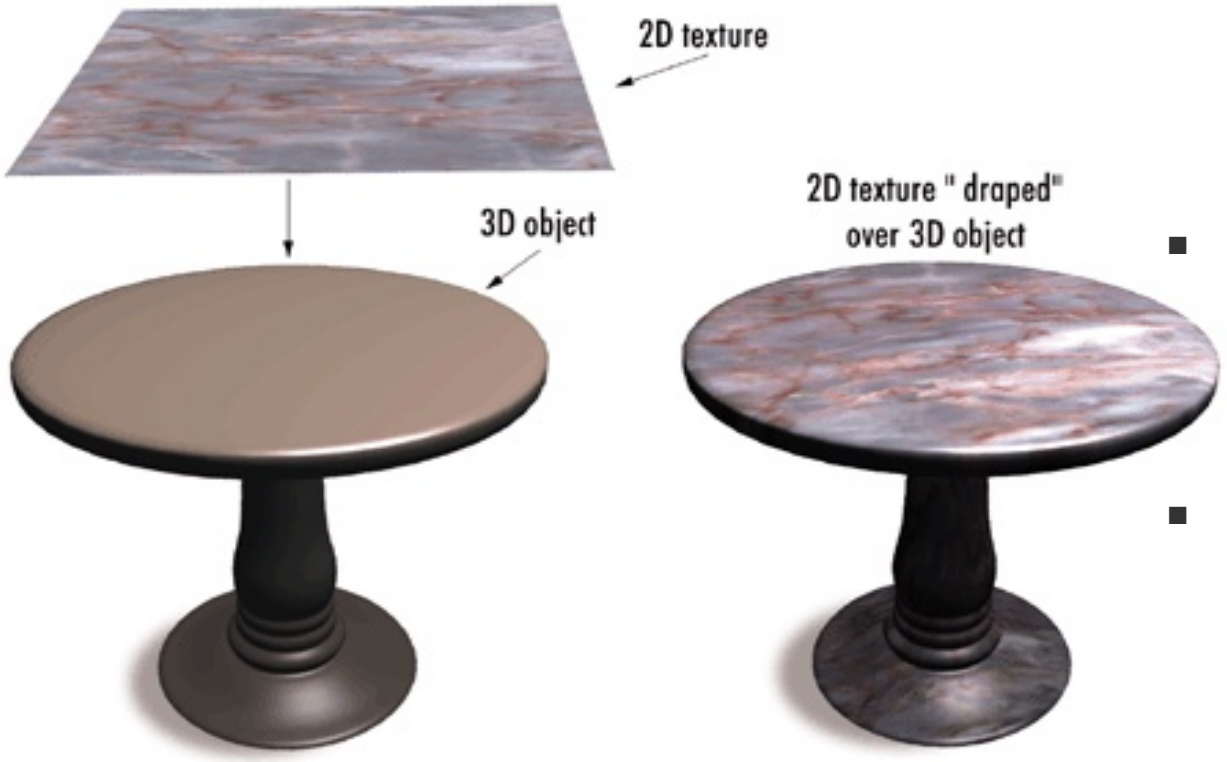
Texture mapping in OpenGL

- Texture Creation
- Texture Parameters
- Using the texture in a shader

Texture basics



- Textures store images and other state
- Colors from the texture can be applied to fragments during fragment processing
- Many other applications (normal maps, lightmaps, etc...)
- Textures will be stored on the graphics server (GPU) just like VBOs.

Creating a texture

- Create a texture object
  - `glGenTextures(...)`
- Bind the texture object
  - `glBindTexture(GLenum target, GLuint textureID);`
    - Most common target is GL_TEXTURE_2D
    - There are other targets (3D, CUBE)
- Upload the texture image
  - `glTexImage2D(...);`
    - Specify format, type, width, height, pointer to image in client memory.
- Specify parameters
  - `glTexParameter*(...)`

Creating a texture

- **`glGenTextures(unsigned int n, unsigned int* ids)`**
  - n: number of ids to generate
  - ids: array in which the generated ids will be stored

