CPSC 314
19 – TEXTURE MAPPING

Textbook: 15
13 (optional)

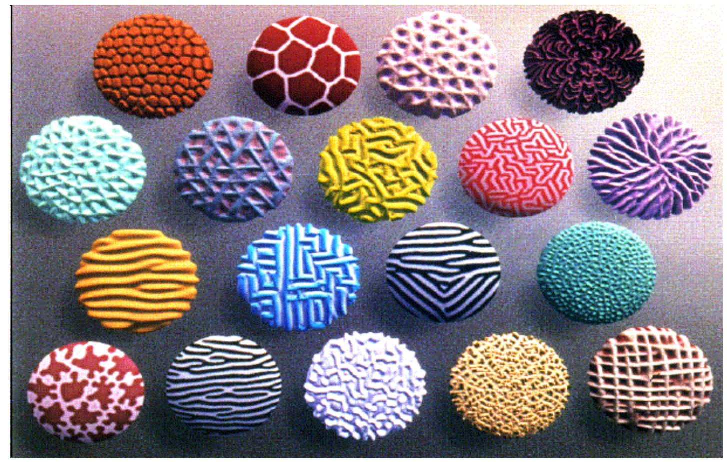
UGRAD.CS.UBC.CA/~CS314

Alla Sheffer

2016

TEXTURE MAPPING

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

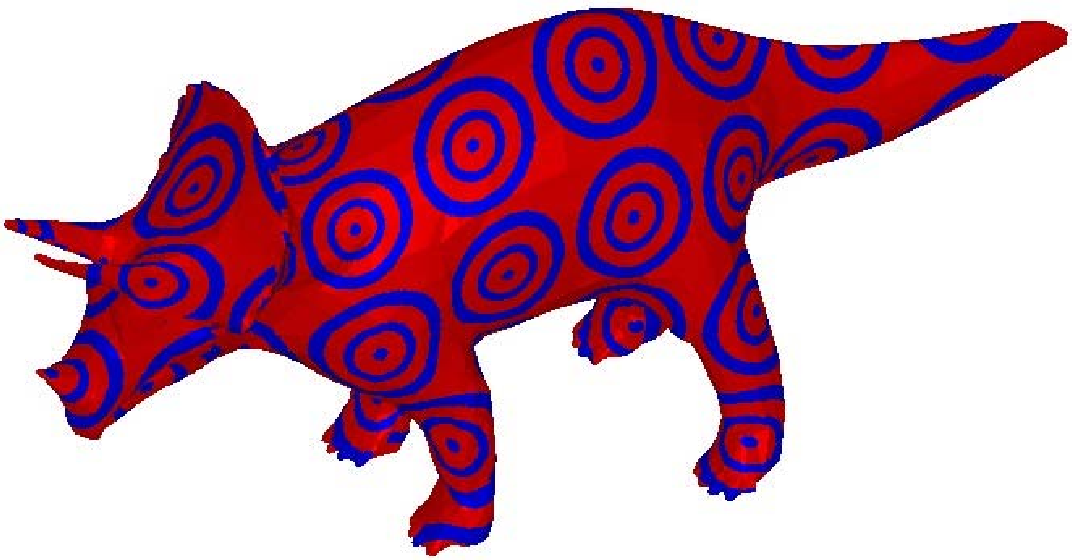


TEXTURE MAPPING

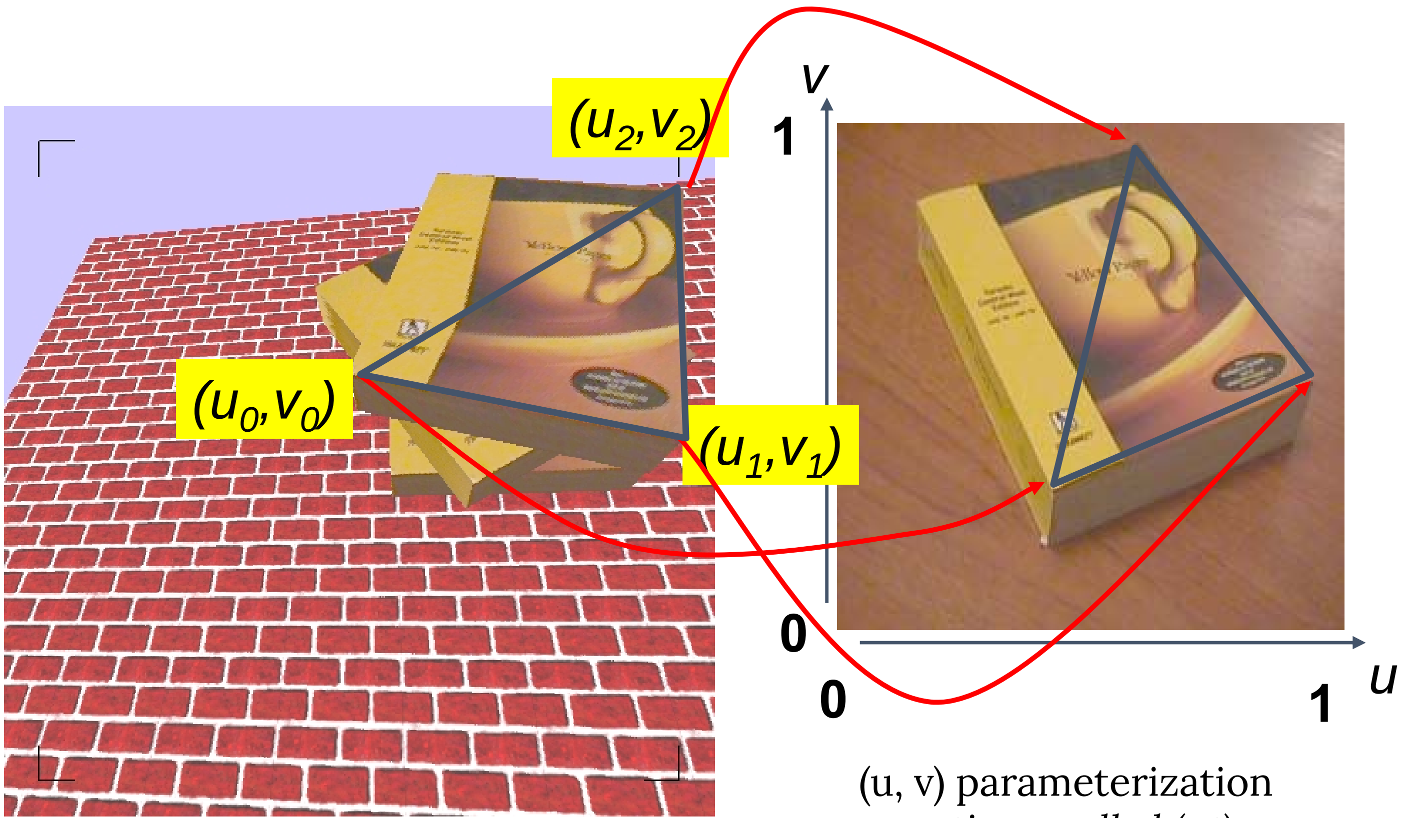
- hide geometric simplicity
  - images convey illusion of geometry
  - map a brick wall texture on a flat polygon
  - create bumpy effect on surface
- usually:
  associate 2D information with a surface in 3D
  - point on surface ↔ point in texture
  - “paint” image onto polygon

COLOR TEXTURE MAPPING

- define color (RGB) for each point on object surface
- other:
    - volumetric texture
    - procedural texture

