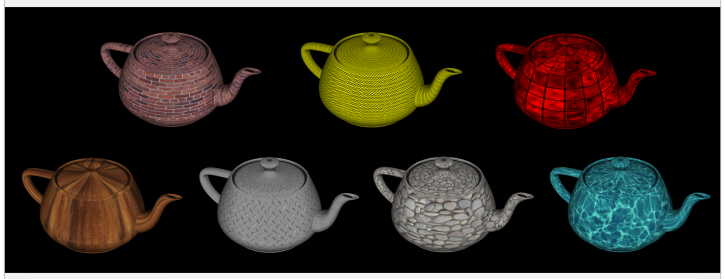
Textures

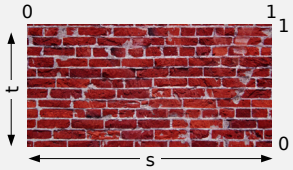
Image Textures

- a *texture* provides a property that varies from pixel to pixel across a surface
- an *image texture* applies to the surface color
  - texture color may replace, modulate, or mix with the material color

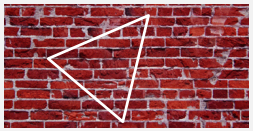


Texture Coordinates

- a texture image has its own local coordinate system
  - `s`, `t` coordinates range from 0 to 1 regardless of the dimensions of the image in pixels

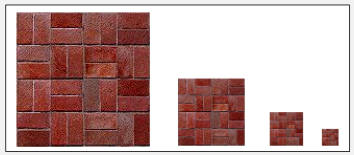


- *texture coordinates* specify how to map the texture onto the surface
  - associated with each vertex of the primitive
  - may be specified as part of the model or generated



Mipmaps

- a *texel* is texture pixel
- typically scene pixels don't match up exactly with texels
  - *nearest texel filtering* – use the color of the nearest texel
  - *linear filtering* – average nearby texels
- linear filtering can be speeded up with *mipmaps*
  - precomputed set of half-scale, quarter-scale, etc images



the mipmap closest in size to the surface is used, then linear filtering needs only a few nearby texels

space requirements are only about 1/3 more than the original texture

Textures in WebGL

- *sampling* is the process of computing a color from an image texture and texture coordinates
- *texture unit* is a hardware component in the GPU which does sampling
- *texture object* is the data structure for a texture
  - includes color data for image texture, mipmaps, properties (minification and magnification filters, texture repeat mode)

Textures in WebGL – Steps

- setting up a texture
  - create texture object
  - associate texture object with a texture unit
  - configure texture object
    - load or generate the texture image
    - set parameters, generate mipmaps
- applying a texture
  - pass information to shaders
    - define or generate texture coordinates
    - pass texture coordinates to vertex shader
    - tell the fragment shader which texture unit(s) to use
  - vertex shader
    - (optionally) apply texture transformation or other manipulation of texture coordinates
    - pass texture coordinates to fragment shader via varying variable
  - fragment shader
    - sample texture to get color
    - (optionally) blend texture color with other colors (e.g. from lighting)

Setting Up a Texture

- create texture object

  `textureObj = gl.createTexture();`
  - allocate memory for texture
- associate texture object with a texture unit

  `gl.activeTexture(gl.TEXTURE`*i*`);`
  - activate texture unit *i*

    numbered 0, 1, 2, ...
    number of available texture units can be determined with `gl.getParameter(gl.MAX_COMBINED_TEXTURE_IMAGE_UNITS)`

  `gl.bindTexture(`*target*`,`*textureObj*`);`
  - associate texture with active texture unit
  - *target* is `gl.TEXTURE_2D`

Setting Up a Texture – Loading Image Data

loads into currently bound texture object

- load from image or `<canvas>` element
  - `gl.texImage2D(`*target*`,0,gl.RGBA,gl.RGBA, gl.UNSIGNED_BYTE,`*image*`)`
    - *target* – `gl.TEXTURE_2D`
    - *mipmap level* – 0 for the main image (generally don't load individual mipmaps)
    - *format* of colors in texture object and in original image – must be the same
      - typically `gl.RGBA` (web images are RGBA) or `gl.RGB` if alpha isn't needed
      - can be `gl.LUMINANCE` or `gl.LUMINANCE_ALPHA` to convert image to grayscale in a way that reflects perceived brightness
    - *data type* – `gl.UNSIGNED_BYTE` (one byte per color component) for web images
    - *image* – DOM image element or `<canvas>` element
      - must be power-of-two size or else no mipmaps (must change default minification filter) and only repeat option is `CLAMP_TO_EDGE`
  - notes
    - images are loaded asynchronously – must set a callback to call `texImage2D` and draw scene when loading is complete
    - also need to flip image – WebGL assumes bottom row first, web images are top row first
    - can also change minification filter and/or generate mipmaps