Creating a texture

- **`glBindTexture(GLenum target, unsigned int id)`**
- target: represents the “dimensionality” of the texture
  - `GL_TEXTURE_1D`
  - `GL_TEXTURE_2D`
  - `GL_TEXTURE_3D`
  - `GL_TEXTURE_CUBE_MAP`
  - `etc...`
- **id**: the id of the texture previously generated with glGenTextures
- The first time a texture is bound, the object is created in the OpenGL state based on the target
- The bound texture object is used for subsequent texture operations and parameter setting

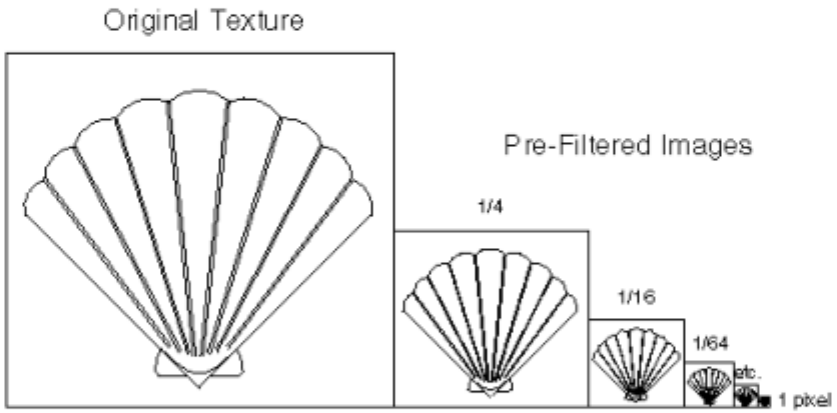
Creating a texture

- Upload the texture data from a typed array
  - `glTexImage2D(target, level, iformat, width, height, border, format, type, pixels);`

    - **level**: mipmap level (0 is the base level)
    - **iformat**: internal format of the texture on GPU
    - **width, height**: size in pixels
    - **border**: texel border width: use **0**
    - **format**: format of the data in the array
    - **type**: data type of the array
    - **pixels**: pointer to the image data

Texture Level-Of-Detail : Mipmaps

```
void glTexImage2D(GLenum target, GLint level,
GLint internalformat, GLsizei width, GLsizei height,
GLint border, GLenum format, GLenum type, const GLvoid *pixels)
```



- Mipmaps are different resolution versions of the same texture image.
- Mipmap level 0 is the largest, most detailed image
- Increasing level by 1 usually decreases width and height by half.

Generating mipmaps

- **Option 1:**
  - You can precompute them yourself and upload each one with glTexImage2D(...)
- **Option 2:**
  - Upload mipmap level 0 using glTexImage2D(...)
  - Call glGenerateMipmap(target)
  - You can add this to the LoadTexture(...) function

Creating a texture

- Upload the texture data from a typed array
  - `glTexImage2D(target, level, iformat, width, height, border, format, type, pixels);`

    - **format** defines how the data is stored in pixels array
    - **format** options
      - GL_RGB
      - GL_RGBA : (A = alpha, for transparency)
      - GL_LUMINANCE: grayscale images
      - GL_LUMINANCE_ALPHA: grayscale with transparency
      - GL_ALPHA: transparency only

