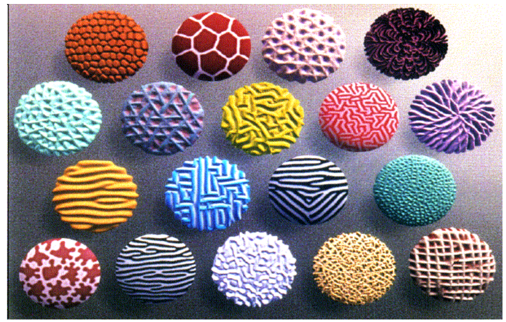
TEXTURE MAPPING

TEXTURE MAPPING

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



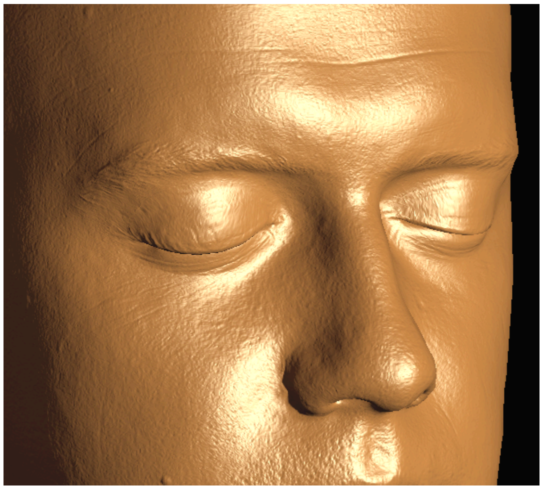
TEXTURE MAPPING

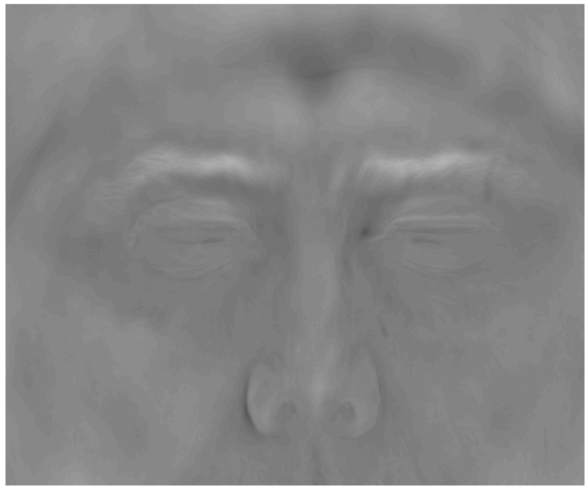
- hide geometric simplicity
  - images convey illusion of geometry
  - map a brick wall texture on a flat polygon
  - create bumpy effect on surface

- 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 !!

