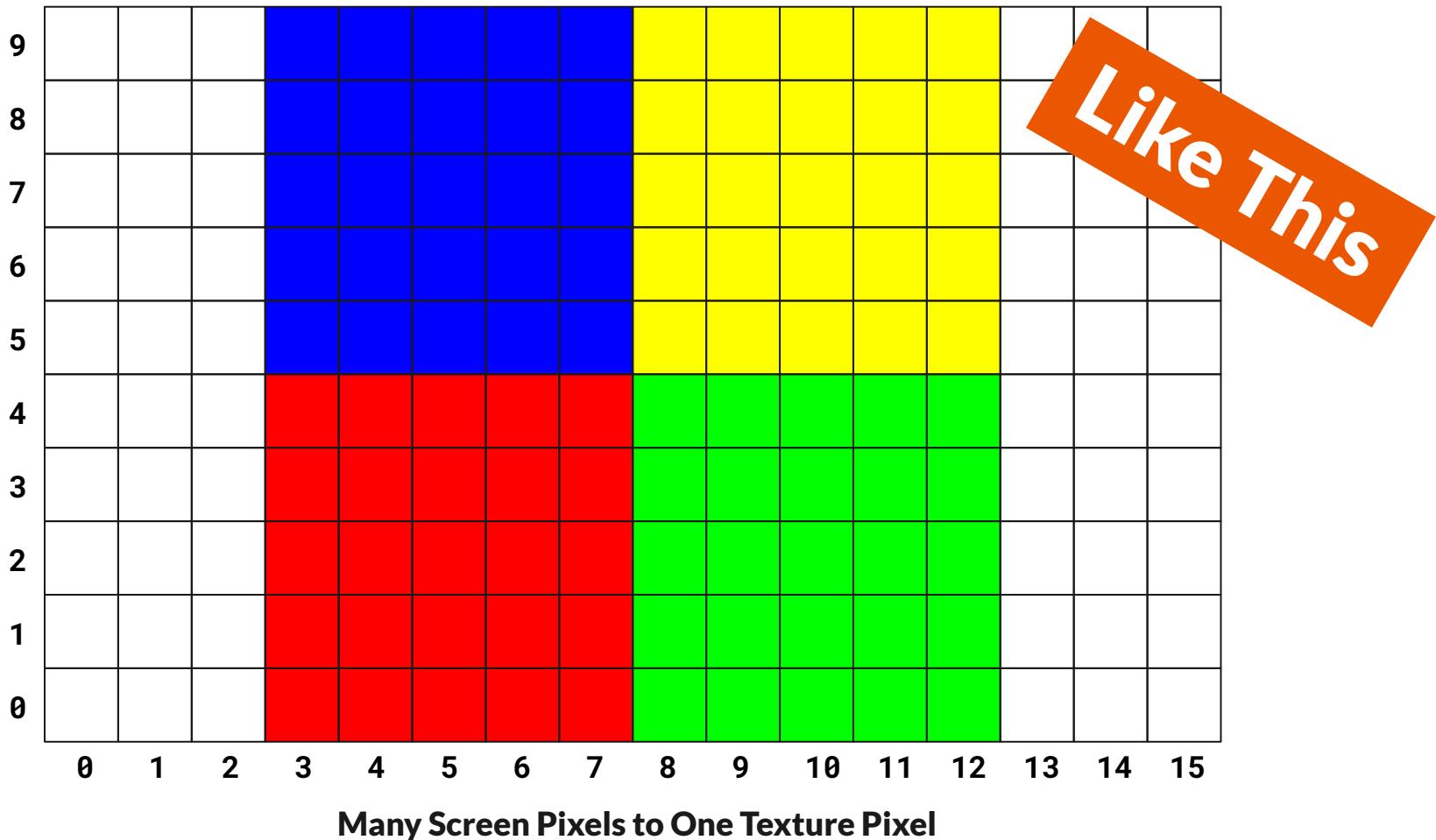


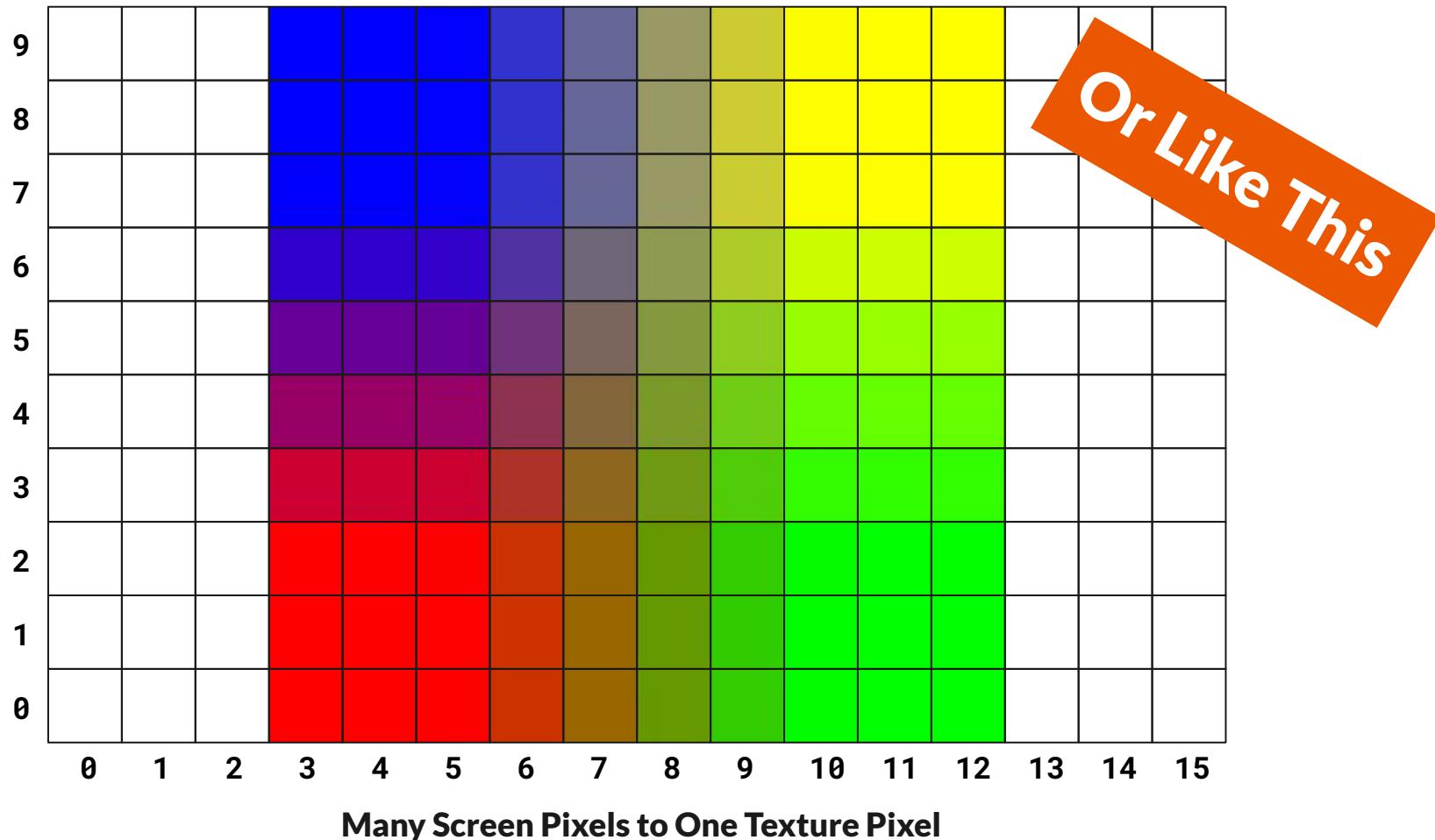
this is our texel we use it to
build the whole image

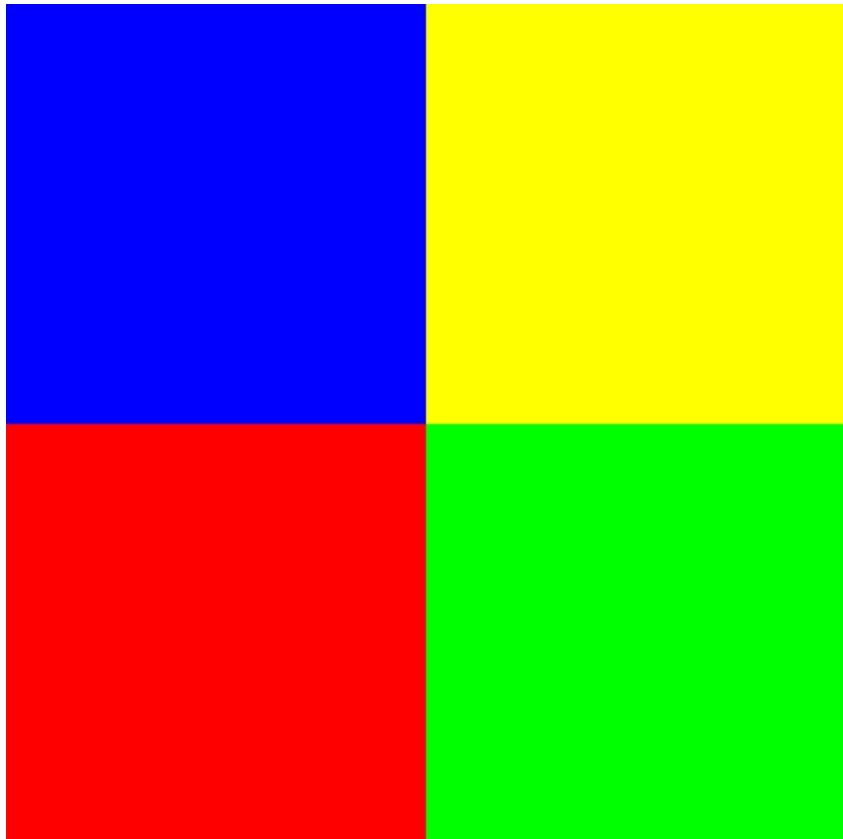




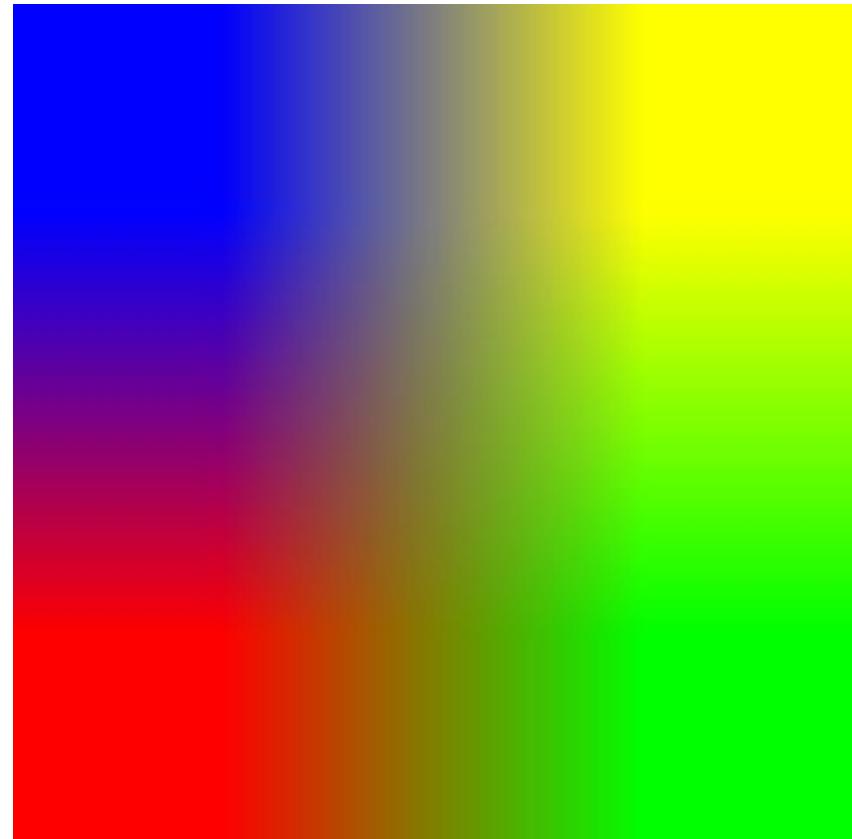
EASY CASE: One Screen Pixel to One Texture Pixel



