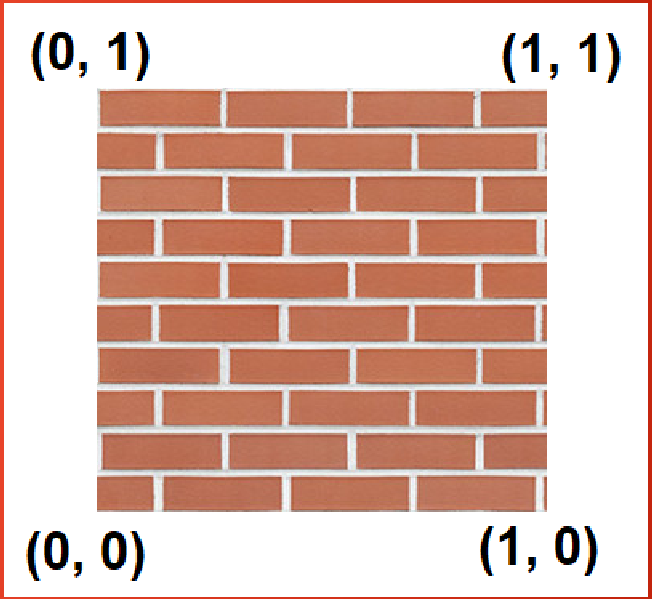
Textures and Image Loading

Textures

- Textures are images used to add extra detail to an object.
- Textures can also be used to hold generic data… but that’s a more advanced topic!
- Usually 2D but can also have 1D and 3D textures.
- Points on textures are “texels”, not pixels.
- Texels are defined between 0 and 1.

Textures

- So to sample a point at the top-middle you reference texel (0.5, 1)
- Map texels to vertices.
- Interpolation over each fragment will calculate appropriate texels in between the assigned texels.



Texture Objects

- Creating textures works much like creating VBOs/VAOs
- glGenTextures(1, &texture);

  glBindTexture(GL_TEXTURE_2D, texture);
- There are different types of texture, such as GL_TEXTURE_1D, GL_TEXTURE_3D and GL_TEXTURE_CUBE_MAP.
- We will deal with the latter, later.

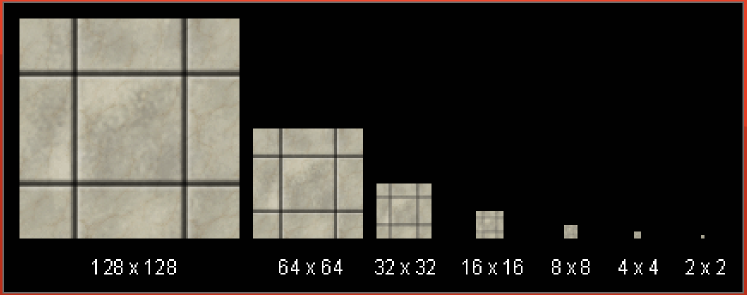
Texture Objects

- `glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height,`
  `                  0, GL_RGB, GL_UNSIGNED_BYTE, data);`

- First Argument: Texture Target (what we have bound our texture to)
- Second Argument: Mipmap Level (see next slide)
- Third Argument: Format of the stored data. RGB is just Red, Green Blue values. Also RGBA which has an Alpha Channel (transparency). Several others, not important here.
- Fourth and Fifth Argument: Width and height of the texture.
- Sixth Argument: This should always be 0. Legacy concept handling texture borders that OpenGL doesn’t utilise anymore.
- Seventh Argument: Format of the data being loaded (as opposed to stored on the third argument).
- Eighth Argument: The data type of the values (int, float, byte, etc).
- Ninth Argument: The data itself.

Mipmaps

- Resolution limitations for textures.
- The closer we get to an object, the more pixelated the texture becomes. Further away, it attempts to render multiple texels on one pixel!
- Solution: Create multiple versions of image at different resolutions and switch between them based on distance.



Texture Parameters - Filters

- What if we try to render off center of texels?
- Two possibilities:
  - Nearest: Use the texel with most overlap (creates a pixelated effect)
  - Linear: Use a weighted average of surrounding texels (blends pixel boundaries)
- Linear more common in most applications.
- Nearest used if you want a pixelated effect (such as a retro game).

