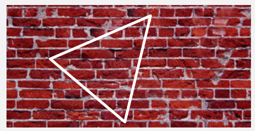
Image Textures

- an image texture is specified as an image (or equivalent)
- *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
- *sample* the texture at the point corresponding to each surface pixel to determine the pixel's color



Textures in WebGL – Steps Overview

- setting up the texture
  - create the texture object
  - configure texture object
    - bind texture object
    - load or generate the texture image
    - set parameters, generate mipmaps

parameters: minification filter, magnification filter, wrapping function

- applying the texture
  - associate texture object with a texture unit
  - pass information to shaders
    - tell the fragment shader which texture unit(s) to use
- defining shaders
  - fragment shader – determine color of pixel
  - vertex shader – pass that which is interpolated rather than computed per-pixel to the fragment shader

Textures in WebGL – Steps Recap

- setting up the texture
  - create the texture object

```
textureObj = gl.createTexture();
```

  - configure texture object
    - bind texture object so subsequent operations apply to it

```
gl.bindTexture(gl.TEXTURE_2D,textureObj);
```

    - load or generate the texture image
      - WebGL expects data bottom up, but web images are top down – must specify that images should be flipped when loaded

```
gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL,1);
```

```
gl.texImage2D(gl.TEXTURE_2D,0,gl.RGBA,gl.RGBA,gl.UNSIGNED_BYTE,image);
gl.texImage2D(gl.TEXTURE_2D,0,gl.RGBA,width,height,border,gl.RGBA,
              gl.UNSIGNED_BYTE,dataArray);
```

    - set parameters, generate mipmaps

```
gl.texParameter(gl.TEXTURE_2D,property,value);
```

      - *property*: gl.TEXTURE_MAG_FILTER, gl.TEXTURE_MIN_FILTER, gl.TEXTURE_WRAP_S, gl.TEXTURE_WRAP_T

```
gl.generateMipmap(gl.TEXTURE_2D);
```

      - requires texture dimensions to be power of two

Textures in WebGL – Steps Recap

- applying the texture
  - associate texture object with a texture unit
    - activate texture unit

```
gl.activeTexture(gl.TEXTUREi);
```

    - bind texture object to currently active texture unit

```
gl.bindTexture(gl.TEXTURE_2D,textureObj);
```

Textures in WebGL – Steps

- defining shaders
  - fragment shader – determine color of pixel
    - obtain texture coordinates for pixel – passed from vertex shader via varying variable or computed directly
    - (optionally) apply texture transformation or other manipulation of texture coordinates
    - sample texture to get color
    - (optionally) blend texture color with other colors (e.g. from lighting)
  - vertex shader – pass that which is interpolated rather than computed per-pixel to the fragment shader
    - for texture coordinates supplied as part of the model geometry –
      - (optionally) apply texture transformation or other manipulation of texture coordinates
      - pass texture coordinates to fragment shader via varying variable

Textures in Shaders

set values for shader attributes and uniforms in JavaScript

- fragment shader
  - texture unit(s) to use are specified with `uniform` sampler variable(s)
    - type `sampler2D`
    - values are 0, 1, 2, …, `gl.MAX_COMBINED_TEXTURE_IMAGE_UNITS-1`
  - obtain texture coordinates for pixel
    - compute directly and (optionally) apply texture transform or other manipulations, or get from vertex shader via a varying variable
  - sample texture to get color – `texture2D` function
  - (optionally) combine texture color with something else

utilizes texture for pixel color

```
<script type="x-shader/x-fragment" id="fshader">
  precision mediump float;
  uniform sampler2D u_texture;
  varying vec2 v_texcoords;

  void main() {
      vec4 texcolor = texture2D(u_texture,v_texcoords);
      gl_FragColor = texcolor;
  }
</script>
```