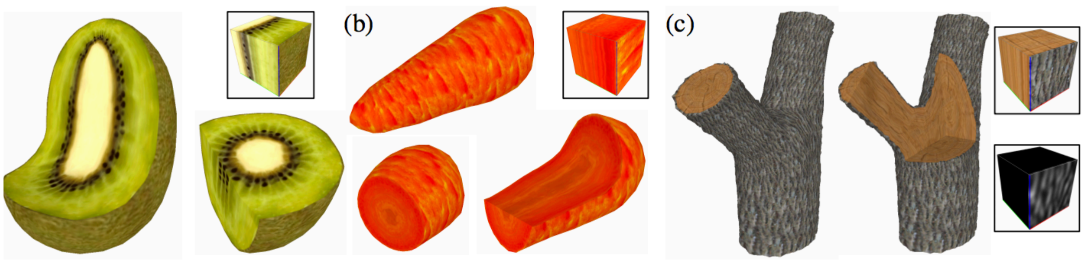




threejs.org: materials/bumpmap

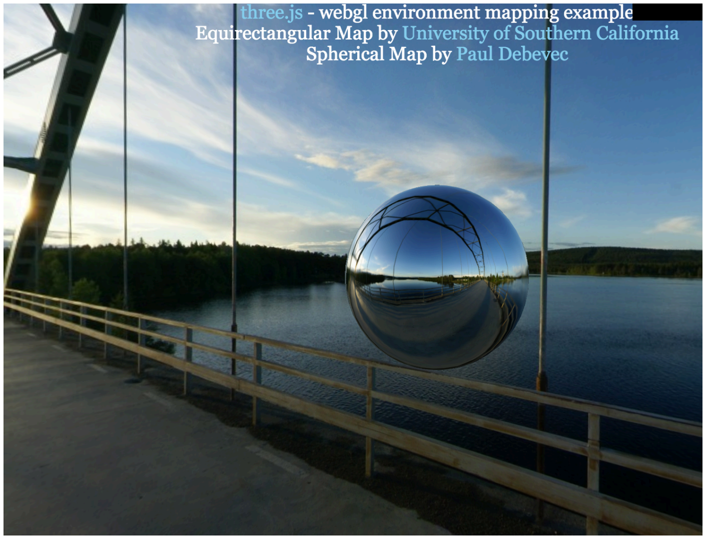
VOLUMETRIC TEXTURES

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



[Lapped Solid Textures, SIGGRAPH 2008]

ENVIRONMENT MAP

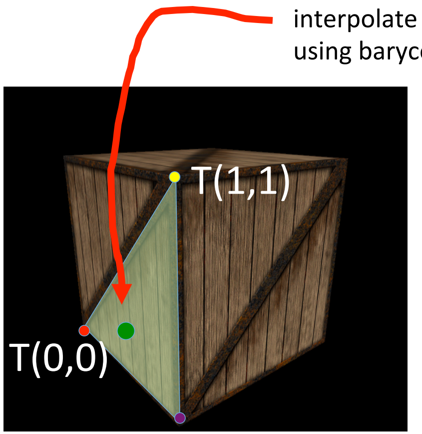


There is an invisible corner seam in this image!

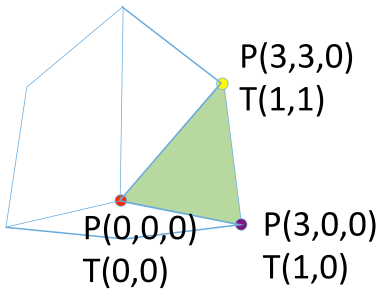
2 of 6 images for a cube map;
as a viewer, you are inside this cube!

Texture coords: (u,v) or (s,t)

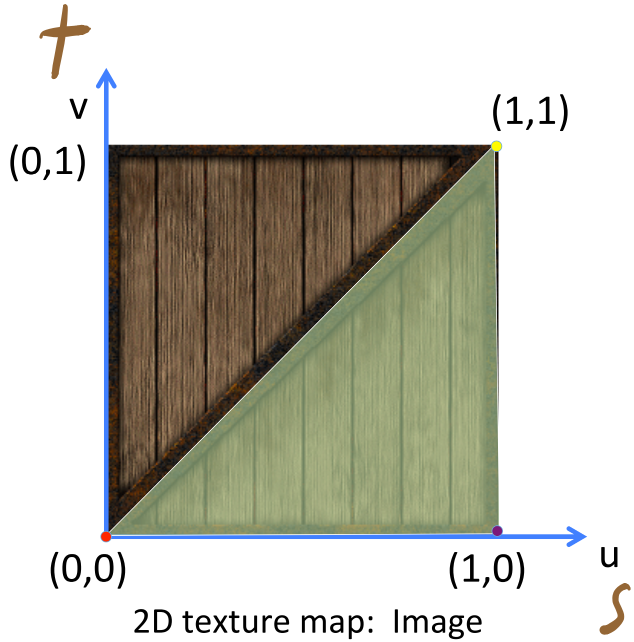
BASIC TEXTURE MAP



rendered image

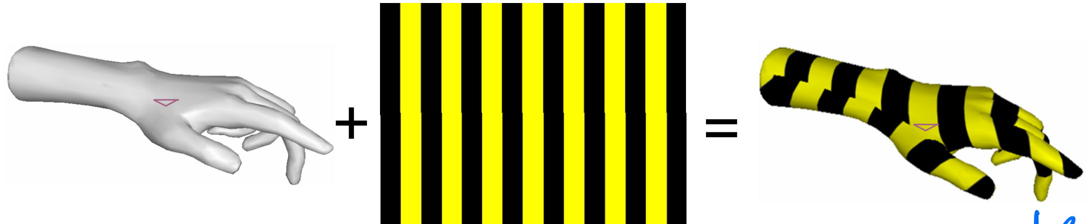


3D model:
u,v texture coodinates
are assigned to vertices
by artist or program.