Texture Parameters - Filters

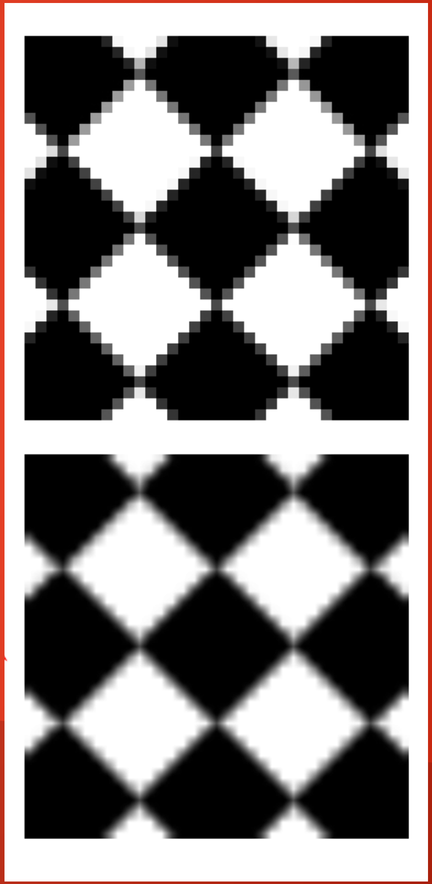
- glTexParameter: used to set texture rendering parameters.
- glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
- GL_TEXTURE_MIN_FILTER: Filter to apply when texture is made smaller (is further away).
- GL_TEXTURE_MAG_FILTER: Filter to apply when texture is made bigger (is closer).

- GL_LINEAR: Linear filter (blends surrounding texels).
- GL_NEAREST: Nearest filter (picks nearest texel to sampling point).

Texture Parameters - Filters

- Top Image: Using Nearest, has a more pixelated look, good for certain visual styles.
- Bottom Image: Using Linear, blends texels together to create a smoother look. Works well on complex textures, less so on simple ones.



Texture Parameters - Wrap

- What if we try to sample a point outside the 0, 1 range?
- Multiple ways to handle it:
  - Repeat the texture.
  - Repeat a mirrored form of the texture.
  - Extend pixels at the edge.
  - Apply a coloured border.
- Can use glTexParameter to define how this is handled.

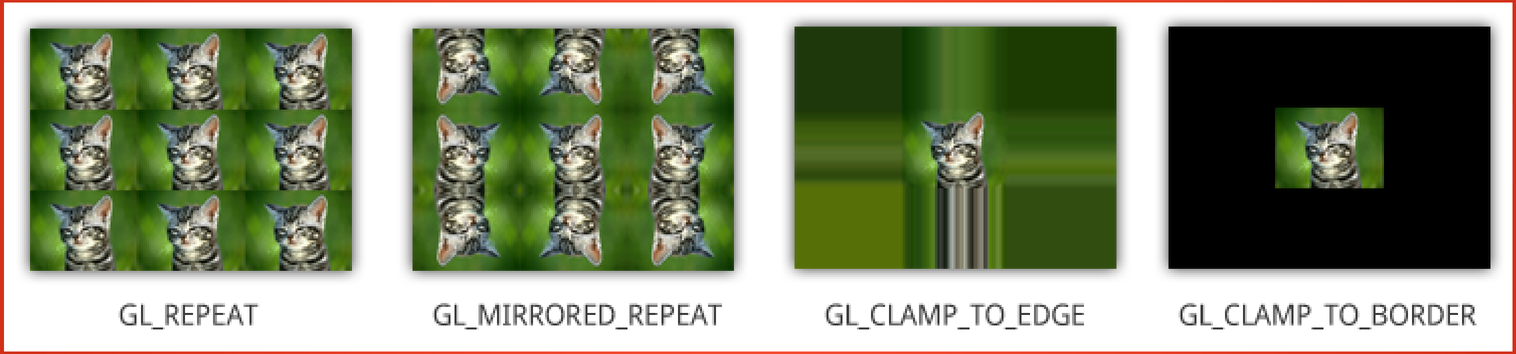
Texture Parameters - Wrap

- glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
- glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

- GL_TEXTURE_WRAP_S: How to handle wrapping on “s-axis” (x-axis).
- GL_TEXTURE_WRAP_T: How to handle wrapping on “t-axis” (y-axis).

- GL_REPEAT: Repeat texture.
- GL_MIRRORED_REPEAT: Repeat and mirror texture.
- GL_CLAMP_TO_EDGE: Extend pixels at edge.
- GL_CLAMP_TO_BORDER: Apply coloured border.