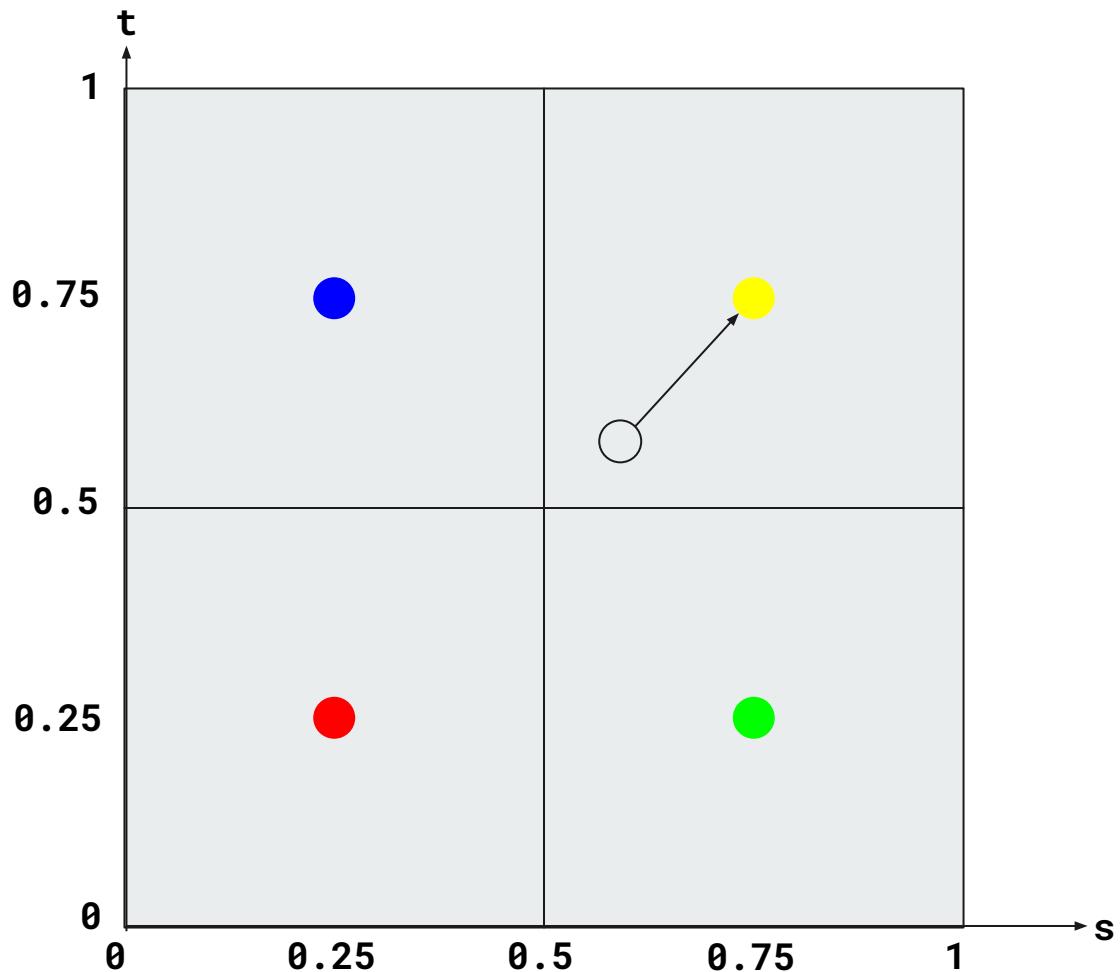




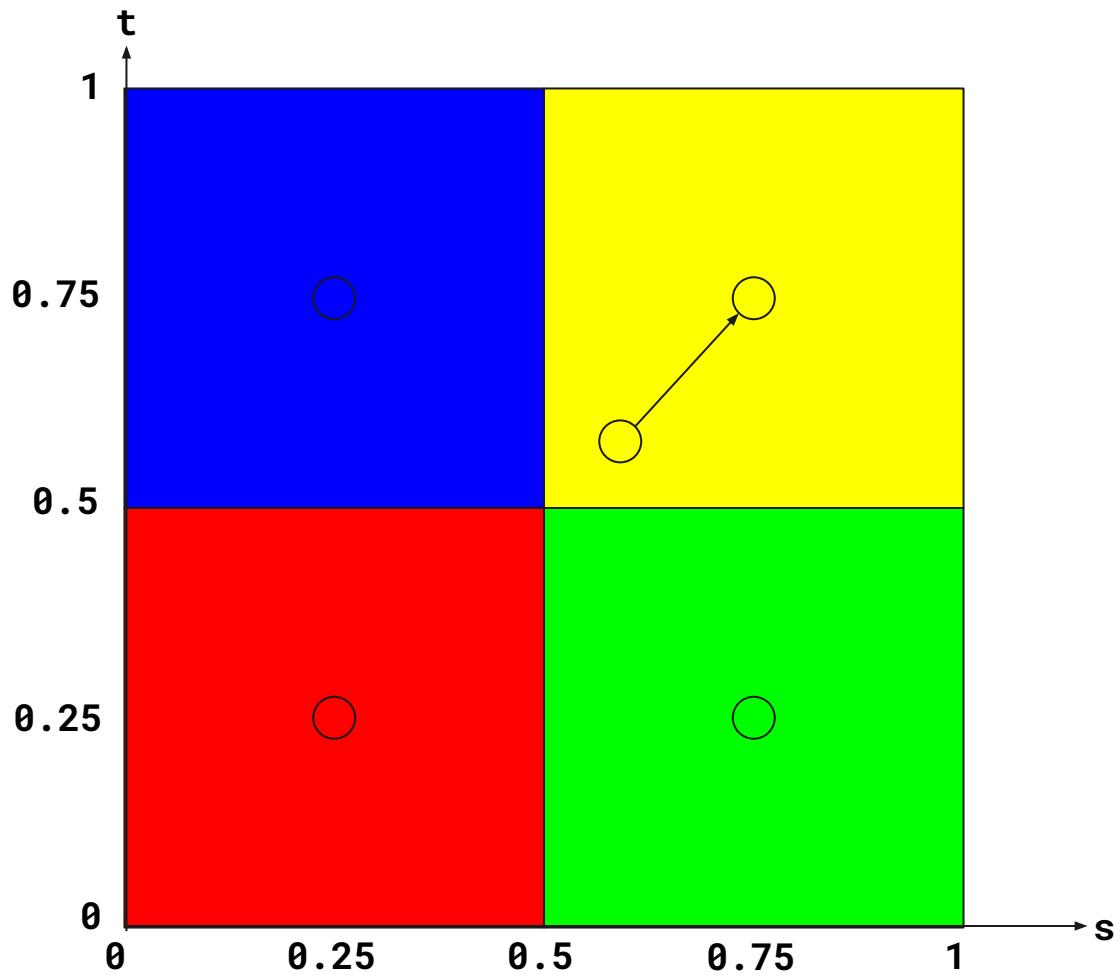
Nearest



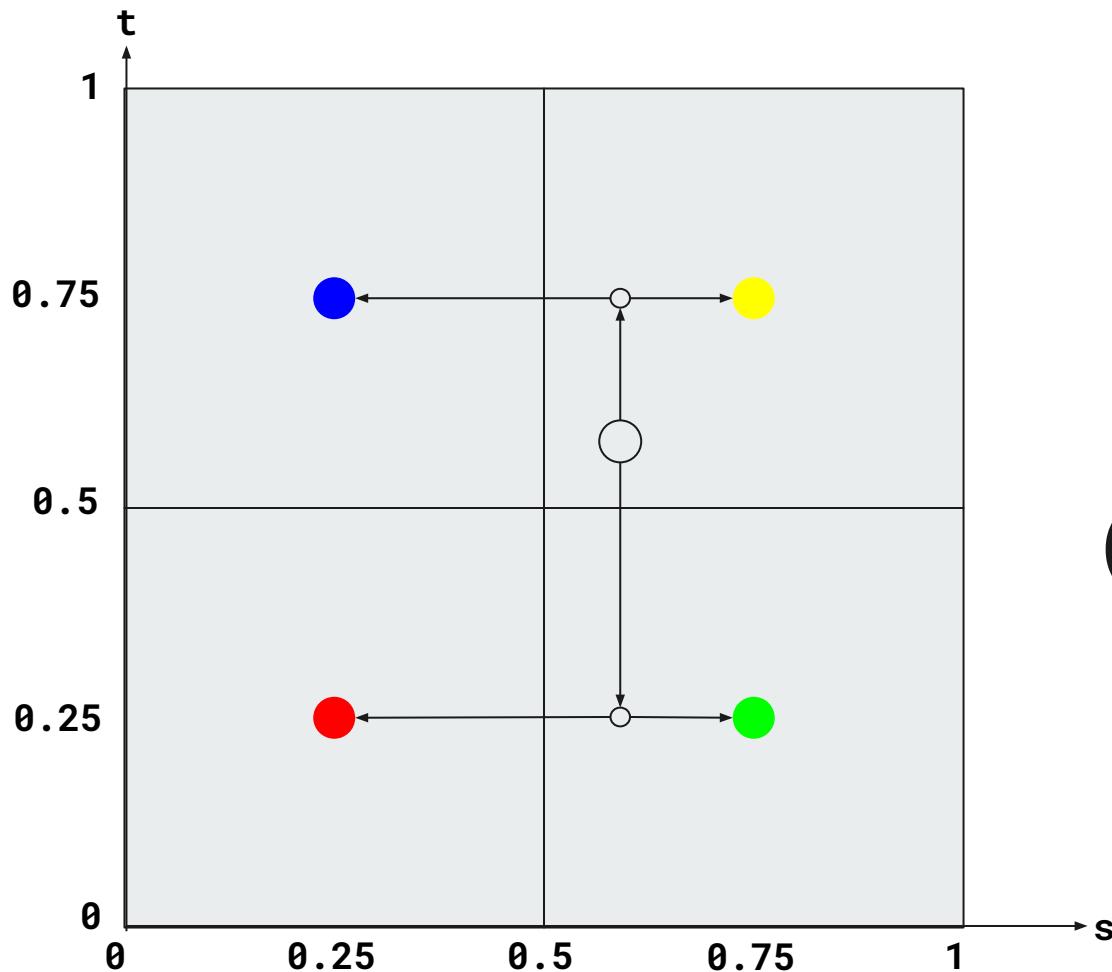
Linear (Bilinear)



