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!!
VOLUMETRIC TEXTURES

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

[Lapped Solid Textures, SIGGRAPH 2008]
ENVIRONMENT MAP

2 of 6 images for a cube map; as a viewer, you are inside this cube!

There is an invisible corner seam in this image!
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”

rendered image

 interpolate (u,v) from vertices using barycentric coordinates
TEXTURE MAPPING EXAMPLE
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
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:
  • pixels that are much larger than texels?
    • minification
      THREE.NearestFilter
      THREE.NearestMipMapNearestFilter
      THREE.NearestMipMapLinearFilter
      THREE.LinearFilter
      THREE.LinearMipMapNearestFilter
      THREE.LinearMipMapLinearFilter
  • pixels that are much smaller than texels?
    • magnification
      THREE.NearestFilter
      THREE.LinearFilter
MIPMAPPING

use "image pyramid" to precompute averaged versions of the texture

Idea: Precompute and store results that can approximate the desired sums/integrals/averages.
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;
```

without

with
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

Bump: gray-scale image that virtually offsets the surface in the direction $\hat{N}$
BUMP MAPPING

- Can compute using cross-product of vectors from finite differences (specific details not covered in this class)

Original surface

A bump map

virtual surface created with the bump map

normals corresponding to this virtual surface
Normal/Bump mapping

For every point on a surface, much like a texture map defines a color for each point:

\[ \vec{N} \in \mathbb{R}^3 \]

1. Offset along \( \vec{N} \)

\( \vec{N} \) is a surface normal.

\( \vec{T}, \vec{B}, \vec{N} \)

One detail we will skip: the most common form of normal maps define normals in a surface-relative coordinate frame, defined by \( (\vec{T}, \vec{B}, \vec{N}) \) where \( \vec{N} = \) surface normal, \( \vec{T} = \) tangent, \( \vec{B} = \) bitangent.
BUMP MAPPING: LIMITATION

- Changed geometry in terrelation shader

Bump map smooth silhouette

Displacement map rough silhouette
DISPLACEMENT MAPPING

- bump mapping gets silhouettes wrong
  - shadows wrong too

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

ENVIRONMENT MAPPING

- generate image of surrounding or reflection
- sphere map or cube map
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/