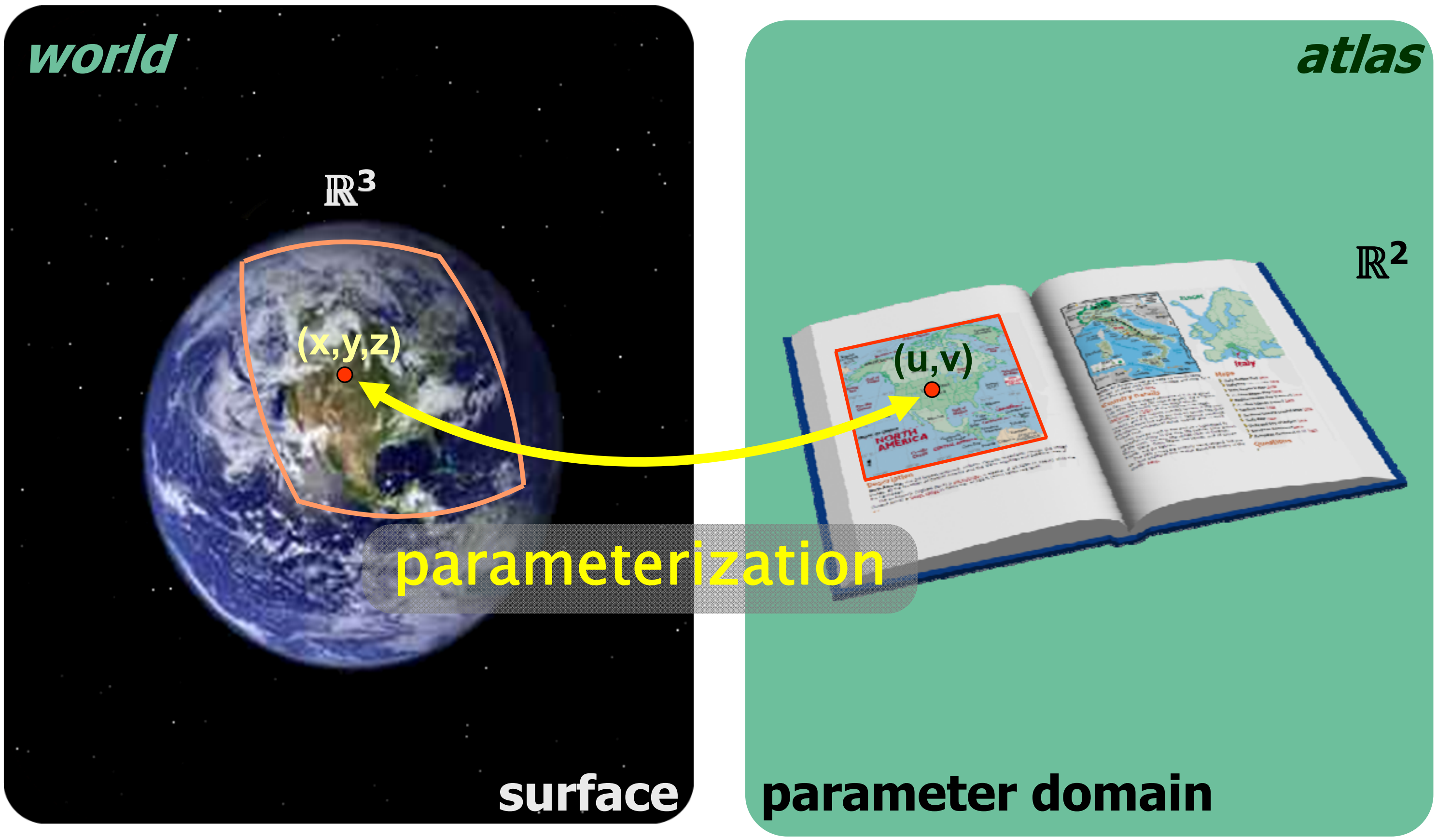


TEXTURE MAPPING



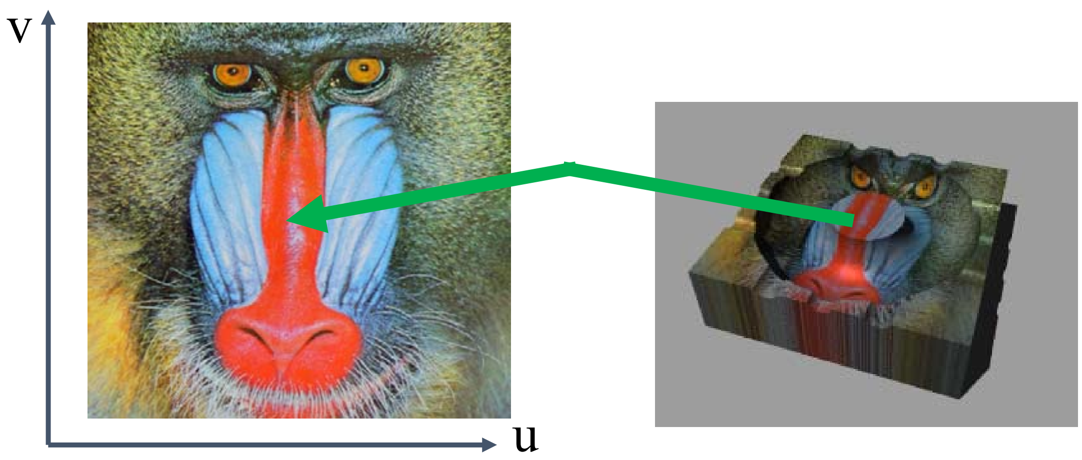
(u, v) parameterization
sometimes called (s,t)