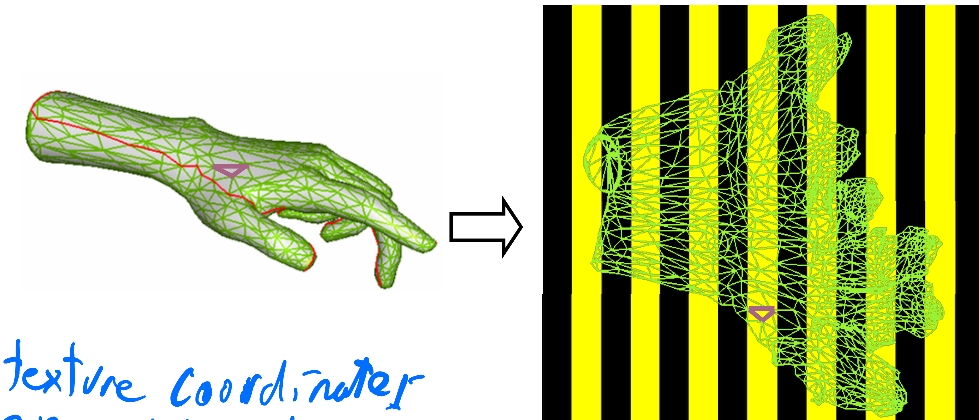


2D texture map: Image
Pixels here are called “texels”

TEXTURE MAPPING EXAMPLE



texture map:
image or
texture
atlas image



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

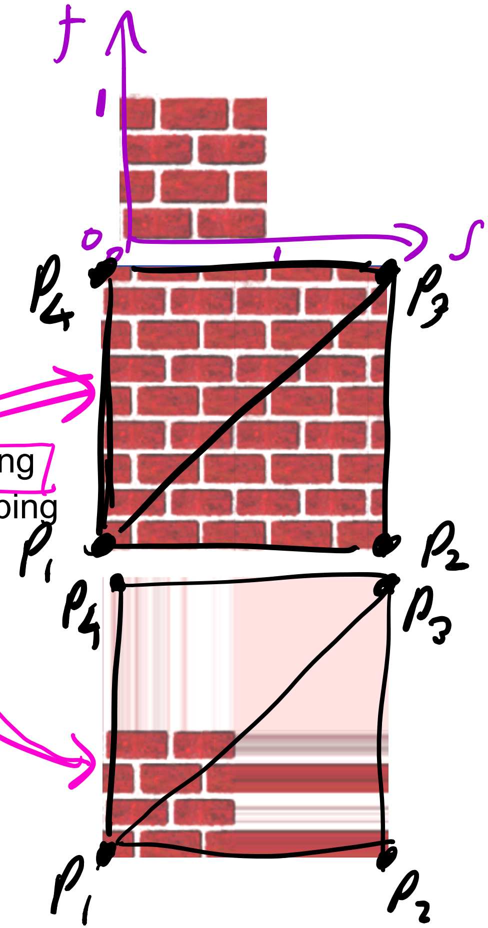
(s,t)

P_1 : $(0,0)$

P_2 : $(2.2, 0)$

P_3 : $(2.2, 2.2)$

P_4 : $(0, 2.2)$



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)