Creating a texture

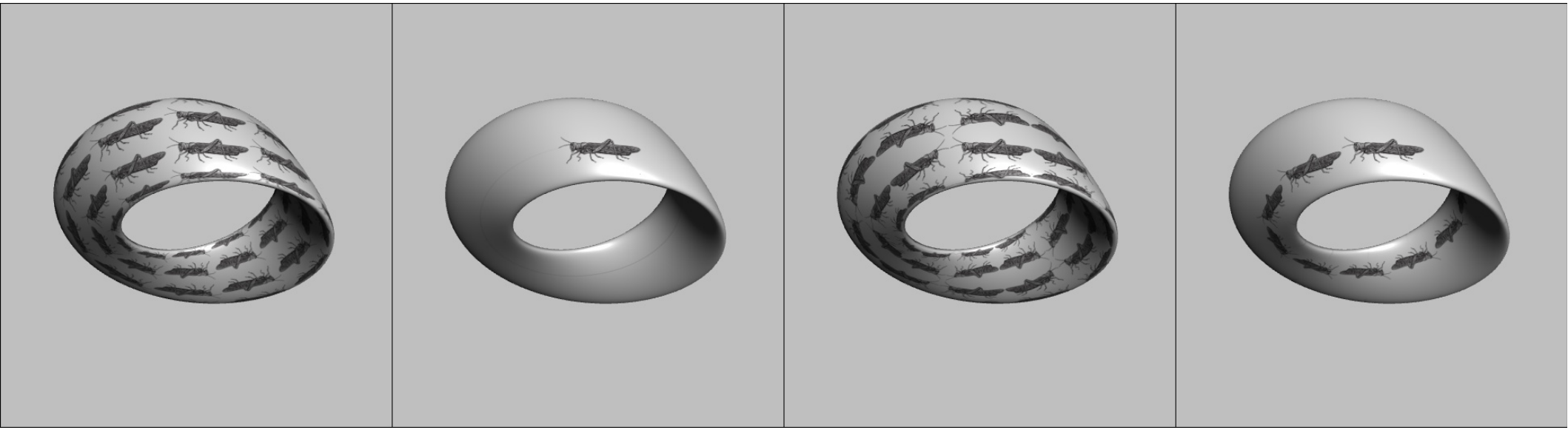
- `glTexImage2D(target, level, `**`iformat`**`, width, height, border, format, type, pixels);`
  - **iformat** defines how the GPU stores pixels
  - **OpenGL** will transform the pixel data from format to iformat, if needed, when uploading to the GPU
  - There are many **iformat** options
    - GL_RGB
    - GL_RGBA : (A = alpha, for transparency)
    - Sized formats, like GL_RGBA32F
    - Compressed formats, like GL_COMPRESSED_RGB

Creating a texture

- Upload the texture data from a typed array
  - `glTexImage2D(target, level, iformat, width, height, border, format, type, pixels);`
    - Most common type and data option
      - **type** = UNSIGNED_BYTE
        - **pixels** must be a unsigned char
    - Other options
      - **type** = UNSIGNED_SHORT_4_4_4_4
      - **type** = UNSIGNED_SHORT_5_6_5
      - **type** = UNSIGNED_SHORT_5_5_5_1
        - **pixels** must be unsigned short

Texture Parameters

- Setting texture parameters
  - Wrapping
    - How to handle tex coords outside [0,1] range
  - Filtering
    - magnification and minification



Textures with different wrap modes

Setting texture parameters

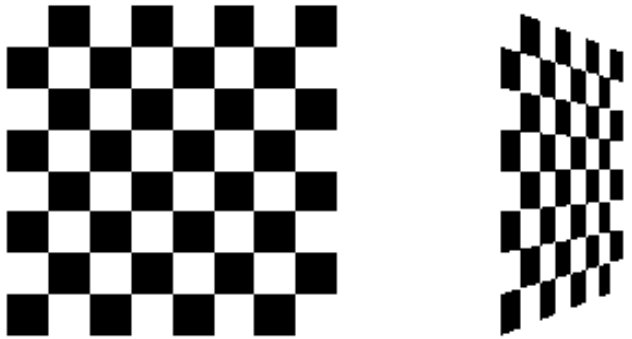
- void **glTexParameteri(target, pname, param);**

  - **pname**: parameter name, e.g. GL_TEXTURE_WRAP_S
  - **param**: parameter value, e.g. GL_REPEAT

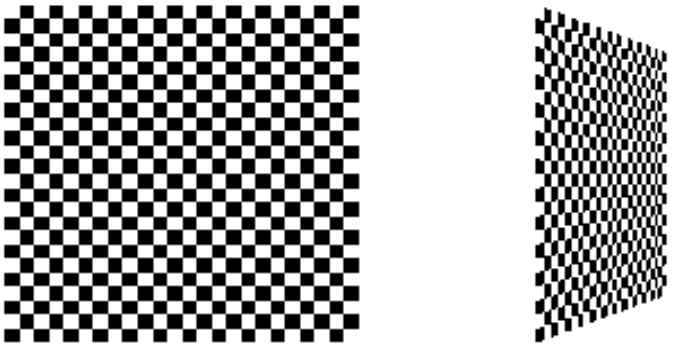
  - **Applies to the currently bound texture**

Texture wrapping

- Wrapping parameters define how OpenGL will handle tex coords outside the range [0, 1]
- Can handle the horizontal and vertical coords differently



