TEXTURE MAPPING
TEXTURE MAPPING

- real life objects have nonuniform colors, normals
- to generate realistic objects, reproduce coloring & normal variations = texture
- can often replace complex geometric details
TEXTURE MAPPING

• hide geometric simplicity
  • images convey illusion of geometry
  • map a brick wall texture on a flat polygon
  • create bumpy effect on surface

• usually: 2D information associated with a 3D surface
  • point on 3D surface ↔ point in 2D texture
  • typically r,g,b colors
  • but can be any attributes that you would like to model over a surface
BUMP MAPS

2D texture maps that are used to model the appearance of surface bumps, by adding small perturbations to the surface normals. The rendered geometry does not actually have bumps, i.e., it is smooth!!

threejs.org: materials/bumpmap
VOLUMETRIC TEXTURES

- model r,g,b for every point in a volume
- often computed using procedural function

[Lapped Solid Textures, SIGGRAPH 2008]
ENVIRONMENT MAP

There is an invisible corner seam in this image!

2 of 6 images for a cube map; as a viewer, you are inside this cube!
Texture coords: \((u, v)\) or \((s, t)\)

**BASIC TEXTURE MAP**

3D model: u,v texture coordinates are assigned to vertices by artist or program.

2D texture map: Image Pixels here are called “texels”.

Interpolate \((u,v)\) from vertices using barycentric coordinates.
TEXTURE MAPPING EXAMPLE

Assigning texture coordinates to the 3D model vertices:
(a) by hand, or by "projecting" the texture onto the object
(b) automatically

Creating the texture map image:
(a) photograph or painted image
(b) painting the 3D object, and then creating a texture atlas.
TEXTURE LOOKUP: TILING AND CLAMPING

• What if s or t is outside [0…1]?
• Multiple choices, e.g.:
  • tex1.wrapS = THREE.RepeatWrapping
  • tex1.wrapS = THREE.ClampToEdgeWrapping
  • tex1.wrapS = THREE.MirroredRepeatWrapping

\[ (\xi, \tau) \]

\[ \begin{align*}
  p_1 & : (0,0) \\
  p_2 & : (2.2, 0) \\
  p_3 & : (2.2, 2.2) \\
  p_4 & : (0, 2.2)
\end{align*} \]
TEXTURES: VERTEX SHADER & FRAGMENT SHADER

• javascript: texture is passed as a “uniform” to the fragment shader:
  (slightly more complex than this due to async image load in js)

```javascript
var myTexture = new THREE.TextureLoader().load( 'textures/crate.gif' );
myTexture.wrapS = THREE.RepeatWrapping;
var material = new THREE.MeshBasicMaterial( { map: myTexture } );
```

• vertex shader

```glsl
attribute vec2 uv;
varying vec2 uvCoords;
uvCoords = uv;
```

• Fragment Shader:

```glsl
uniform sampler2D myTexture;
varying vec2 uvCoords;
vec4 texColor = texture2D(myTexture, uvCoords);
gl_FragColor = texColor;
```
RECONSTRUCTION

- how to deal with:
  - larger pixels that are much larger than texels?
  - minification
    - THREE.NearestFilter
    - THREE.LinearFilter
  - smaller pixels that are much smaller than texels?
  - magnification
    - THREE.NearestFilter
    - THREE.NearestMipMapNearestFilter
    - THREE.NearestMipMapLinearFilter
    - THREE.LinearFilter
    - THREE.LinearMipMapNearestFilter
    - THREE.LinearMipMapLinearFilter

"best" mode.
MIPMAPPING

use “image pyramid” to precompute averaged versions of the texture

For a given pixel with textal area $A$, e.g. $A=10$, then do lookups in the two nearest mipmaps levels and then interpolate.
MIPMAPS

- **multum in parvo** -- many things in a small place
  - prespecify a series of prefILTERed texture maps of decreasing resolutions
  - requires more texture storage
  - avoid shimmering and flashing as objects move

**E.g.:**

```javascript
texture.magFilter = THREE.NearestFilter;
texture.minFilter = THREE.LinearMipMapLinearFilter;
```

Extra memory needed for a Mipmap:

\[ 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots = \sum \frac{1}{2^k} = \frac{1}{1 - \frac{1}{2}} = 2 \]

\[ 1 + a + a^2 + \ldots = \frac{1}{1-a} \]

\( a = \frac{4}{3} \Rightarrow \frac{1}{1-a} = \frac{3}{4} = \frac{4}{3} \)
BUMP MAPPING: NORMALS AS TEXTURE

- object surface often not smooth – to recreate correctly need complex geometry model
- can control shape “effect” by locally perturbing surface normal
  - random perturbation
  - directional change over region

Spherical geometry (triangles)  “bump map”
gray-scale image  Using bump-map to modify normals on a per-pixel basis.
BUMP MAPPING

Virtual surface created with the bump map

\[ N'(u) = P(u) + b(u) \hat{N}(u) \]

Compute \( N'(u) \) using finite difference and a cross product.

Original surface

bump map

normals corresponding to this virtual surface

Note: we could also provide \( N'(u) \): This is a normal map

Render surface \( P(u), N'(u) \)
Normal/Bump mapping

(a) 4M faces
(b) 8K faces
(c) 8K faces, normal-mapped
(d) normal-map

Achieve the appearance of high-resolution geometry using lower resolution geometry + bump map
BUMP MAPPING: LIMITATION

bump mapped

displacement map

realistic silhouette

beyond scope of class; usually done in the "tessellation shader"
DISPLACEMENT MAPPING

- bump mapping gets silhouettes wrong
  - shadows wrong too

- change surface geometry instead
  - need to subdivide surface
  - use tesselation shader

https://en.wikipedia.org/wiki/Displacement_mapping#media/
ENVIRONMENT MAPPING

• generate image of surrounding or reflection
• sphere map or cube map

these are two common types of environment maps.
CUBE MAP

- 6 planar textures, sides of cube
  - point camera in 6 different directions, facing out from origin
• Cube map: direction of vector selects the face of the cube to be indexed
  • co-ordinate with largest magnitude
    • e.g., the vector (-0.2, 0.5, -0.84) selects the –Z face
    • remaining two coordinates select the pixel from the face.
SPHERE MAP

- texture is distorted fish-eye view
  - point camera at mirrored sphere
  - spherical texture mapping creates texture coordinates that correctly index into this texture map
VOLUMETRIC TEXTURE

• define texture pattern over 3D domain - 3D space containing the object
• texture function can be digitized or procedural
• for each point on object compute texture from point location in space
• e.g., ShaderToy

• computation often cheaper than memory access
PROCEDURAL TEXTURES: PERLIN NOISE

• several good explanations
  • http://www.noisemachine.com/talk1
  • http://freespace.virgin.net/hugo.elias/models/m_perlin.htm
  • http://www.robo-murito.net/code/perlin-noise-math-faq.html

http://mrl.nyu.edu/~perlin/planet/