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

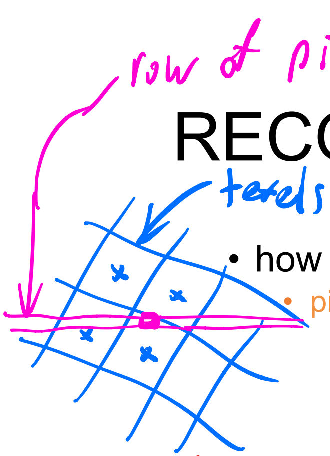
- vertex shader

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

- Fragment Shader:

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

RECONSTRUCTION



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

```
THREE.NearestFilter
THREE.LinearFilter
```

→ use nearest texel

→ interpolate btwn nearest 4 texels

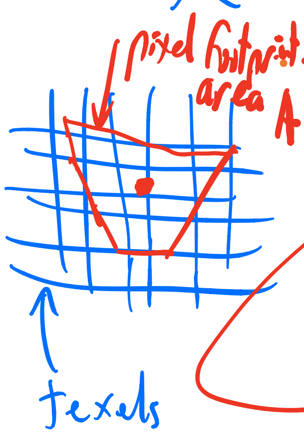
  - pixels that are much ~~smaller~~ larger than texels ?
    - ~~magnification~~ minification (zoom out)

```
THREE.NearestFilter
THREE.NearestMipMapNearestFilter
THREE.NearestMipMapLinearFilter
THREE.LinearFilter
THREE.LinearMipMapNearestFilter
THREE.LinearMipMapLinearFilter
```

NearestFilter → use nearest texel in original image.

LinearFilter → interpolate in original image.

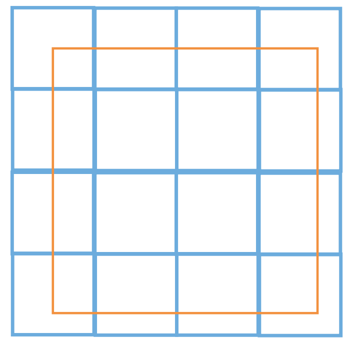
LinearMipMapLinearFilter → "best" mode.

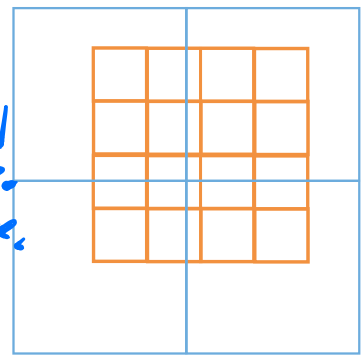


Nearest MipMap Linear Filter

Nearest ↳ use nearest MipMap level

Linear ↳ interpolate within MipMap level.

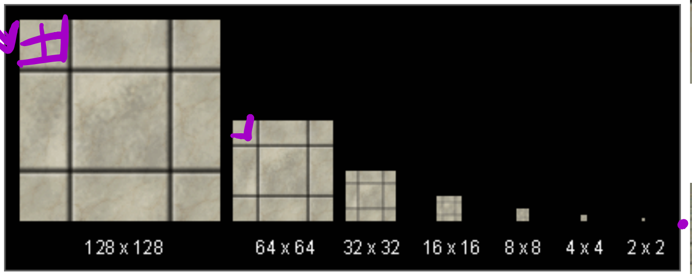




MIPMAPPING

$2^n \times 2^m$ original

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



MipMap level: 0 1 2 3 4 5 6 7

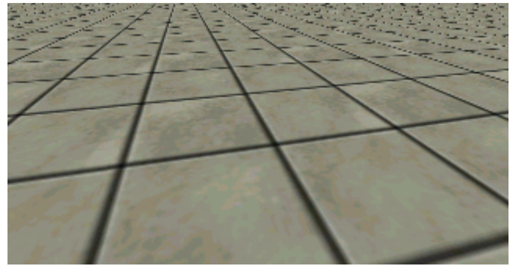
Area: # of orig texels covered: 4^0 4^1 4^2 4^3 4^4 ...