2D TEXTURING = PARAMETERIZATION

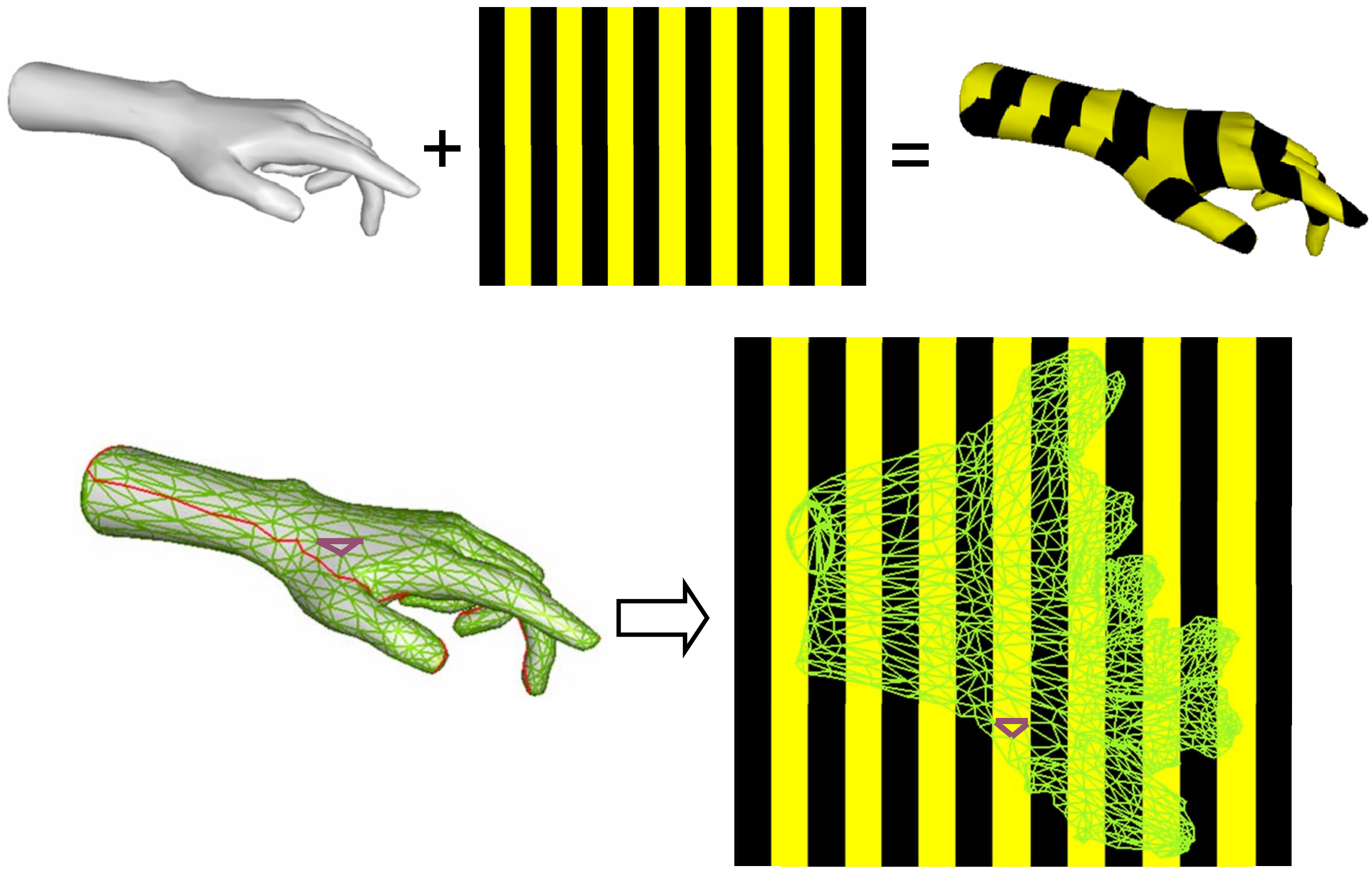


SURFACE TEXTURE

- Define texture pattern over (u,v) domain (Image)
  - Image – 2D array of "texels"
- Assign (u,v) coordinates to each point on object surface
  - How: depends on surface type
- For polygons (triangle)
  - Inside – use barycentric coordinates
  - For vertices need mapping function (artist/programmer)

