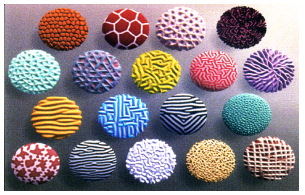
TEXTURE MAPPING

TEXTURE MAPPING

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

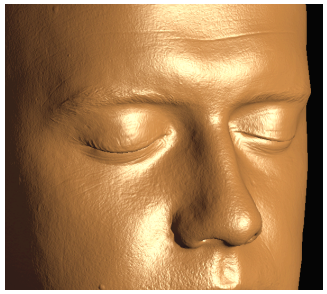


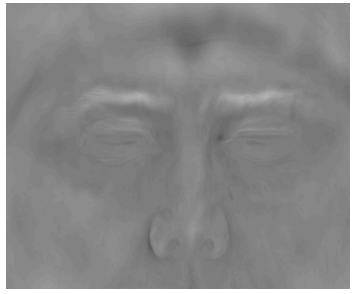
TEXTURE MAPPING

- 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 in2D 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 !!

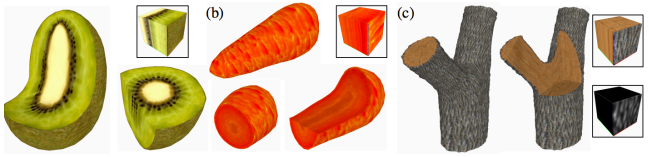




threejs.org: materials/bumpmap

VOLUMETRIC TEXTURES

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



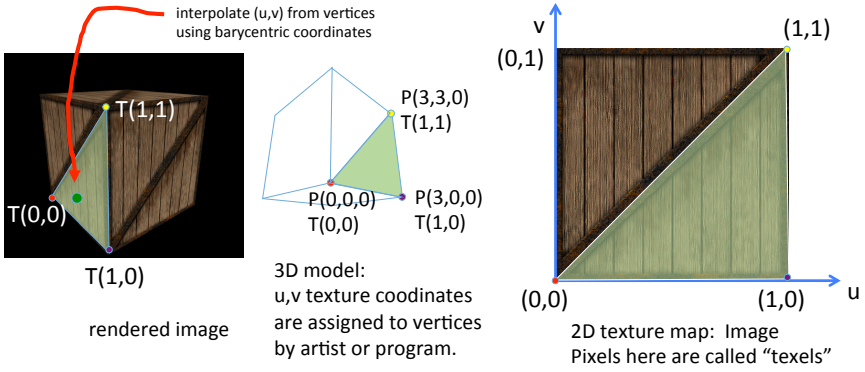
[Lapped Solid Textures, SIGGRAPH 2008]

ENVIRONMENT MAP

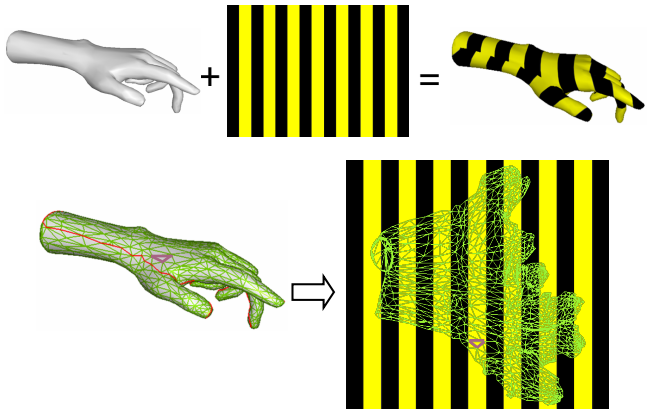
There is an invisible corner seam in this image!

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

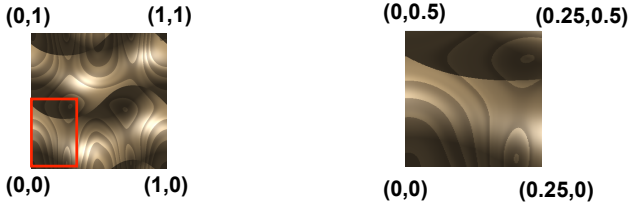
BASIC TEXTURE MAP



TEXTURE MAPPING EXAMPLE



FRACTIONAL TEXTURE COORDINATES



THREE.JS

- pass texture as a uniform:

```
var uniforms = {
    texture1: { type: "t", value: THREE.ImageUtils.loadTexture( "texture.jpg" ) }};
var material = new THREE.ShaderMaterial( { uniforms, …});
```