1 4 16

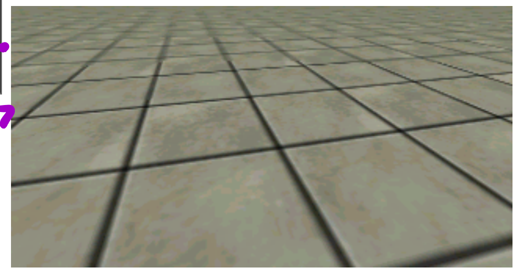
For a given pixel with texel area A, e.g. A=10

then do lookups in the two nearest Mipmaps levels

and then interpolate



Without MIP-mapping



With MIP-mapping

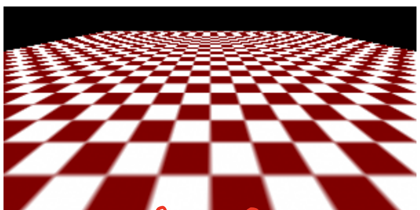
MIPMAPS

- **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

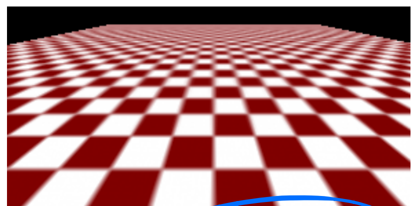
e.g.:

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

without



with



area of original image

Extra memory needed for a MipMap:

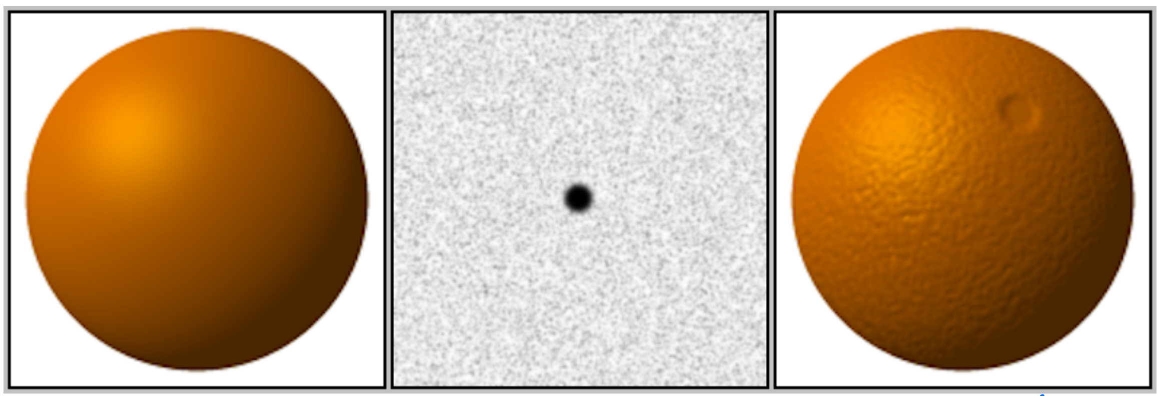
$1 + \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \ldots =$

extra 1/3 space

$\Sigma\ 1 + a + a^2 \ldots = \frac{1}{1-a} = \frac{1}{3/4} = \frac{4}{3}$

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

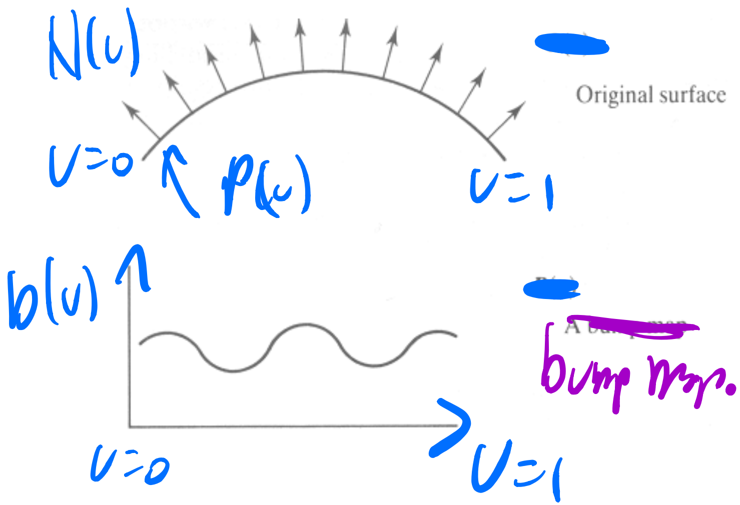


Spherical geometry (triangles)