Plane with tex coords between 0, 1



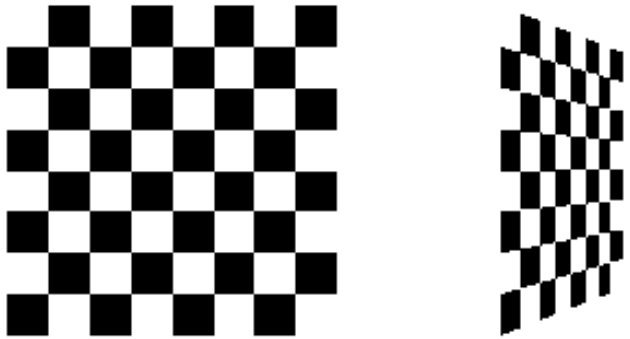
Plane with tex coords between 0, 3

GL_TEXTURE_WRAP_S = GL_REPEAT

GL_TEXTURE_WRAP_T = GL_REPEAT

Texture wrapping

- Wrapping parameters define how OpenGL will handle tex coords outside the range [0, 1]
- Can handle the horizontal and vertical coords differently



Plane with tex coords between 0, 1

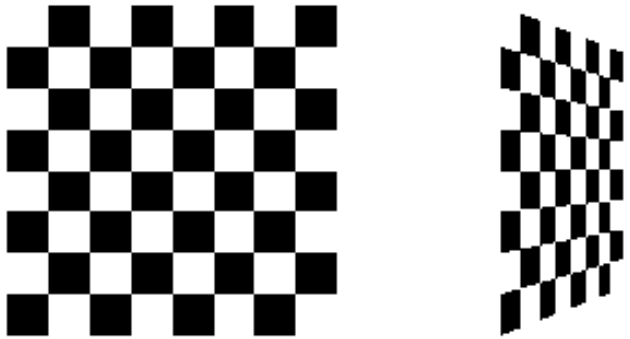
Plane with tex coords between 0, 3

GL_TEXTURE_WRAP_S = GL_CLAMP_TO_EDGE

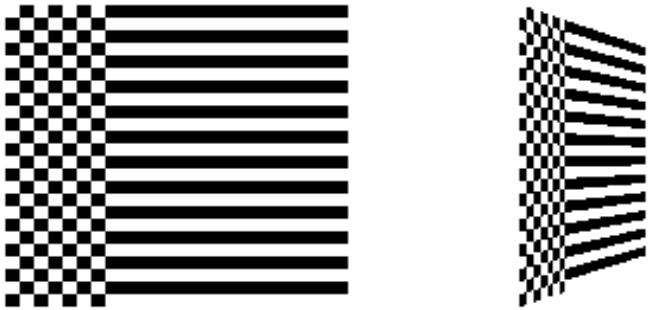
GL_TEXTURE_WRAP_T = GL_CLAMP_TO_EDGE

Texture wrapping

- Wrapping parameters define how OpenGL will handle tex coords outside the range [0, 1]
- Can handle the horizontal and vertical coords differently



Plane with tex coords between 0, 1



Plane with tex coords between 0, 3

GL_TEXTURE_WRAP_S = GL_CLAMP_TO_EDGE

GL_TEXTURE_WRAP_T = GL_REPEAT

Clamping vs. repeating

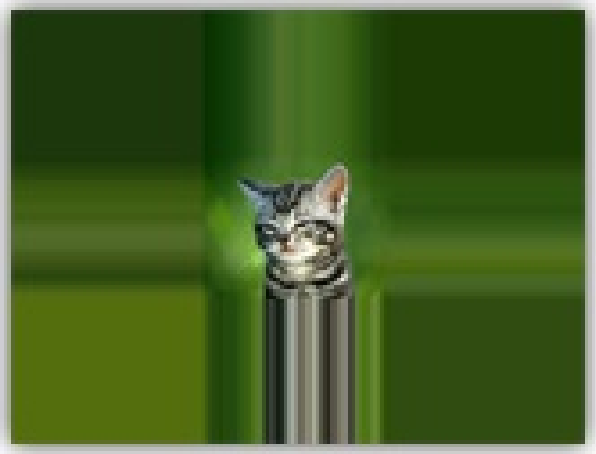
- Clamp
  - If coord < 0 then coord = 0
  - If coord > 1 then coord = 1
- Repeat
  - Ignore the integer part of texture coords
    - E.g. If coord == 1.5 then use coord = 0.5