- vertex shader
  - for texture coordinates supplied as part of the model geometry –
    - texture coordinates are an attribute of type `vec2`
    - (optionally) apply texture transformation to texture coordinates
    - pass texture coordinates to fragment shader (varying variable)

passes values that are interpolated rather than computed per-pixel to the fragment shader

```
<script type="x-shader/x-vertex" id="vshader">
  attribute vec2 a_coords;
  attribute vec2 a_texcoords;
  varying vec2 v_texcoords;

  void main() {
      gl_Position = vec4(a_coords,0,1);
      v_texcoords = a_texcoords;
  }
</script>
```


Applying the Texture

- set the values for shader attributes and uniforms in JavaScript

```
gl.uniform1i(u_texture_loc, texunit);  // set sampler var
```

texunit is an integer 0, 1, 2, …
specifying which texture unit

```
<script type="x-shader/x-vertex" id="vshader">
  attribute vec2 a_coords;
  attribute vec2 a_texcoords;
  varying vec2 v_texcoords;

  void main() {
      gl_Position = vec4(a_coords,0,1);
      v_texcoords = a_texcoords;
  }
</script>
```

```
<script type="x-shader/x-fragment" id="fshader">
  precision mediump float;
  uniform sampler2D u_texture;
  varying vec2 v_texcoords;

  void main() {
      vec4 texcolor = texture2D(u_texture,v_texcoords);
      gl_FragColor = texcolor;
  }
</script>
```


Usage Patterns

- usage patterns for working with multiple textures
  - single texture object, single texture unit
    - to use a new texture, load a new image into the texture object
    - inefficient
  - different texture objects for each texture, single texture unit
    - to use a new texture, use `gl.bindTexture` to bind a new texture object to the active texture unit
  - different texture objects for each texture, different texture units
    - bind textures to different texture units
    - to use a new texture, pass a different value for the sampler variable to the fragment shader
    - necessary if more than one texture is to be applied to the same primitive (use multiple sampler variables)

Textures + Lighting

```
<script type="x-shader/x-fragment" id="fshader">
  precision mediump float;
  uniform sampler2D u_texture;
  varying vec2 v_texcoords;

  void main() {
      vec4 texcolor = texture2D(u_texture,v_texcoords);
      gl_FragColor = texcolor;
  }
</script>
```

- using only the texture color ignores the lighting in the scene
- other options
  - *mix* – combine the texture color and the lighting equation color
    - can use GLSL `mix` function `mix(x,y,t) = x*(1-t) + y*t`
  - *replace* – use the texture color in place of the object's ambient and diffuse colors in the lighting equation
    - appropriate for full-color textures

$$I = ma\, I_a + \sum_{all\,lights} \left[md\, I_s max\left(0,(N \cdot L)\right) + ms\, I_s max\left(0,(R \cdot V)\right)^{mh} \right]$$

  - *modulate* – texture color multiplies the ambient and diffuse terms
    - appropriate for grayscale textures
  - replace, modulate require lighting to be done in the fragment shader – texture color is a per-pixel operation

Texture Transforms

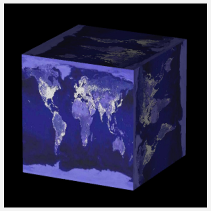
- OpenGL supports a current texture transform along with modelview and projection matrices
- with WebGL –
  - maintain a texture transform (JavaScript variable)
    - `mat3` because texture coordinates are 2D
  - for texture coordinates defined as part of the model geometry –
    - pass texture transform to vertex shader just like modelview and projection matrices
    - vertex shader applies transform to the texture coordinates it is provided

```
vec3 texcoords = u_textureTransform*vec3(a_texcoords,1.0);
v_texcoords = texcoords.xy;
```

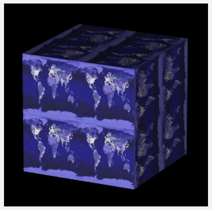
      - `a_texcoords`, `u_textureTransform` are shader parameters
      - `v_texcoords` is a varying parameter
  - for generated texture coordinates –
    - pass texture transform to fragment shader
    - fragment shader computes texture coordinates and applies transform

Texture Transforms

- the effect on the appearance of the texture is the inverse of the transformations specified
  - e.g. `scale(2,2)` makes it appear as if the texture has shrunk by a factor of 2



scale factor 1
texture coordinates for the front face of the cube are (0,0), (1,0), (1,1), (0,1)



scale factor 2
texture coordinates for the front face of the cube are transformed to (0,0), (2,0), (2,2), (0,2)