Texture Parameters - Wrap



Loading Images for Textures

- Could write own image loader…
- But will get confusing for handling each image type (bmp, jpg, png, gif, tga, etc…).
- Image Loader libraries do the work for us.
- Popular library: Simple OpenGL Image Library (SOIL).
- For the sake of simplicity, we'll use a smaller library…
- stb_image

Using stb_image

- Only requires the header file, so lightweight.
- Must start project with:

  #define STB_IMAGE_IMPLEMENTATION
- unsigned char *data =

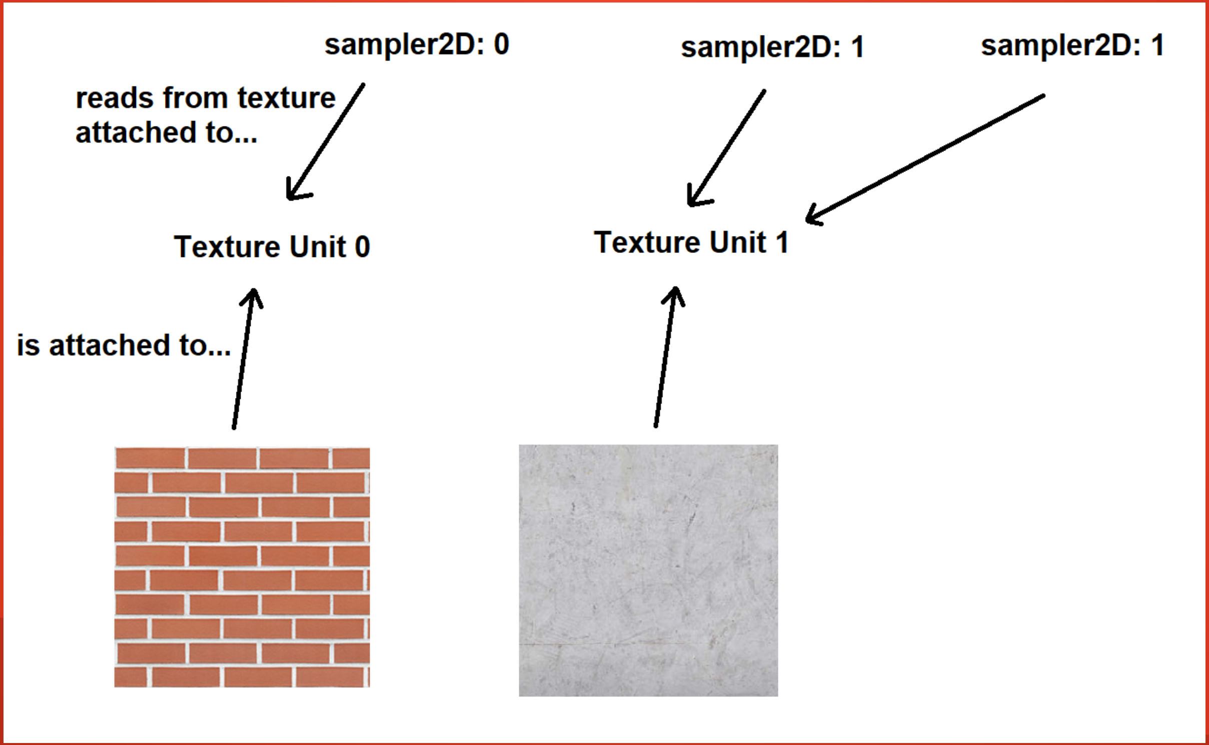
  stbi_load("image.jpg", &width, &height, &bitDepth, 0);

- Might need to flip image.
- stbi_set_flip_vertically_on_load(true);

Texture Samplers

- Textures in Shaders are accessed via “Samplers”.
- Textures are attached to a “Texture Unit”.
- Samplers access textures attached to their Texture Unit.
- In shader, use “sampler2D” type.
- To get the value of a texel, use GLSL “texture” function.
- texture(textureSampler, TexCoord);
- textureSampler: the sampler2D object.
- TexCoord: the interpolated texel co-ordinate in the Fragment Shader.

Texture Units



Texture Units

- Bind texture to desired Texture Unit:

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, textureID);
```

- Ensure sampler2D variables know which Texture Unit to access:

```
glUniform1i(uniformTextureSampler, 0);
```

- Value attached to uniform is the Texture Unit number.

Summary

- Textures use texels between 0 and 1.
- Texels are bound to vertices and values are interpolated.
- Mipmaps handle level-of-detail more efficiently.
- Texture Filtering changes how texels are blended (or ren’t blended) based on size on screen.
- Texture Wrapping changes how textures are handled for texel values outside of the 0, 1 range.
- Wrapping and Filtering are defined using the glTexParameteri function.
- Load images with third-party libraries for convenience.
- SOIL is a popular library for this but stb_image is more lightweight and good for this project.
- Textures attach to Texture Units, samplers read from Textures attached to Texture Units.

See you next video!