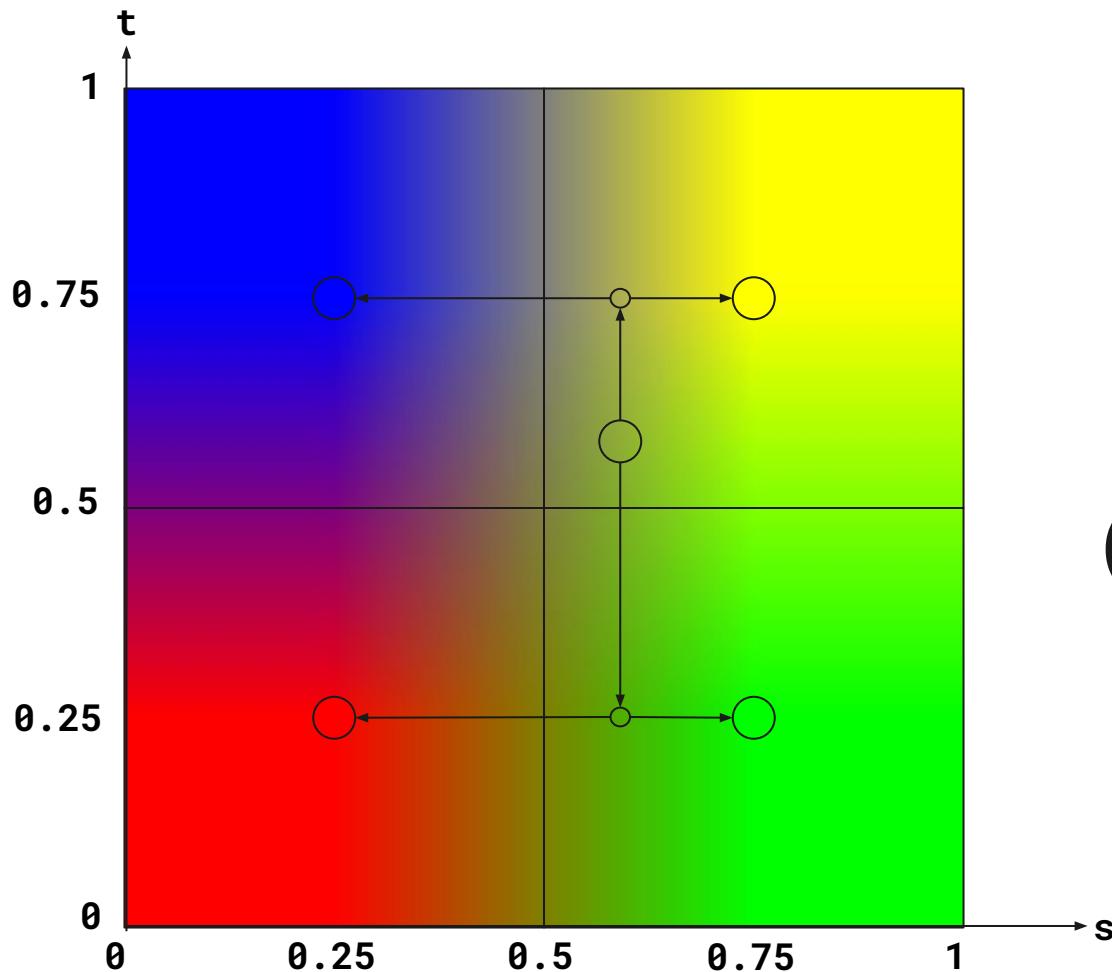
Nearest



Nearest



**Linear
(Bilinear)**



**Linear
(Bilinear)**

Magnification Filter

Many Screen Pixels to One Texture Pixel

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```

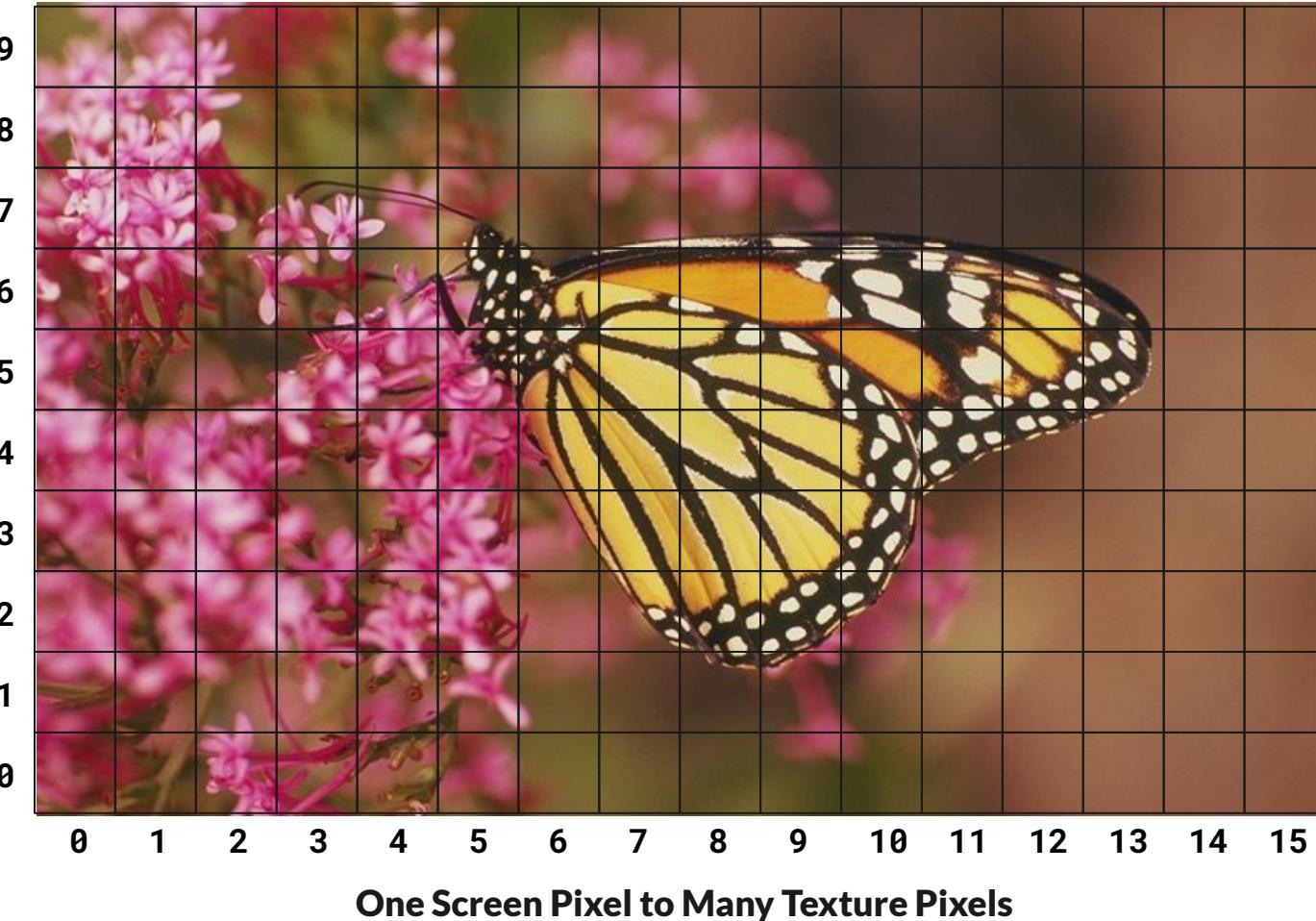
NEAREST

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

LINEAR (BILINEAR)

**Must
compute
average of
many texture
pixels to get
color.**

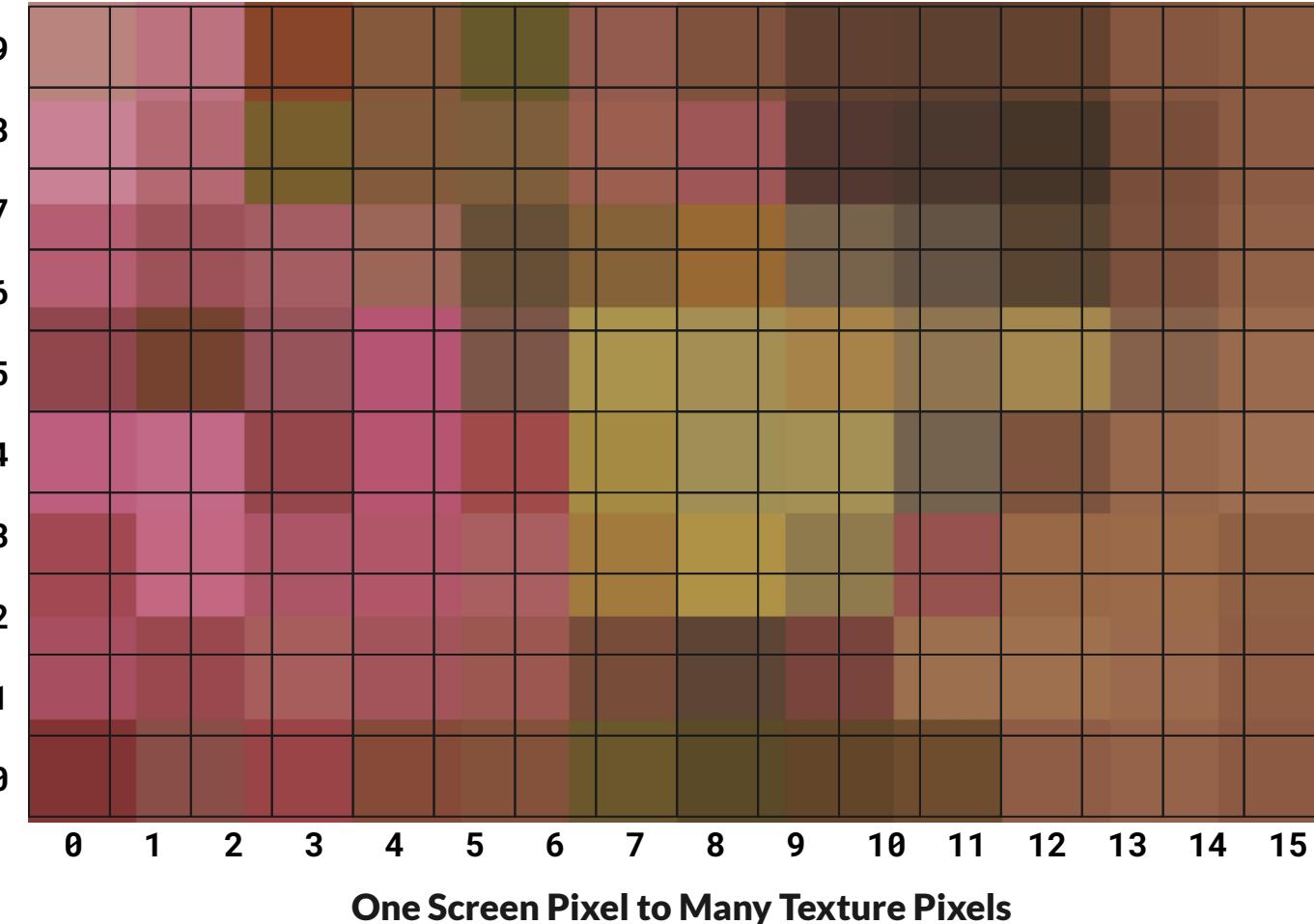
**TOO
EXPENSIVE.**



Should we really pick Level 6?