"bump map" gray-scale image

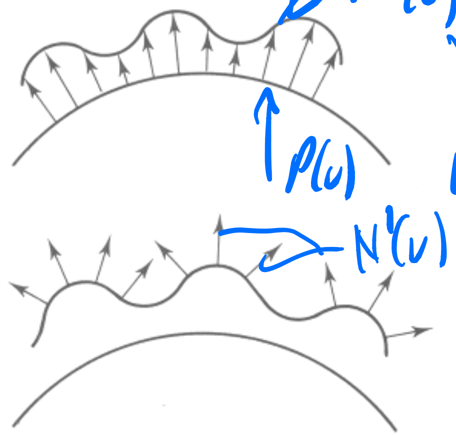
using bump-map to modify normals on a per-pixel basis.

BUMP MAPPING



virtual surface
created with the bump map

$P'(u) = P(u) + b(u)\vec{N}(u)$



normals corresponding to this
virtual surface

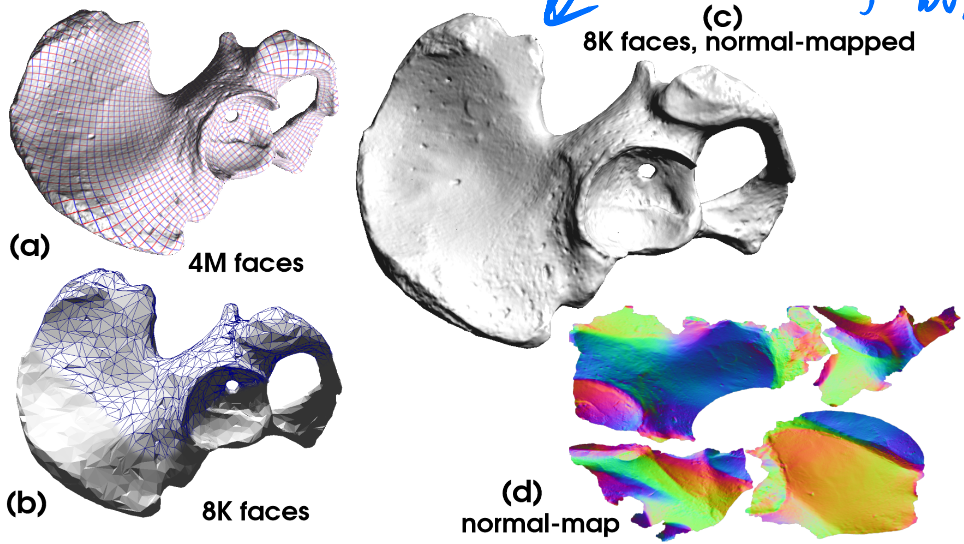
Compute $N'(u)$ using finite ~~diff~~ difference and a cross product.