Generating Texture Coordinates

- texture coordinates may not be supplied as part of the object
- complex objects can be difficult to determine texture coordinates for
- there are many different ways to generate texture coordinates
  - some work better than others for certain kinds of shapes
- texture coordinates are generally computed in OC so texture sticks with the object
- can be computed in the vertex shader if linear interpolation is appropriate, otherwise compute in fragment shader

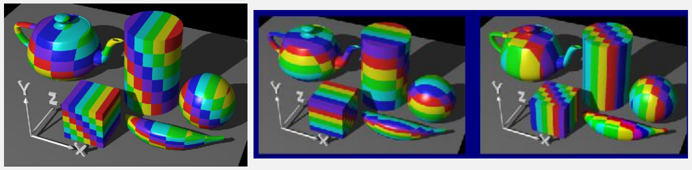
Generating Texture Coordinates

Strategies –

- projection
- shrinkwrapping
- intermediate (map) shapes

Projection – Plane

- projection onto a plane
  - take xy, yz, or xz part of OC point

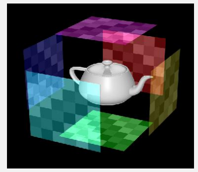


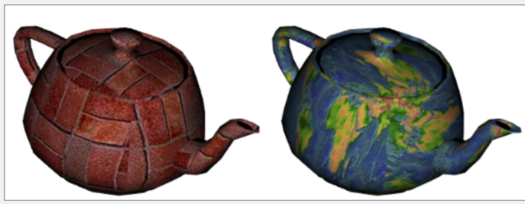
  - suitable for faces more or less parallel to projection plane
  - very poor for faces perpendicular to projection plane

http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/

Projection – Cube

- cubical projection
  - use plane perpendicular to the component of the surface normal with the greatest magnitude (i.e. axis-aligned plane closest to parallel to the surface)
    - flip components when projecting along a negative axis to avoid having mirror-reversed texture on surface
    - can be done in the vertex shader for flat shading (polygon normals) but should be done in the fragment shader for smooth shading (vertex normals)





- good for cubes
- often good for other shapes, but with seams

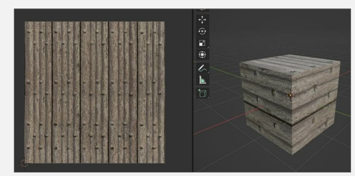
Shrinkwrapping

- directly map the surface to the image texture
- suitable for simple shapes, such as
  - cube
  - sphere
  - cylinder
  - infinite cylinder

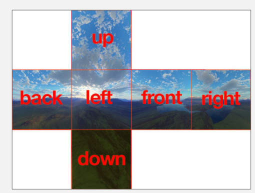
https://www.bigshrink.com/bulk-shrink-wrap-32-100-7

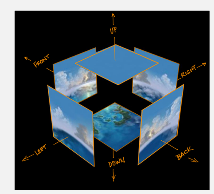
Shrinkwrapping Cubes

- apply image to each face using plane projection



- use a cubemap

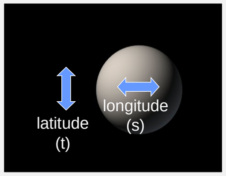




https://i.all3dp.com/workers/images/fit=scale-down,w=1200,h=630,gravity=0.5x0.5,format=jpeg/wp-content/uploads/2023/03/20181630/cube-mesh-with-medieval-wooden-texture-aftab-ali-via-all3dp-230123.jpg
https://en.wikipedia.org/wiki/Cube_mapping
https://scalibq.wordpress.com/2013/06/23/cubemaps/

Shrinkwrapping Spheres

x = r cos(lat) sin(long)
y = r sin(lat)
z = r cos(lat) cos(long)



latitude is between –90 and 90
longitude is between –180 and 180

- convert (x,y,z) to (long,lat)
- map long → s, lat → t

https://i.sstatic.net/oX2gX.jpg