Use a lower
Mip Level.
CHEAP

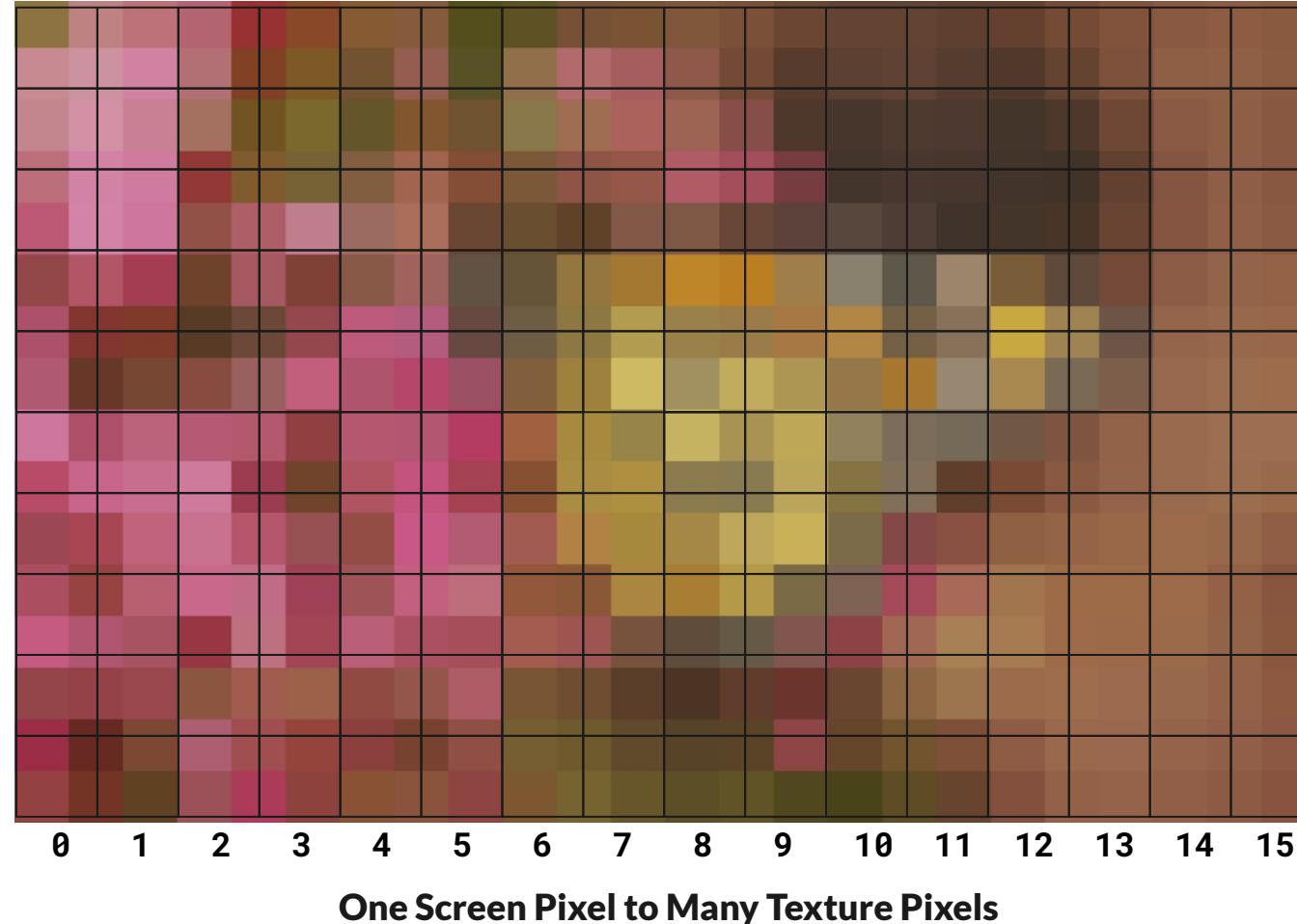
Level 6 Size
12x8
Screen Size
16x10

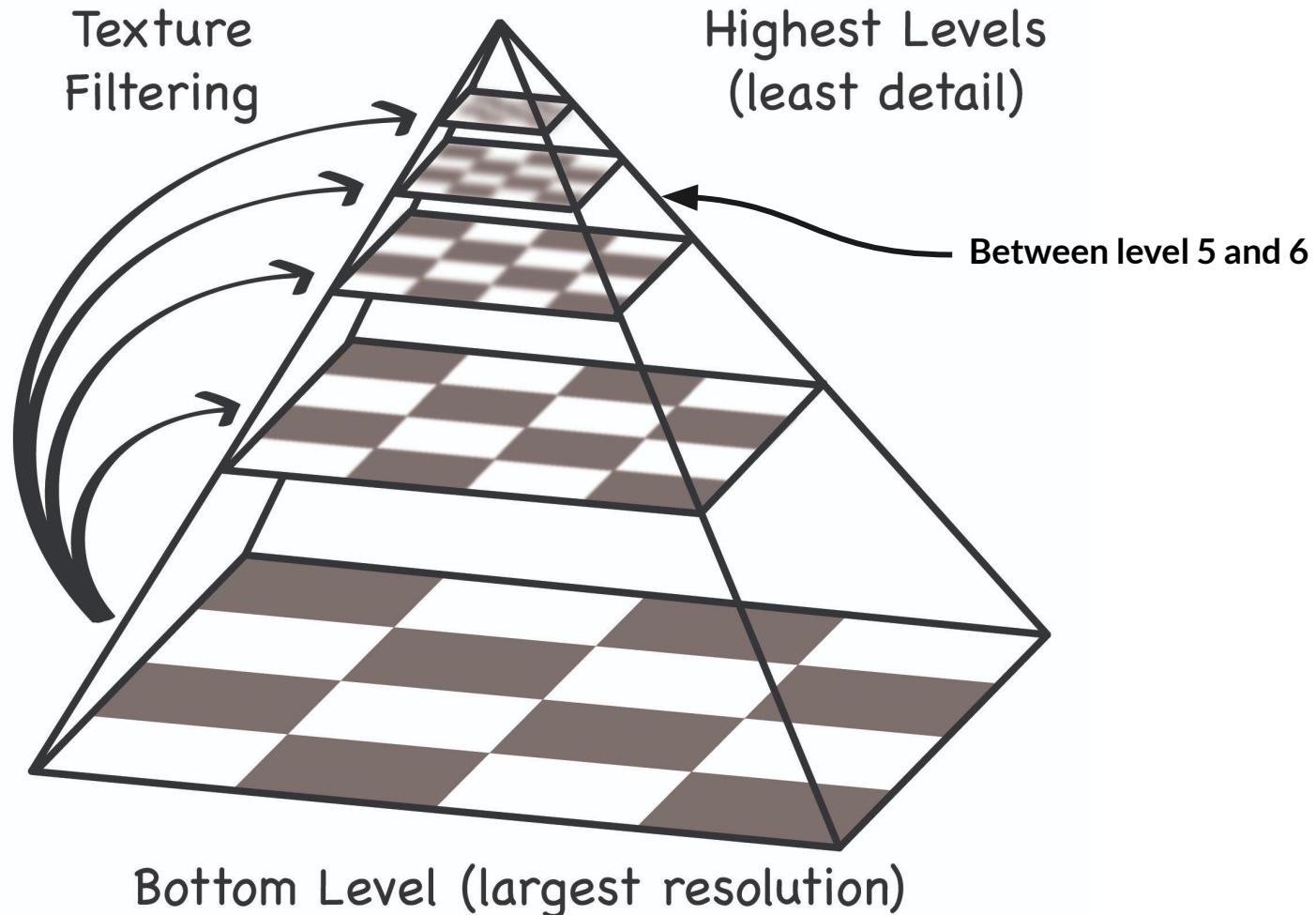


Or should we really pick Level 5?