- uv will be passed on to the vertex shader *(no need to write this)*:

```
attribute vec2 uv;
```

- use it, e.g., in Fragment Shader:

```
uniform sampler2D texture1;
varying vec2 texCoord;
vec4 texColor = texture2D(texture1, texCoord);
```

HOW TO USE COLOR TEXTURES

- Replace
  - Set fragment color to texture color

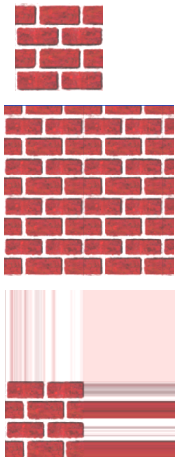
```
gl_FragColor = texColor;
```

- Modulate
  - Use texture color as reflection color in illumination equation

```
kd = texColor; ka = texColor;
gl_FragColor = ka*ia + kd*id*dotProduct + …;
```

TEXTURE LOOKUP: TILING AND CLAMPING

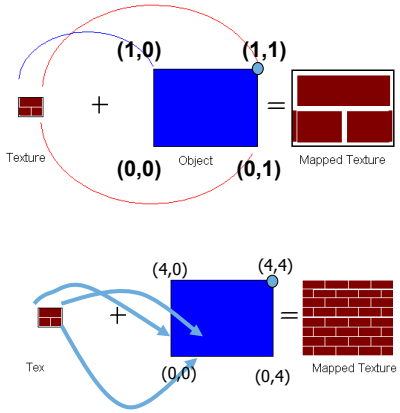
- What if s or t is outside [0…1] ?
- Multiple choices
  - Use fractional part of texture coordinates
    - Cyclic repetition (*repeat*)
  - Clamp every component to range [0…1]
    - Re-use color values from texture image border



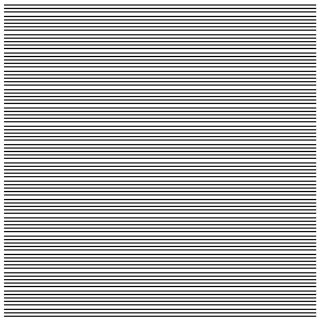
IN THREE.JS

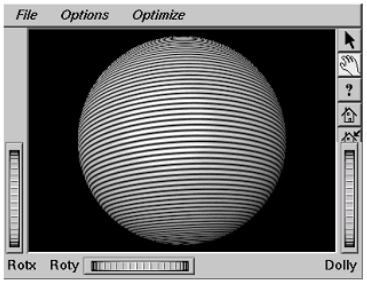
```
var texture = THREE.ImageUtils.loadTexture( "textures/
water.jpg" );
texture.wrapS = THREE.RepeatWrapping;
texture.wrapT = THREE.ClampToEdgeWrapping;
texture.repeat.set( 4, 4 );
```

TILED TEXTURE MAP



RECONSTRUCTION

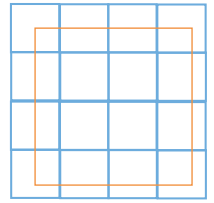


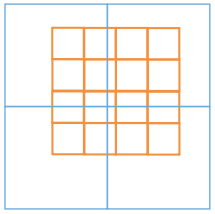


(image courtesy of Kiriakos Kutulakos, U Rochester)

RECONSTRUCTION

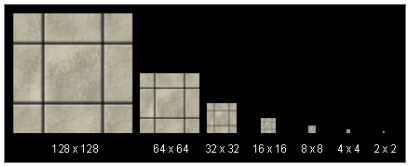
- how to deal with:
  - pixels that are much larger than texels?
    - minification
  - pixels that are much smaller than texels ?
    - magnification



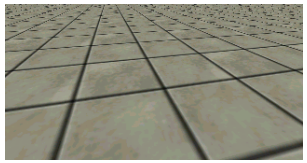


MIPMAPPING

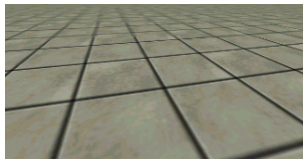
use “image pyramid” to precompute averaged versions of the texture



store whole pyramid in single block of memory



Without MIP-mapping

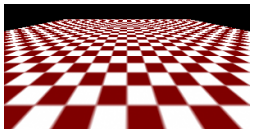


With MIP-mapping

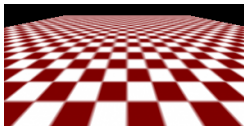
MIPMAPS

- **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
- `texture.generateMipmaps = true`
  - automatically constructs a family of textures from original texture size down to 1x1
- `texture.mipmaps[…]`