Setting Up a Texture – Loading Image Data

- load from an array of numbers specifying color component values

loads into currently bound texture object

  - `gl.texImage2D(`*`target`*`,0,gl.RGBA,`*`width`*`,`*`height`*`,`*`border`*`,`
    `gl.RGBA,gl.UNSIGNED_BYTE,`*`dataArray`*`)`
    - *target* – `gl.TEXTURE_2D`
    - *mipmap level* – 0 for the main image
    - *format* of colors in texture object and in *dataArray* – must be the same
      - same options as for loading from an image
    - *width*, *height* – dimensions of texture image
    - *data type* for color data – `gl.UNSIGNED_BYTE` (one byte per color component) to match *dataArray*
    - *dataArray* – typed array of type `Uint8Array` or `Uint16Array`, to match data format of texture
      - `Uint8Array` means color component values 0-255, RGBA needs four color components per pixel
      - length of array is 4*width*height; row-major order, with bottom row first
  - notes
    - can also change minification filter and/or generate mipmaps

Setting Up a Texture – Flip Image

- flip the image
  - `gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL,1);`
    - affects images loaded after the property is set

Setting Up a Texture – Configure Texture Object

- generate mipmaps
  - `gl.generateMipmap(`*`target`*`)`
    - *target* is `gl.TEXTURE_2D`
    - requires image size to be a power of two

Setting Up a Texture – Configure Texture Object

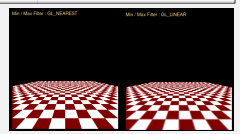
- set parameters for texture objects
  - `gl.texParameter(`*`target`*`,`*`property`*`,`*`value`*`)`
    - *target* is `gl.TEXTURE_2D`
    - *property*, *value*

minification, magnification filters address how to match scene pixels to texels

`gl.TEXTURE_MAG_FILTER`	Texture magnification filter	`gl.LINEAR` (default value), `gl.NEAREST`.
`gl.TEXTURE_MIN_FILTER`	Texture minification filter	`gl.LINEAR`, `gl.NEAREST`, `gl.NEAREST_MIPMAP_NEAREST`, `gl.LINEAR_MIPMAP_NEAREST`, `gl.NEAREST_MIPMAP_LINEAR` (default value), `gl.LINEAR_MIPMAP_LINEAR`.

`gl.LINEAR` magnification can cause blending (blurring) along distinct edges, especially for generated textures – use `gl.NEAREST` instead

`gl.a_MIPMAP_b` – *a* is how to locate the mipmap to use, *b* is used within a mipmap (note that the default uses mipmaps – must be changed if mipmaps aren't being used!)



non-power-of-two textures are limited to LINEAR or NEAREST

https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/texParameter
https://learnopengl.com/Getting-started/Textures
https://www.flipcode.com/archives/Advanced_OpenGL_Texture_Mapping.shtml

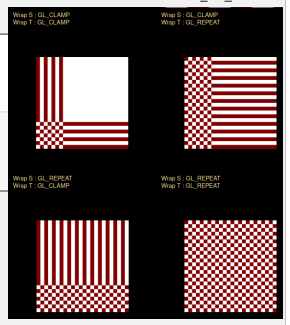
Setting Up a Texture – Configure Texture Object

- set parameters for texture objects
  - `gl.texParameter(target,property,value)`
    - *target* is `gl.TEXTURE_2D`
    - *property*, *value*

wrap addresses how to handle texture coordinates outside the range 0-1

non-power-of-two textures are limited to CLAMP_TO_EDGE

`gl.TEXTURE_WRAP_S`	Wrapping function for texture coordinate `s`	`gl.REPEAT` (default value), `gl.CLAMP_TO_EDGE`, `gl.MIRRORED_REPEAT`.
`gl.TEXTURE_WRAP_T`	Wrapping function for texture coordinate `t`	`gl.REPEAT` (default value), `gl.CLAMP_TO_EDGE`, `gl.MIRRORED_REPEAT`.



GL_REPEAT GL_MIRRORED_REPEAT GL_CLAMP_TO_EDGE

https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/texParameter
https://learnopengl.com/Getting-started/Textures
https://www.flipcode.com/archives/Advanced_OpenGL_Texture_Mapping.shtml