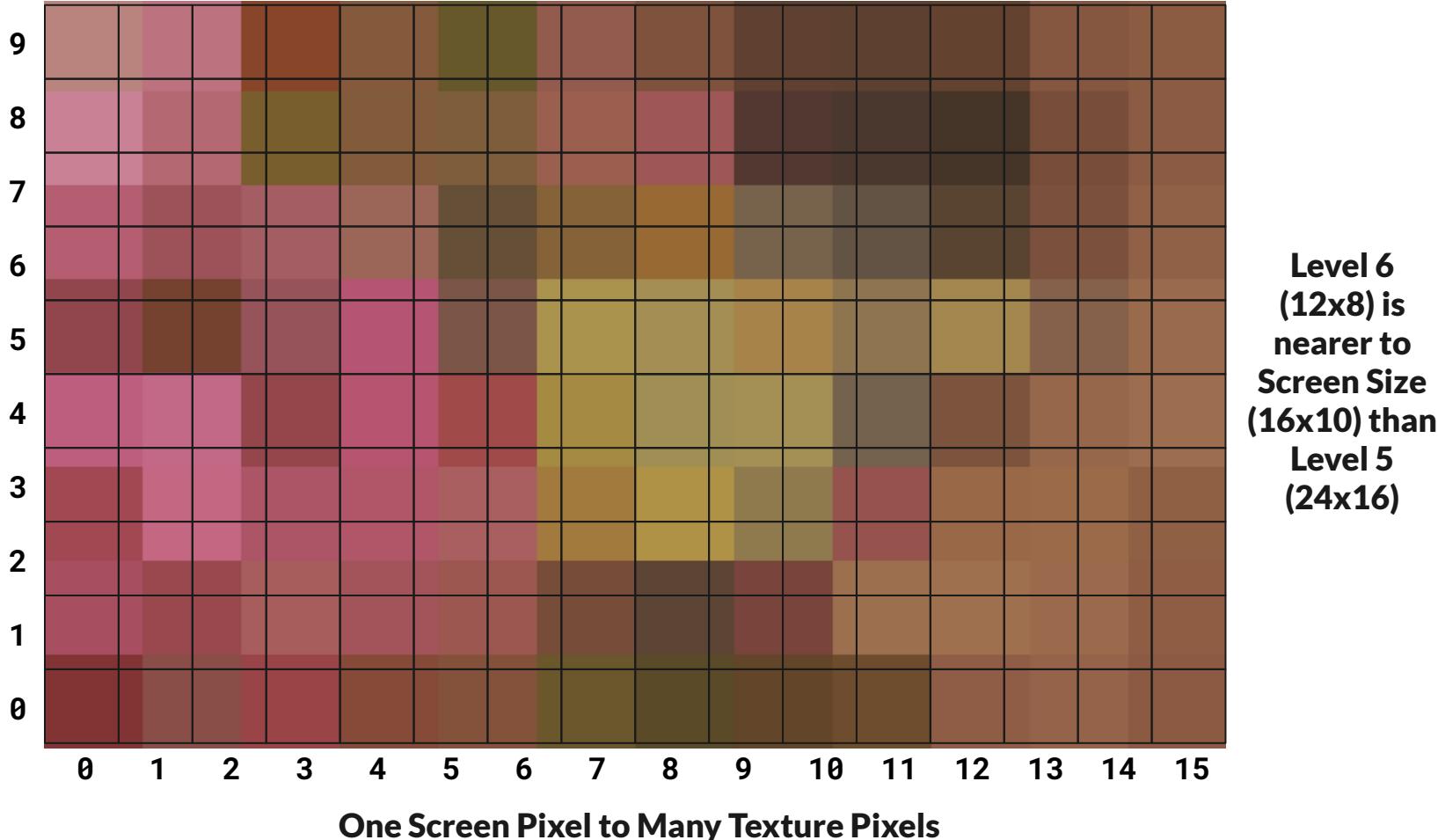
Use a lower
Mip Level.
CHEAP

Level 5 Size
24x16
Screen Size
16x10

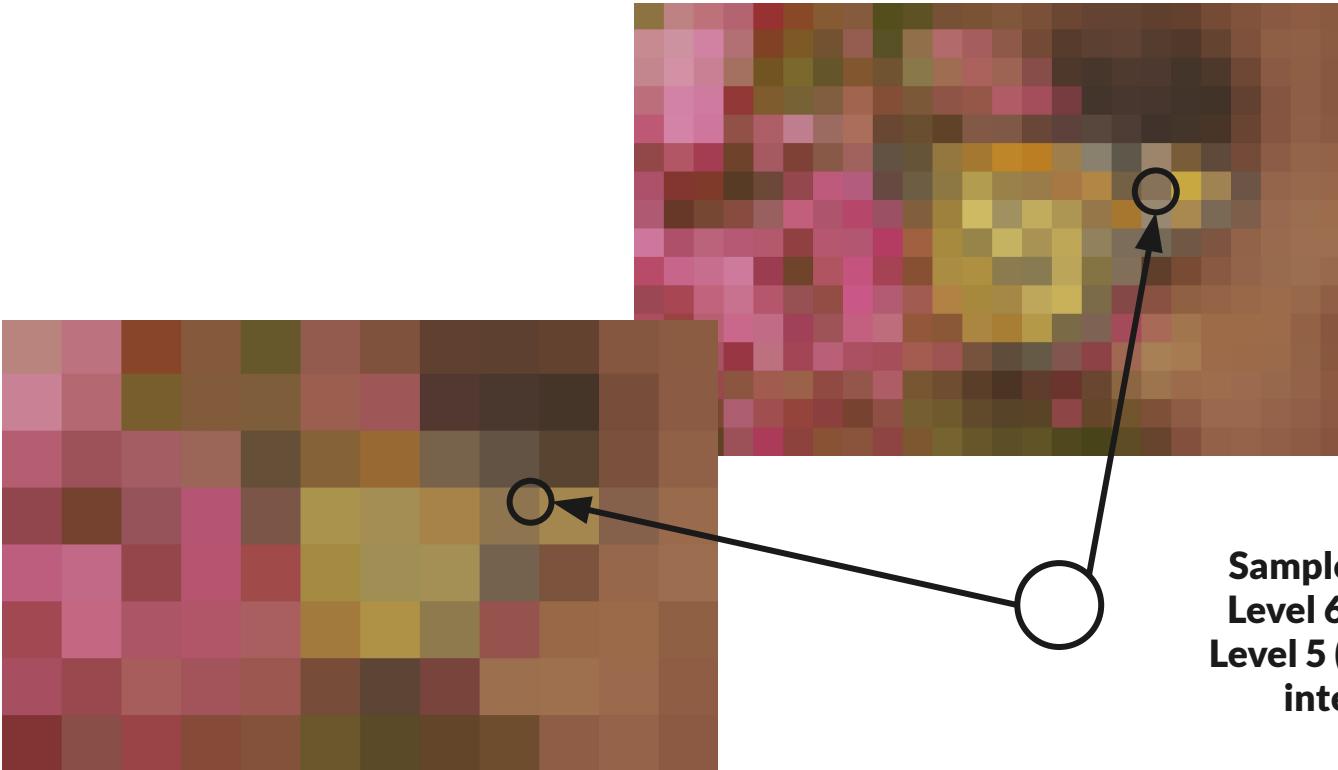




Option 1: “NEAREST” Mip Level



Option 2: “LINEAR” Mip Filtering



Minification Filter

One Screen Pixels to Many Texture Pixel

DOESN'T USE THE MIPMAP

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

NEAREST (FROM MIP LEVEL 0)

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

LINEAR (FROM MIP LEVEL 0)

THIS WILL JITTER A LOT WHILE MOVING

Minification Filter

One Screen Pixels to Many Texture Pixel

WITH MIPMAPS

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_NEAREST);
```

PICK NEAREST MIP LEVEL THEN PICK NEAREST PIXEL

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_NEAREST);
```

PICK NEAREST MIP LEVEL THEN FILTER PIXELS LINEARLY

THE DETAILS WILL CHANGE SUDDENLY AS
OBJECT DISTANCE CHANGES

Minification Filter

One Screen Pixels to Many Texture Pixel

WITH MIPMAPS

```
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_LINEAR);
```

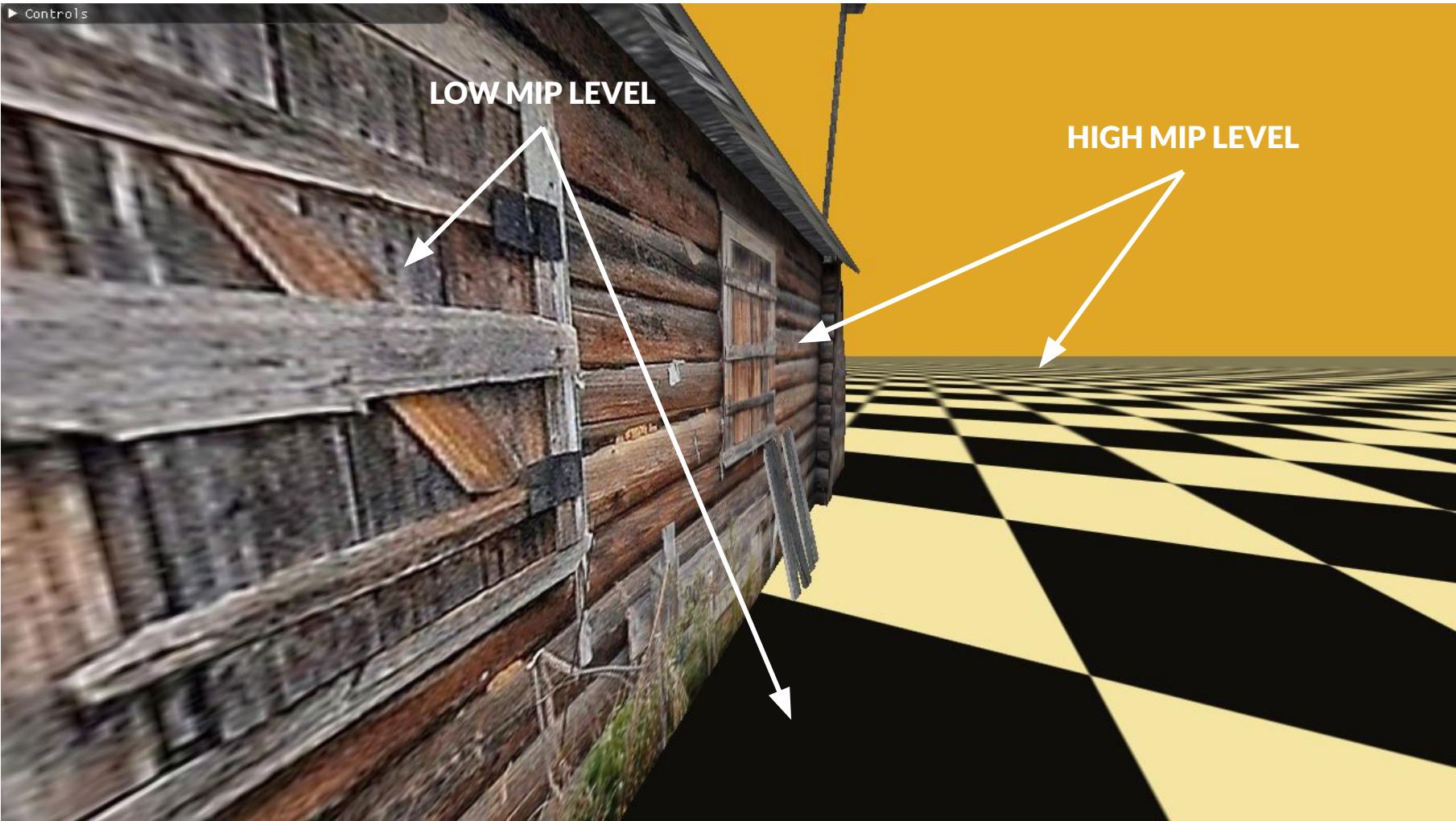
PICK THE NEAREST PIXEL IN EACH MIP LEVEL THEN INTERPOLATE LINEARLY BETWEEN LEVELS

```
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
```

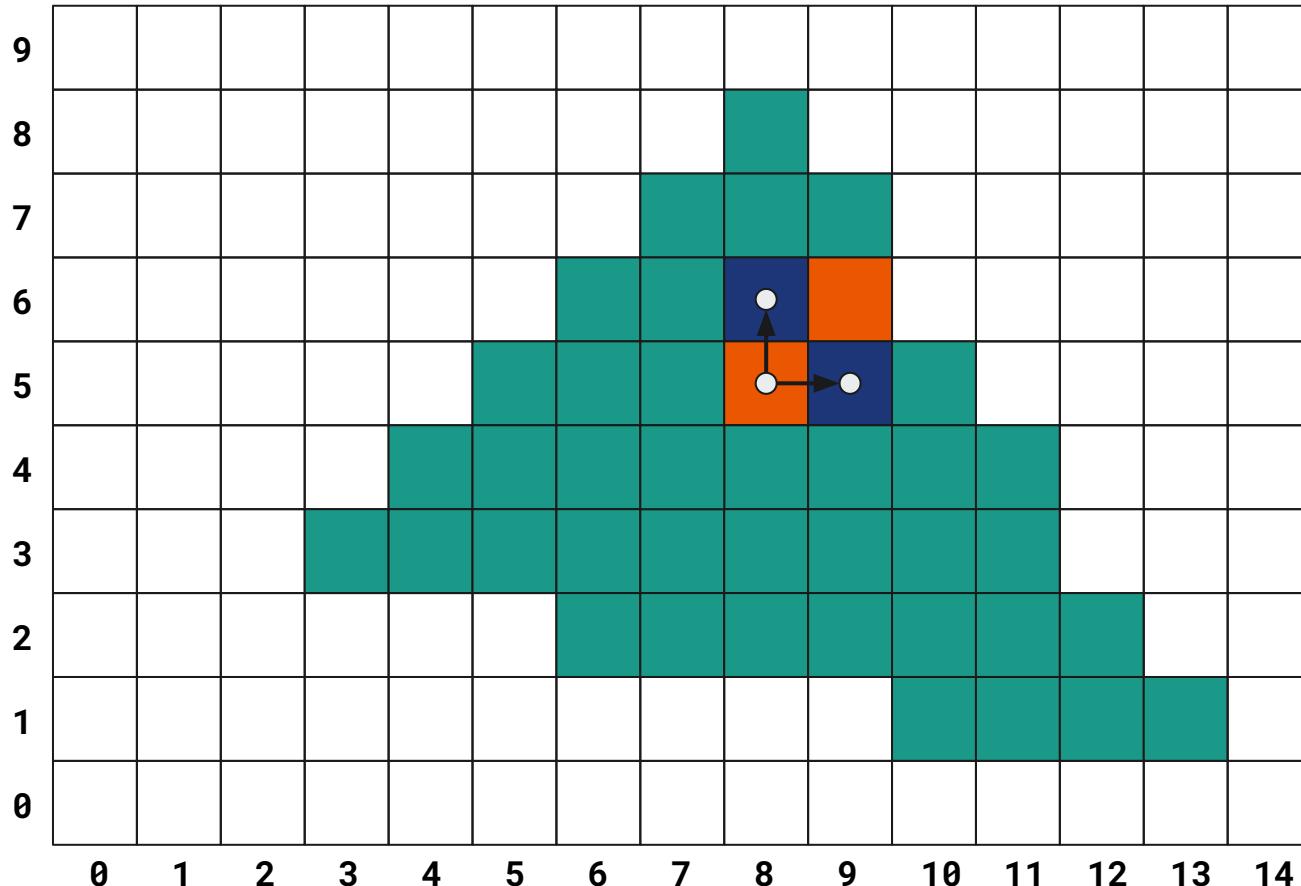
FILTER PIXELS LINEARLY IN EACH MIP LEVEL THEN INTERPOLATE LINEARLY BETWEEN LEVELS

GL_LINEAR_MIP_LINEAR IS THE BEST (AND MOST EXPENSIVE) OPTION FOR REALISM. COMMONLY CALLED “TRILINEAR FILTERING”

► Controls



How to calculate the required mip level



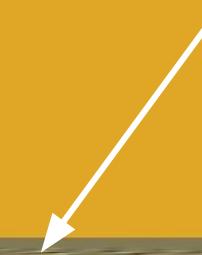
Use the numerical derivative of the texture coordinates.
GPUs compute fragments in a 2x2 block.