Render surface $P(u), N'(u)$

Note: we could also provide $N'(u)$: This is a Normal map

Normal/Bump mapping

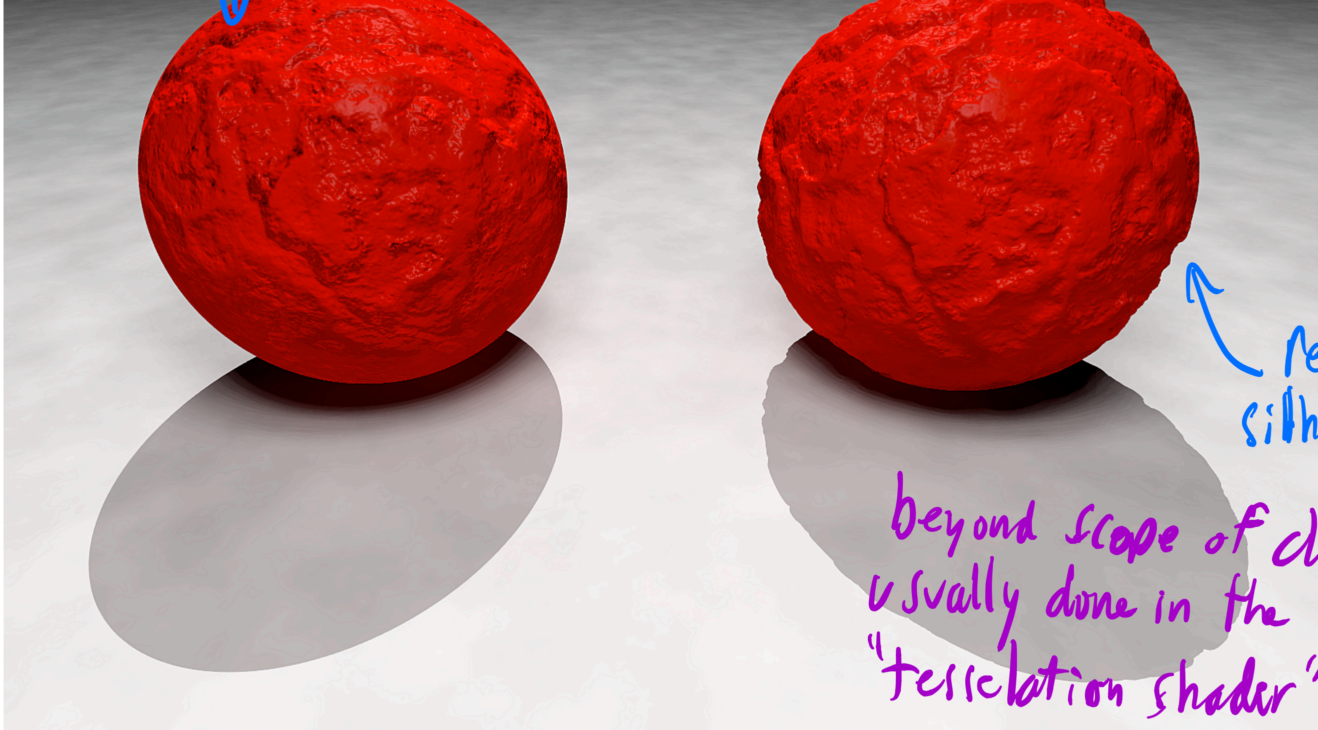
Achieve the appearance of high-resolution geometry using lower resolution geom. + bump map



bump mapped.

displacement map

BUMP MAPPING: LIMITATION

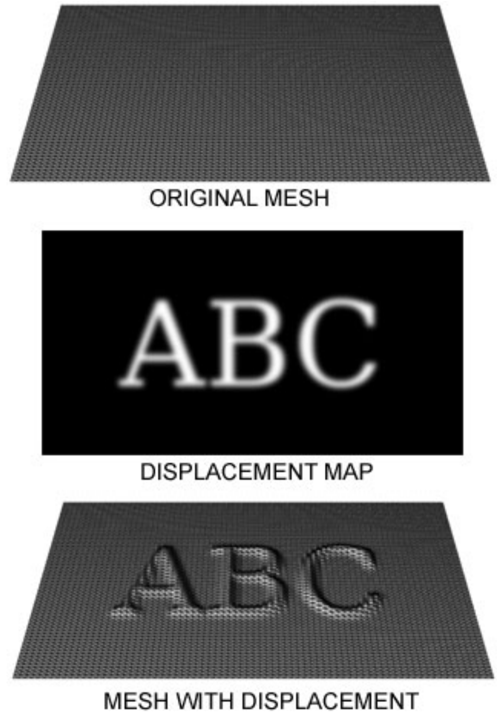


realistic silhouette

beyond scope of class; usually done in the "tesselation shader"

DISPLACEMENT MAPPING

- bump mapping gets silhouettes wrong
  - shadows wrong too
- change surface geometry instead
  - need to subdivide surface
  - use tesselation shader