Texture wrapping

- Another option: GL_MIRRORED_REPEAT

GL_REPEAT

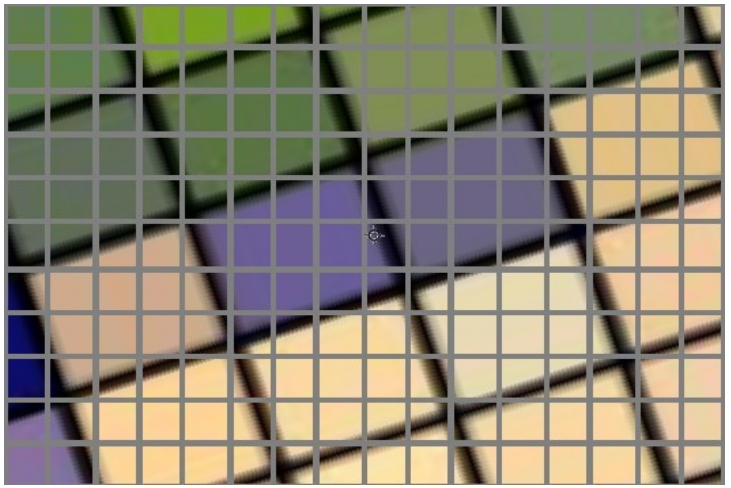
GL_MIRRORED_REPEAT



GL_CLAMP_TO_EDGE

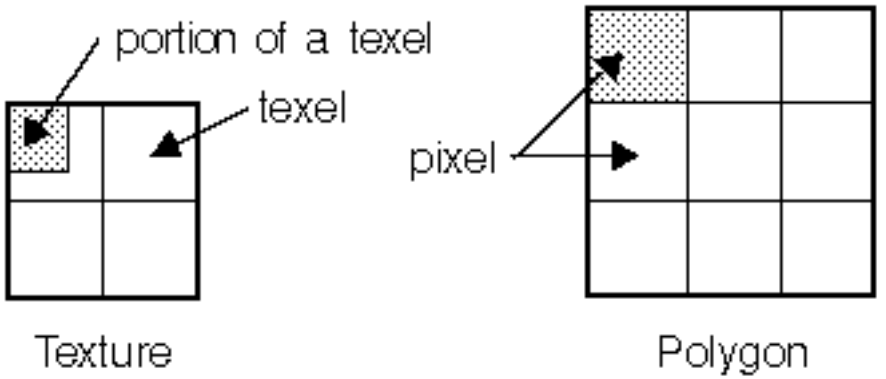
Filtering

- How to handle tex coords that fall between texels?
- How to handle pixel sizes that don't match texel sizes?
  - Sometimes textures are **magnified** in screen space
    - Small texture on large object
  - Sometimes textures are **minified** in screen space
    - Large texture on small object



Filtering

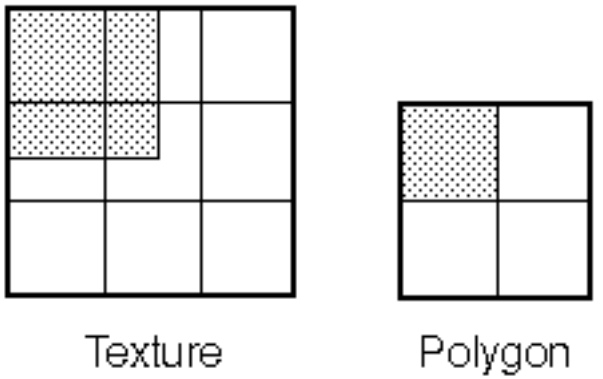
2x2 texture mapped onto 3x3 pixel quad.