without



with

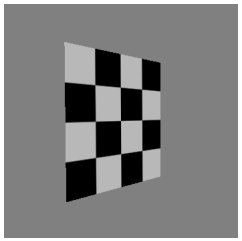


MIPMAP STORAGE

- only 1/3 more space required

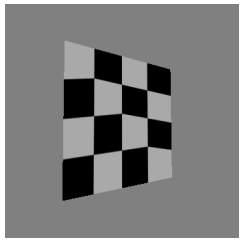
HOW TO INTERPOLATE S,T?

TEXTURE COORDINATE INTERPOLATION



$$u = \frac{\alpha \cdot u_1 / h_1 + \beta \cdot u_2 / h_2 + \gamma \cdot u_3 / h_3}{\alpha / h_1 + \beta / h_2 + \gamma / h_3}$$

Perspective correct interpolation
(see Scan Conversion notes)

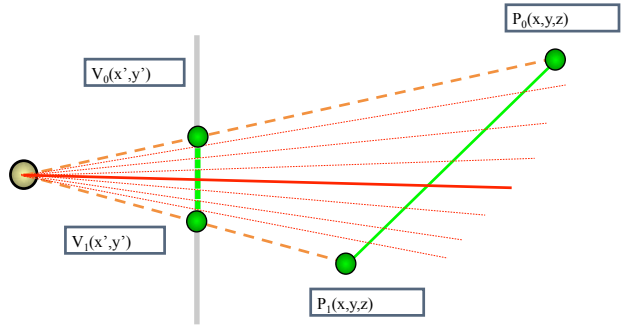


$$u = \alpha \cdot u_1 + \beta \cdot u_2 + \gamma \cdot u_3$$

Linear interpolation
i.e., using barycentric coordinates

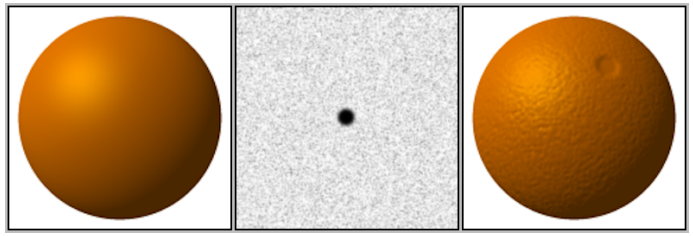
INTERPOLATION: SCREEN VS. WORLD SPACE

- Screen space interpolation incorrect under perspective
  - Problem ignored with shading, but artifacts more visible with texturing

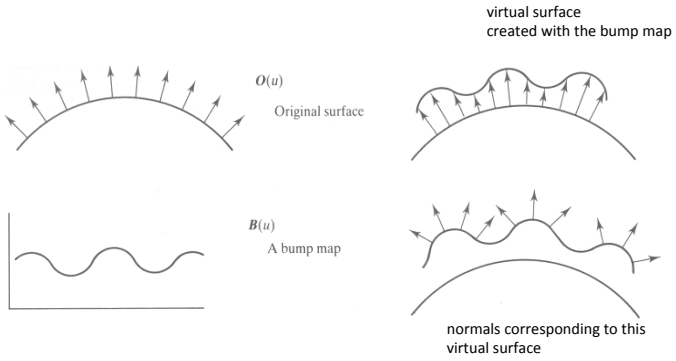


BUMP MAPPING: NORMALS AS TEXTURE

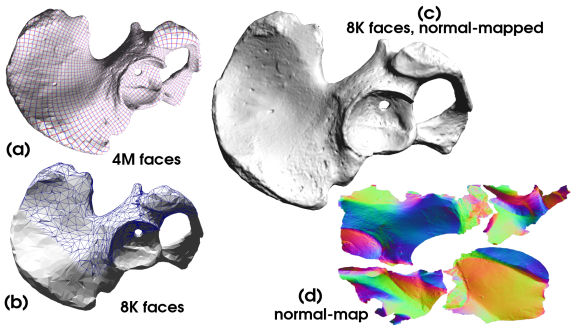
- object surface often not smooth – to recreate correctly need complex geometry model
- can control shape "effect" by locally perturbing surface normal
  - random perturbation
  - directional change over region