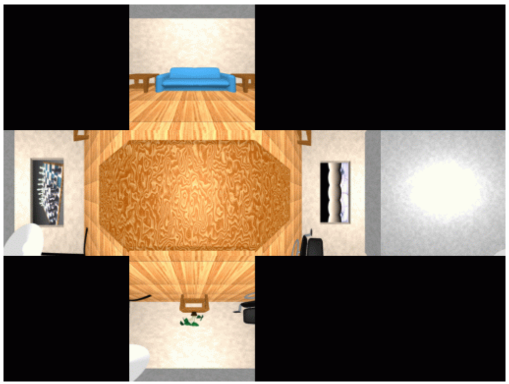
https://en.wikipedia.org/wiki/Displacement_mapping#/media/

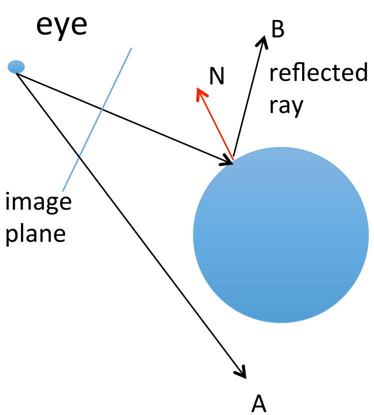
ENVIRONMENT MAPPING

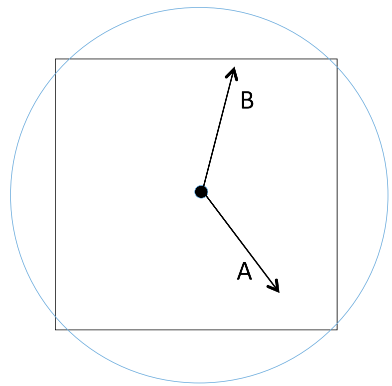
- generate image of surrounding or reflection
- sphere map or cube map ← these are two common types of environment maps.

CUBE MAP

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





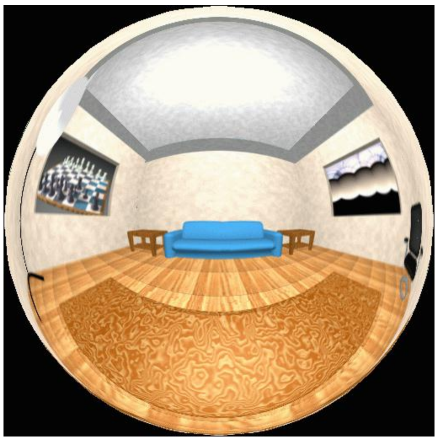


note: viewpoint is
always at the center!

- 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.

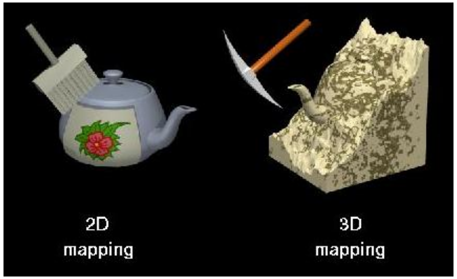
SPHERE MAP

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



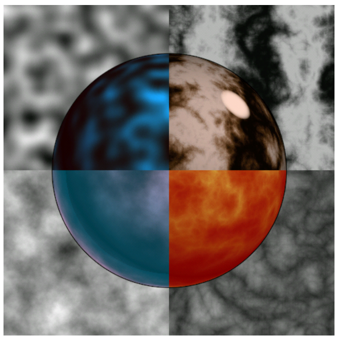
VOLUMETRIC TEXTURE

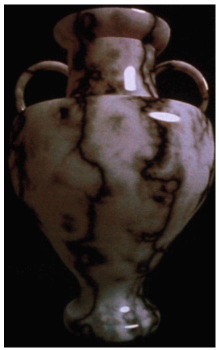
- define texture pattern over 3D domain - 3D space containing the object
- texture function can be digitized or procedural
- for each point on object compute texture from point location in space
- e.g., ShaderToy
- computation often cheaper than memory access



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/