Magnification

This case is handled by
GL_TEXTURE_MAG_FILTER

3x3 texture mapped onto 2x2 pixel quad

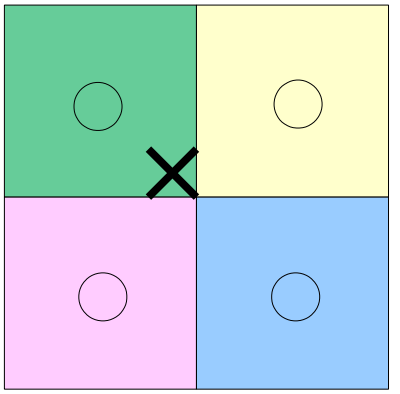


Minification

This case is handled by
GL_TEXTURE_MIN_FILTER

Texture Magnification Options

- **GL_NEAREST**
  - Take nearest texel value
  - Example : result = green



- **GL_LINEAR**
  - Take weighted average of neighbors
  - Example :

$$\text{result} = w_1 * \text{green} + w_2 * \text{yellow} + w_3 * \text{blue} + w_4 * \text{pink}$$

Texture Magnification

GL_NEAREST

GL_LINEAR

Texture Minification

- GL_NEAREST
- GL_LINEAR
- And modes involving ***mipmapping***
  - OpenGL will find the mipmap level that most closely matches the size of the texture on the screen

Texture Minification

- GL_NEAREST
- GL_LINEAR
- And modes involving ***mipmapping***
  - OpenGL will find the mipmap level that most closely matches the size of the texture on the screen

- Ex. GL_LINEAR_MIPMAP_NEAREST

Filtering **within** a mipmap level

Filtering **across** mipmap levels

Texture filtering

(required number of texture reads)

- **Magnification**
  - GL_NEAREST – nearest texel center (1)
  - GL_LINEAR – weighted average of surrounding 4 texels (4)
- **Minification**
  - GL_NEAREST (1), GL_LINEAR (4)
  - GL_NEAREST_MIPMAP_NEAREST (1)
    - Choose mipmap with texel size nearest the pixel size
    - Do nearest neighbor filtering within that mipmap
  - GL_LINEAR_MIPMAP_NEAREST (bilinear) (4)
    - Linear filtering within the nearest mipmap
  - GL_NEAREST_MIPMAP_LINEAR (2)
    - Nearest filtering within two mipmap levels, then average
  - GL_LINEAR_MIPMAP_LINEAR (trilinear) (8)
    - Linear filtering within two mipmap levels, then average

Connecting client code to shader

- If a GLSL vertex or fragment shader declares:
  - uniform sampler2D diffuseTex;
- Then set the texture by:

```
glUseProgram(prog);
glActiveTexture(GL_TEXTURE0); //set texture unit
glBindTexture(GL_TEXTURE_2D, texID); //set texture target
glUniform1i(glGetUniformLocation(prog, “diffuseTex”), 0);
//use texture unit 1 for the next texture...
```

Note that :

- The sampler type in glsl and the texture target in OpenGL match.
- The texture unit selected by the glActiveTexture call and the value the sampler is set to match

Reading texture data in the shader

- Shader variable type is sampler*
  - `uniform sampler2D color_tex;`
- Access using the glsl texture function
  - `vec4 color = texture(sampler2D color_tex, vec2 texcoord);`

Related glsl functions

- texture(sampler, texcoord, **bias**)
  - bias is added to the internally computed mipmap level

- OpenGL refers to mipmap level as texture **LOD** (level-of-detail)
- textureLod(sampler, texcoord, **lod**)
  - Use the specified lod instead of letting OpenGL compute
- texelFetch(sampler, itexcoord, **lod**)
  - Read unfiltered texels from specified mipmap level

Querying textures and mipmaps in shader

- ivec2 textureSize(sampler, **lod**)
  - Returns the size of the given lod level of the texture bound to sampler
- int textureQueryLevels(sampler)
  - Returns the number of mipmap levels in the texture
- int textureQueryLod(sampler, texcoord)
  - Returns the mipmap level that would be sampled at the specified texture coordinate

Texture mapping wrap up

- Important considerations
  - When creating and loading textures
    - format and internal format
  - Setting parameters
    - wrapping and filtering
    - compute mipmaps or not
  - Using in display
    - glActiveTexture / glUniform1i
  - Using in shader
    - Use texture(...) to get filtered texels
      - optional bias parameter
    - Use texelFetch(...) to get unfiltered texels
    - Can query sizes / mipmap levels in shader