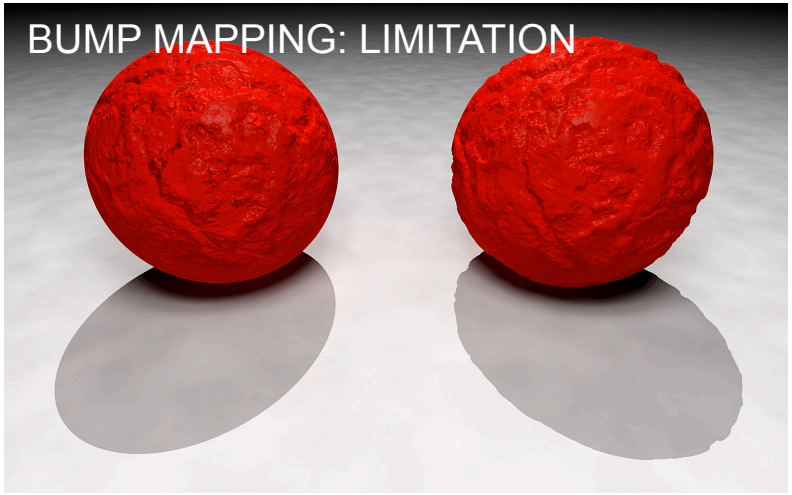


BUMP MAPPING



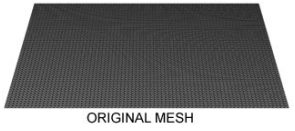
Normal/Bump mapping





DISPLACEMENT MAPPING

- bump mapping gets silhouettes wrong
  - shadows wrong too
- change surface geometry instead
  - only recently available with realtime graphics
  - need to subdivide surface


ORIGINAL MESH

DISPLACEMENT MAP

MESH WITH DISPLACEMENT

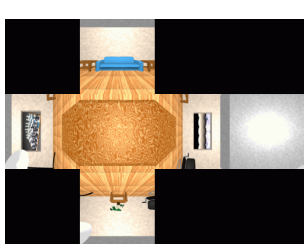
https://en.wikipedia.org/wiki/Displacement_mapping#/media/

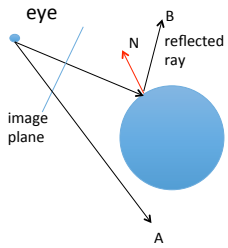
ENVIRONMENT MAPPING

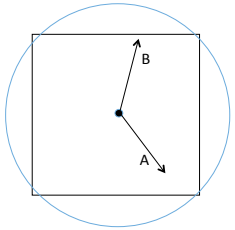
- generate image of surrounding or reflection
- sphere map or cube map

CUBE MAP

- 6 planar textures, sides of cube
  - point camera in 6 different directions, facing out from origin





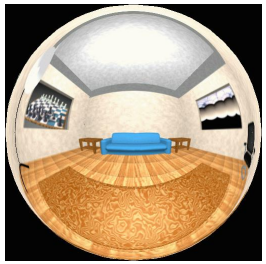


note: viewpoint is always at the center!

- 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

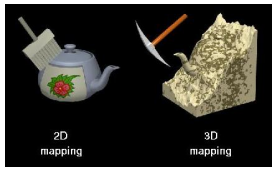
- texture is distorted fish-eye view
  - point camera at mirrored sphere
  - spherical texture mapping creates texture coordinates that correctly index into this texture map



VOLUMETRIC TEXTURE

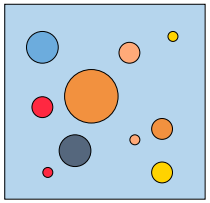
- define texture pattern over 3D domain - 3D space containing the object
- texture function can be digitized or procedural
- for each point on object compute texture from point location in space
- e.g., ShaderToy

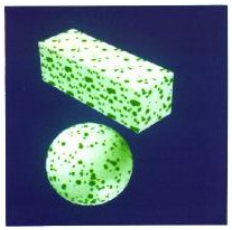
- computing is cheap, memory access not as much



PROCEDURAL TEXTURE EFFECTS: BOMBING

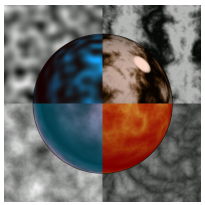
- randomly drop bombs of various shapes, sizes and orientation into texture space (store data in table)
  - for point P search table and determine if inside shape
    - if so, color by shape's color
    - otherwise, color by object's color





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/

THE RENDERING PIPELINE