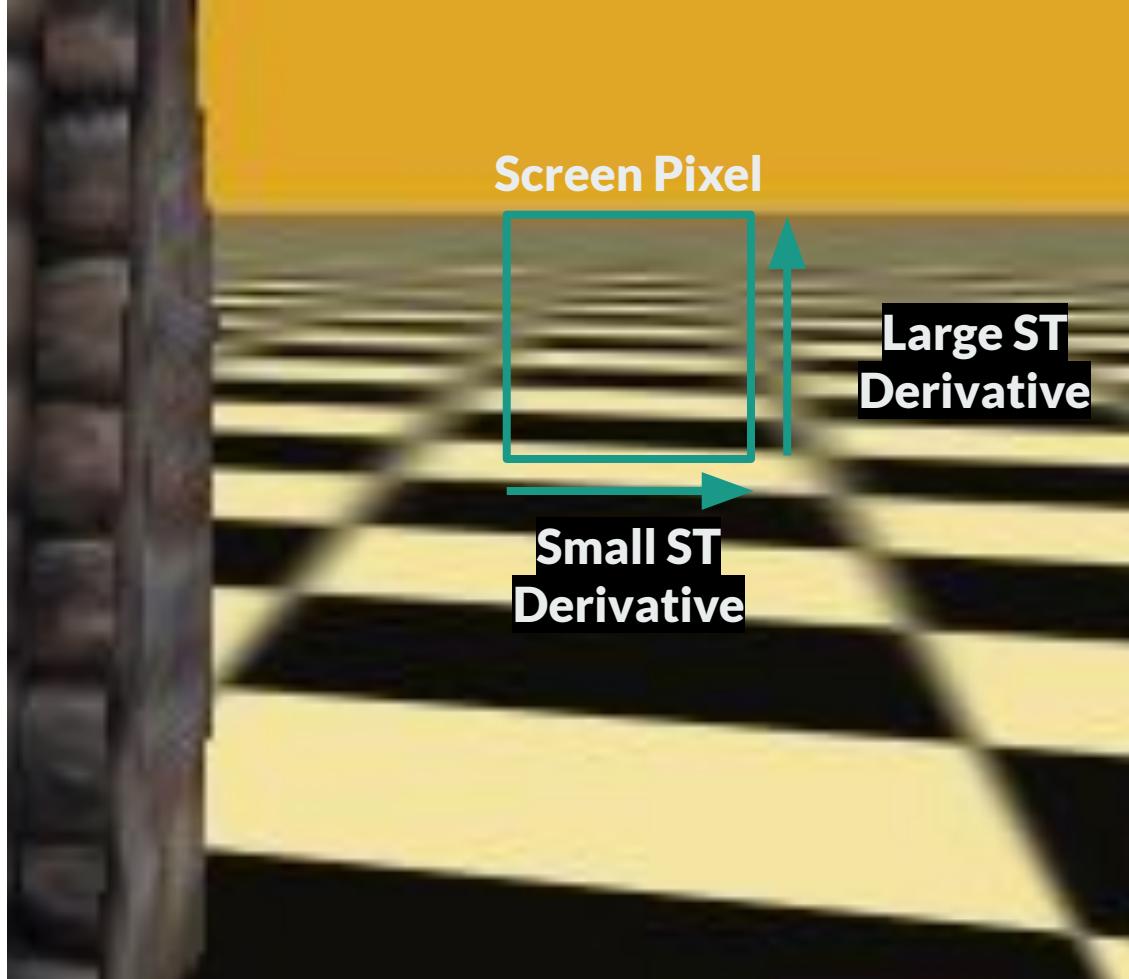
```
vec2 pixel = tex_coord *  
textureSize(sampler, 0);  
  
vec2 dx = dFdx(pixel);  
vec2 dy = dFdy(pixel);  
  
float d = max(  
    dot(dx, dx),  
    dot(dy, dy)  
)  
  
float lod = 0.5 * log2(d);
```

This is automatically computed in “texture” function. No need to write it ourselves.

► Controls

Too Blurry. Why?





Since the mip level formula picks the maximum derivative, the mip level will be high even if the derivative is small on one dimension

```
vec2 pixel = tex_coord *  
textureSize(sampler, 0);
```

```
vec2 dx = dFdx(pixel);  
vec2 dy = dFdy(pixel);
```

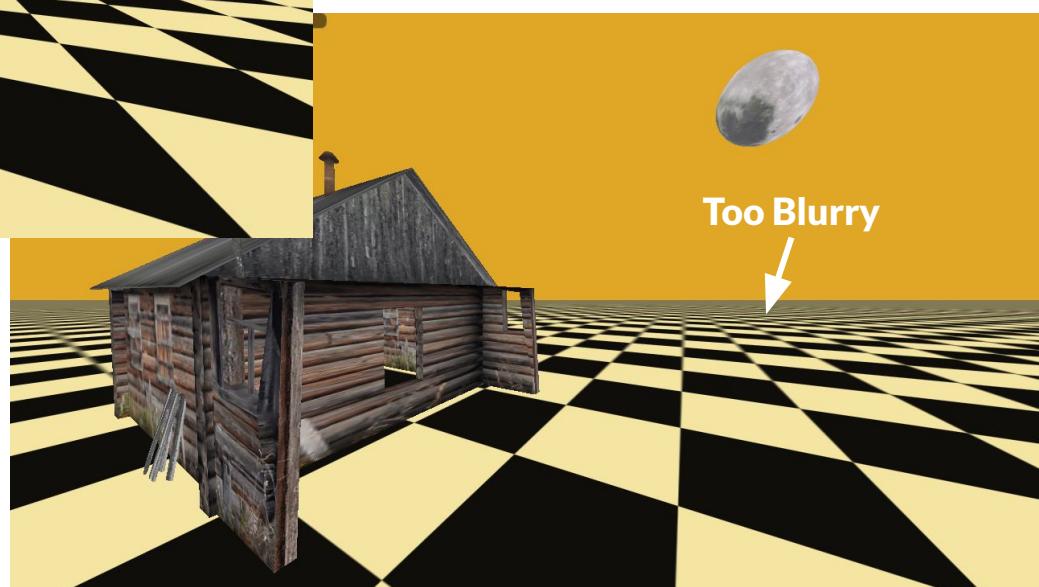
```
float d = max(  
    dot(dx, dx),  
    dot(dy, dy)  
)
```

```
float lod = 0.5 * log2(d);
```

► Controls

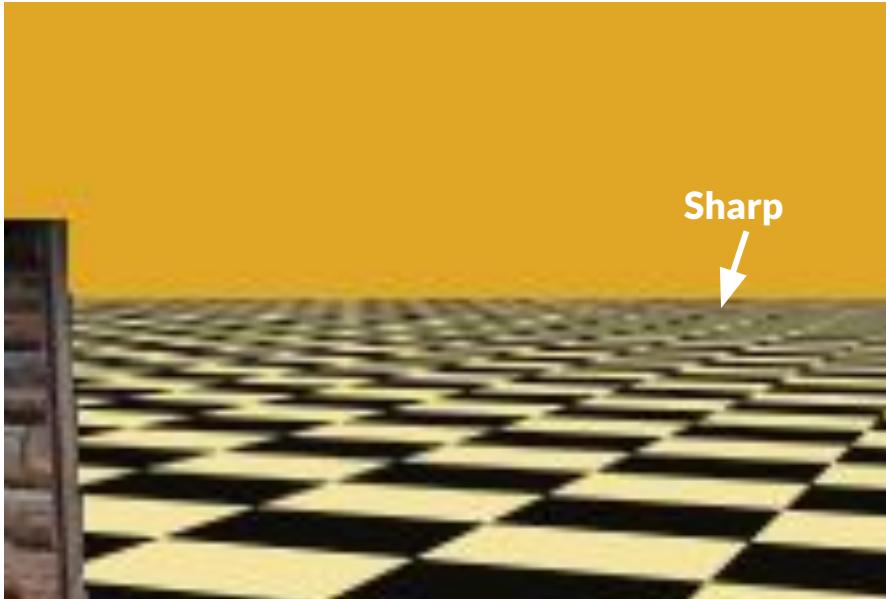


Maximum Anisotropy = 16

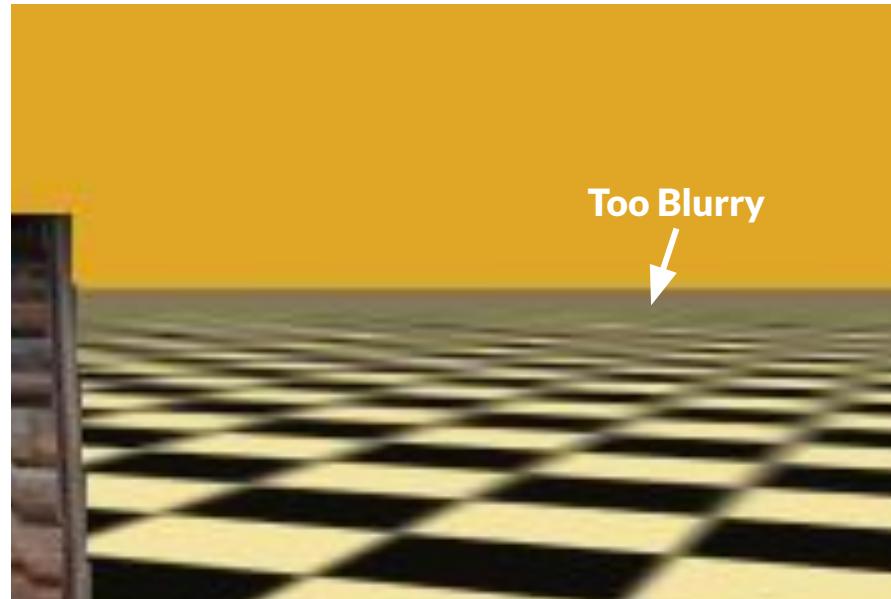


Maximum Anisotropy = 1

Too Blurry

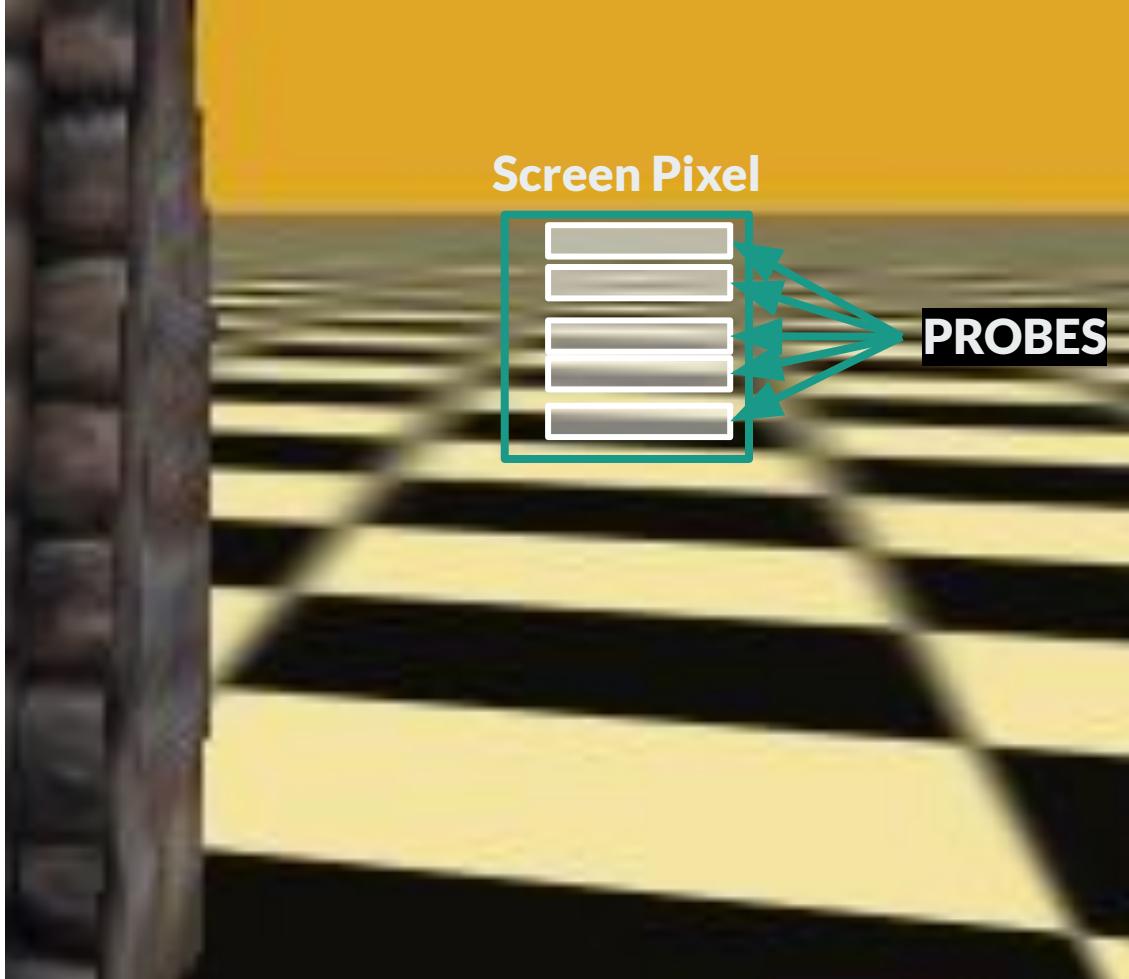


Maximum Anisotropy = 16



Maximum Anisotropy = 1

Too Blurry



Anisotropic Filtering uses multiple probes to sample the color and the samples are then merged together.

Each probe is sampled with filtering (with mipmaps).

The details of the anisotropic filtering mechanism is not specified by OpenGL so it is vendor specific.

Anisotropic filtering is not a replacement for mipmap filtering; they are used commonly together.

Anisotropic Filtering

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, 16.0f);
```

Pick maximum anisotropy for filtering the texture (Minimum possible value = 1.0f)

```
GLfloat max_anisotropy_upper_bound;  
glGetFloatv(GL_MAX_TEXTURE_MAX_ANISOTROPY_EXT, &max_anisotropy_upper_bound);
```

Ask OpenGL for the maximum possible value for “Maximum Anisotropy”.

The highest possible maximum anisotropy depends on the GPU capabilities.

0.0, 1.0

1.0, 1.0

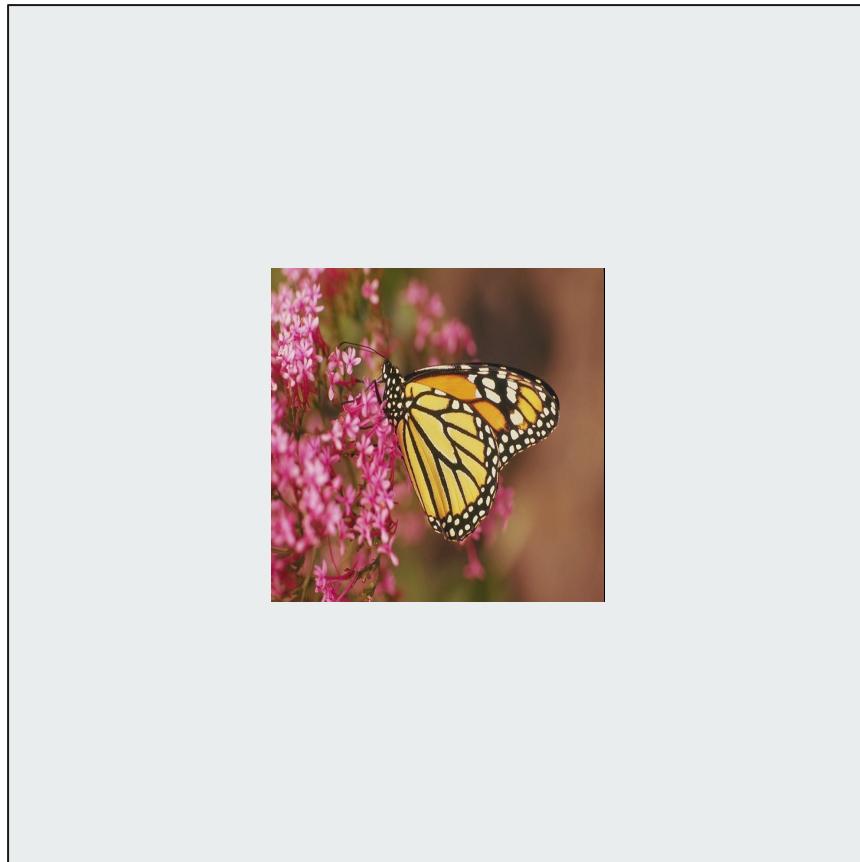


0.0, 0.0

1.0, 0.0

-1.0, 2.0

2.0, 2.0



-1.0, -1.0

2.0, -1.0



-1.0, 2.0

2.0, 2.0



GL_REPEAT

-1.0, -1.0

2.0, -1.0

-1.0, 2.0

2.0, 2.0

GL_MIRRORED_REPEAT



-1.0, -1.0

2.0, -1.0

-1.0, 2.0

2.0, 2.0

GL_CLAMP_TO_EDGE



-1.0, -1.0

2.0, -1.0

-1.0, 2.0

2.0, 2.0

GL_CLAMP_TO_BORDER



-1.0, -1.0

2.0, -1.0

Wrapping Mode

Wrapping mode over the S coordinate

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
```

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);
```

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
```

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
```

Wrapping mode over the T coordinate

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

```
...
```

Wrapping Mode

```
float color[] = {1.0f, 0.0f, 0.0f, 1.0f};  
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, color);
```

Choose border color

Sampler State

```
glBindTexture(GL_TEXTURE_2D, texture);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, magnification_filter);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, minification_filter);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, wrap_s);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, wrap_t);
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, border_color);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, max_anisotropy);
```

There are more states that are not defined here

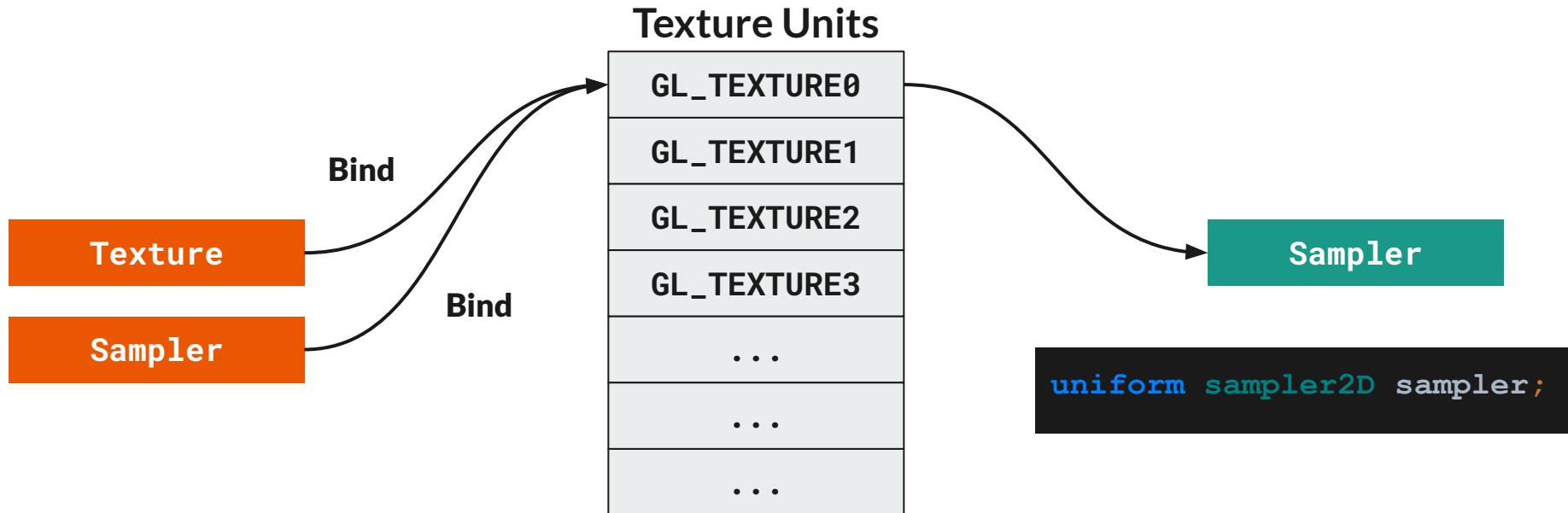
Using “`glTexParameter`”, the sampler state must be defined for each texture individually.

Sampler Objects

```
GLuint sampler;  
glGenSamplers(1, &sampler);  
  
glSamplerParameteri(sampler, GL_TEXTURE_MAG_FILTER, magnification_filter);  
glSamplerParameteri(sampler, GL_TEXTURE_MIN_FILTER, minification_filter);  
glSamplerParameteri(sampler, GL_TEXTURE_WRAP_S, wrap_s);  
glSamplerParameteri(sampler, GL_TEXTURE_WRAP_T, wrap_t);  
glSamplerParameterfv(sampler, GL_TEXTURE_BORDER_COLOR, border_color);  
glSamplerParameterf(sampler, GL_TEXTURE_MAX_ANISOTROPY_EXT, max_anisotropy);
```

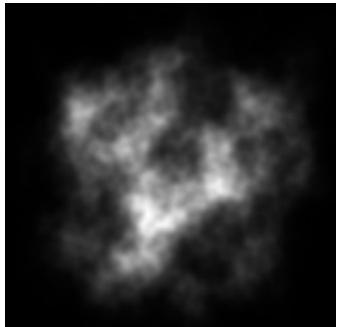
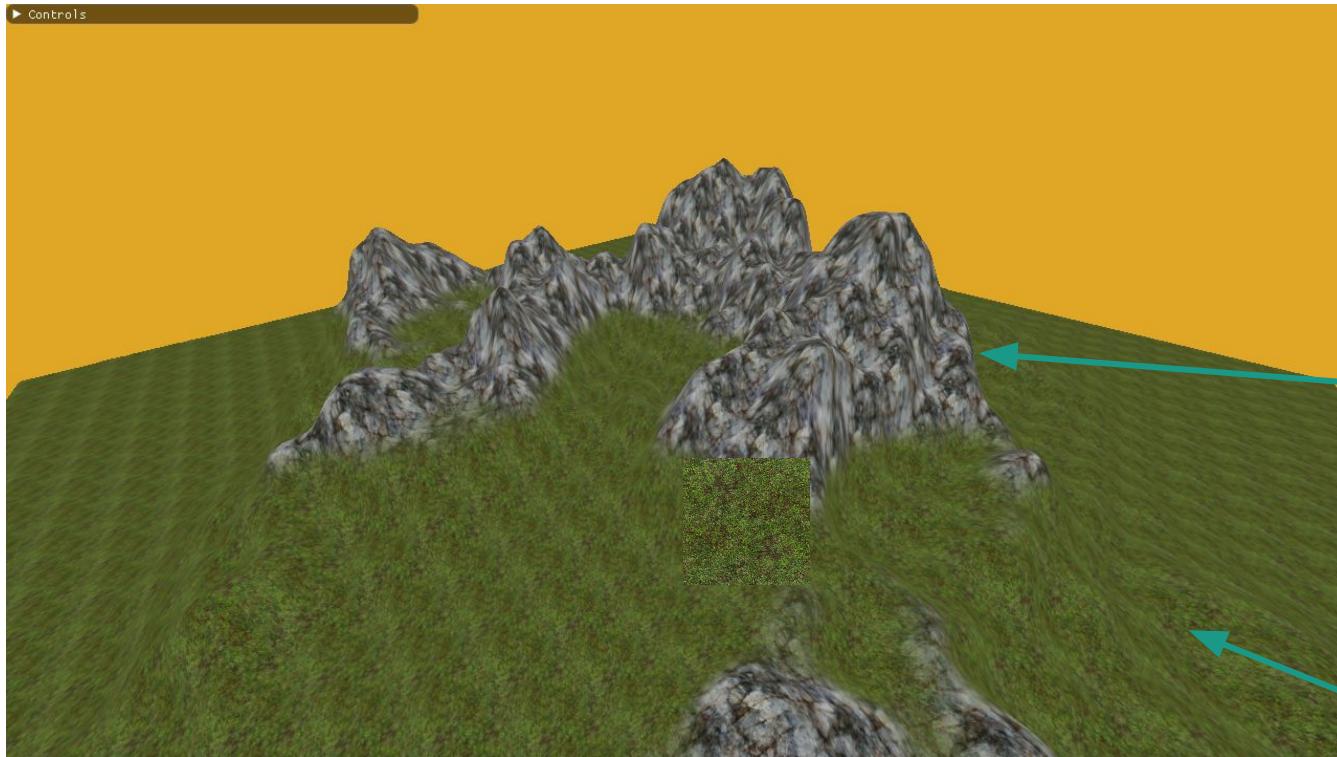
A sampler object can be used with multiple textures
(no need to redefine sampler state for each texture).

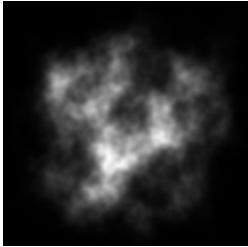
```
glActiveTexture(GL_TEXTURE0);
 glBindTexture(GL_TEXTURE_2D, texture);
 glBindSampler(0, sampler);
 GLuint sampler_loc = glGetUniformLocation(program, "sampler");
 glUniform1i(sampler_loc, 0);
```



Textures were mainly created to store color data

**But you can store any kind of data
in it as long as they are vectors.**





Height Sampler



Color Sampler

Texture Units

GL_TEXTURE0

GL_TEXTURE1

GL_TEXTURE2

GL_TEXTURE3

...

...

...

```
uniform sampler2D height_sampler;  
uniform sampler2D top_sampler;  
uniform sampler2D bottom_sampler;
```

```
glGenSamplers(1, &height_sampler);

glSamplerParameteri(height_sampler, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glSamplerParameteri(height_sampler, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glSamplerParameteri(height_sampler, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glSamplerParameteri(height_sampler, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

```
glGenSamplers(1, &color_sampler);

glSamplerParameteri(color_sampler, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glSamplerParameteri(color_sampler, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glSamplerParameteri(color_sampler, GL_TEXTURE_WRAP_S, GL_REPEAT);
glSamplerParameteri(color_sampler, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, height_texture);
glBindSampler(0, height_sampler);
glUniform1i(glGetUniformLocation(program, "height_sampler") , 0);

glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, top_texture);
glBindSampler(1, color_sampler);
glUniform1i(glGetUniformLocation(program, "top_sampler") , 1);

glActiveTexture(GL_TEXTURE2);
glBindTexture(GL_TEXTURE_2D, bottom_texture);
glBindSampler(2, color_sampler);
glUniform1i(glGetUniformLocation(program, "bottom_sampler") , 2);
```

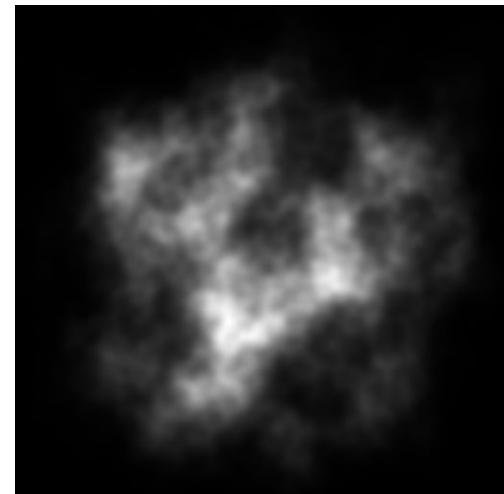
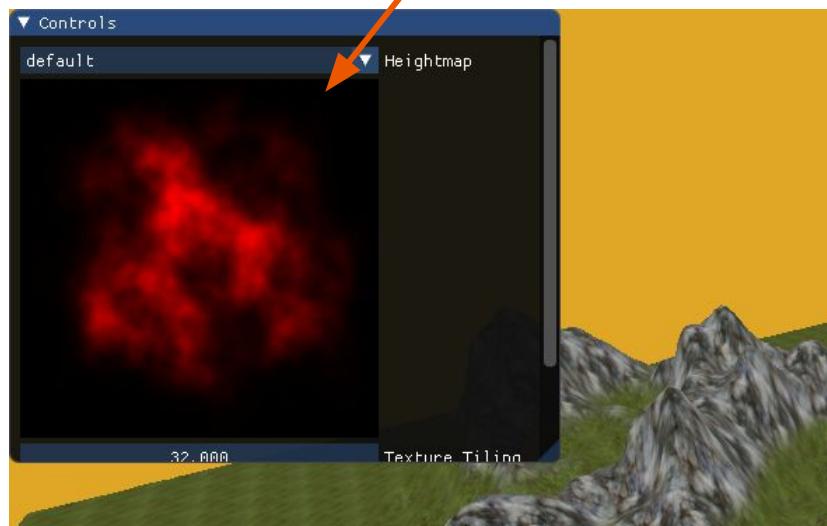
glad/gl.h

```
#define GL_TEXTURE0 0x84C0
#define GL_TEXTURE1 0x84C1
#define GL_TEXTURE2 0x84C2
#define GL_TEXTURE3 0x84C3
#define GL_TEXTURE4 0x84C4
#define GL_TEXTURE5 0x84C5
#define GL_TEXTURE6 0x84C6
#define GL_TEXTURE7 0x84C7
...
```

$$\text{GL_TEXTUREN} = \text{GL_TEXTURE0} + N$$

```
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);  
glTexImage2D(GL_TEXTURE_2D, 0, GL_R8, size.x, size.y, 0, GL_RED, GL_UNSIGNED_BYTE, data);
```

Internal Format (only red channel)



Part of Vertex Shader

```
float height = texture(height_sampler, tex_coord).r;  
  
gl_Position = transform * vec4(position + vec3(0, height, 0), 1.0);  
  
vsout.terrain_height = height;
```

Part of Fragment Shader

```
vec4 top_color = texture(terrain_top_sampler, fsin.tex_coord);  
vec4 bottom_color = texture(terrain_bottom_sampler, fsin.tex_coord);  
  
float mix_factor = smoothstep(bottom, top, fsin.terrain_height);  
  
frag_color = mix(bottom_color, top_color, mix_factor);
```

Frame Buffers



- 2D Array of Pixels
- Color Data
- We can draw on it



- 2D Array of Pixels
- Color Data
- Can we draw on it???

Create and Bind Frame Buffer

```
GLuint frame_buffer;  
glGenFramebuffers(1, &frame_buffer);  
 glBindFramebuffer(GL_DRAW_FRAMEBUFFER, frame_buffer);
```



Bind Target can be:

- GL_DRAW_FRAMEBUFFER to which we can draw.
- GL_READ_FRAMEBUFFER from which we can read pixels.
- GL_FRAMEBUFFER which can be used for both reading and drawing.

Create and Attach a Color Target Texture

```
glGenTextures(1, &texture);
 glBindTexture(GL_TEXTURE_2D, texture);

 GLuint mip_levels = glm::floor(glm::log2(glm::max<float>(width, height))) + 1;
 glTexStorage2D(GL_TEXTURE_2D, mip_levels, GL_RGBA8, width, height);
```

Create an Empty Texture

How many mips to allocate

```
glFramebufferTexture2D(GL_DRAW_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D,
 texture, 0);
```

Level (must be 0)

You can attach multiple color target to a single framebuffer (useful for some techniques).

Create and Attach a Depth Target Texture

```
glGenTextures(1, &texture);
 glBindTexture(GL_TEXTURE_2D, texture);

 glTexStorage2D(GL_TEXTURE_2D, 1, GL_DEPTH_COMPONENT32, width, height);

 glFramebufferTexture2D(GL_DRAW_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D,
 texture, 0);
```

Depth only needs 1 mip level

Level (must be 0)

Depth should be attached a depth attachment



Check if Framebuffer is complete

```
if (glCheckFramebufferStatus(GL_DRAW_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE) {
    std::cerr << "Frame buffer is incomplete" << std::endl;
}
```

There are some rules for FrameBuffer completeness. All of them are mentioned here:
[https://www.khronos.org/opengl/wiki/Framebuffer Object#Framebuffer Completeness](https://www.khronos.org/opengl/wiki/Framebuffer_Object#Framebuffer_Completeness)

Draw

STEP 1:

```
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, frame_buffer);  
glViewport(0, 0, framebuffer_width, framebuffer_height);
```

STEP 2:

Draw stuff to framebuffer

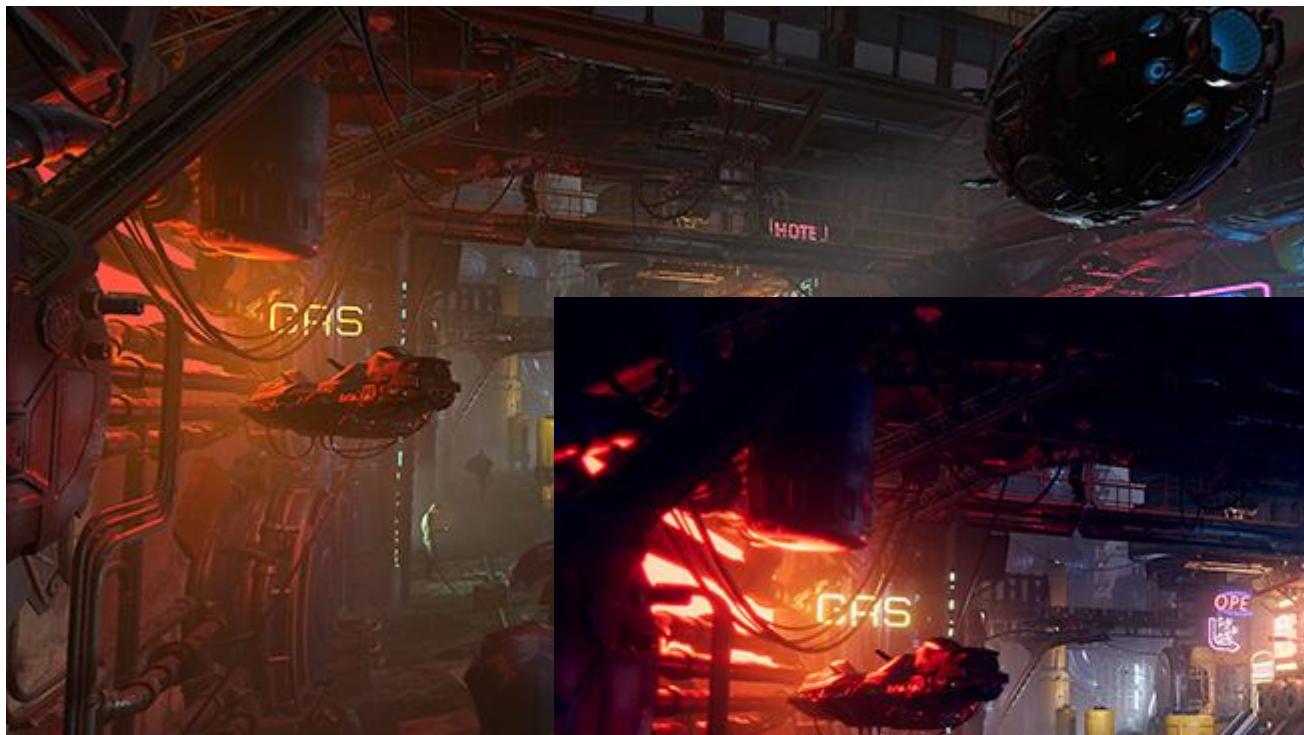
STEP 3:

```
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);  
glViewport(0, 0, window_width, window_height);
```

STEP 4:

Draw stuff to window (You can use the render target texture here).

Without Post Processing



With Post Processing





Planar Reflection



Thank you