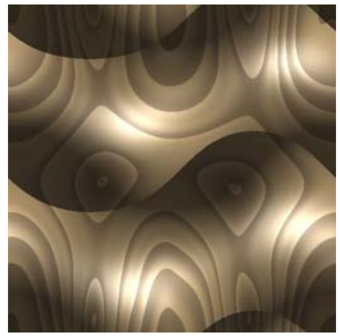


TEXTURE MAPPING EXAMPLE



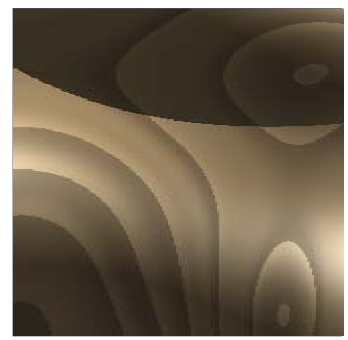
FRACTIONAL TEXTURE COORDINATES

texture image



(0,1) (1,1)

(0,0) (1,0)



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

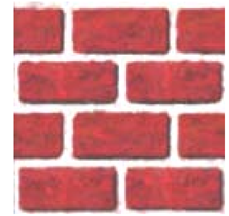
```
gl_FragColor = texColor;
```

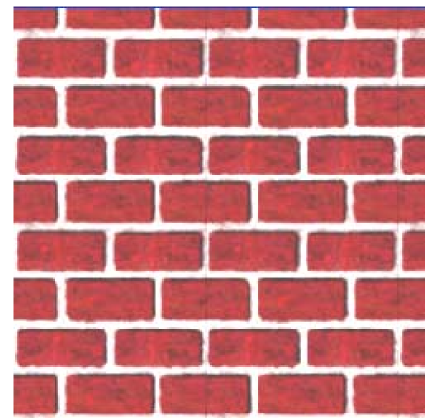
- Modulate
  - Use texture color as reflection color in illumination equation

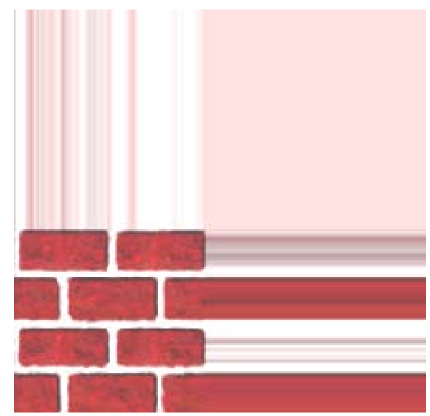
```
kd = texColor; ka = texColor;
gl_FragColor = ka*ia + kd*id*dotProduct + …;
```

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



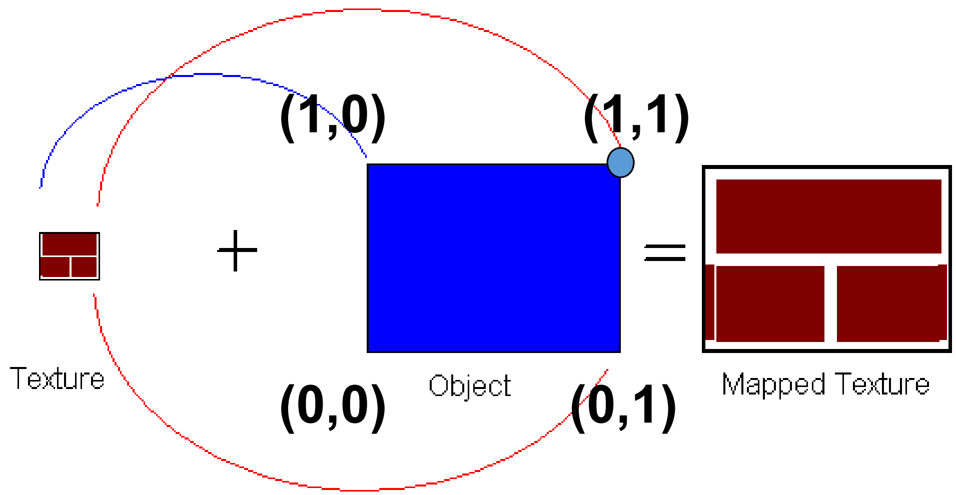


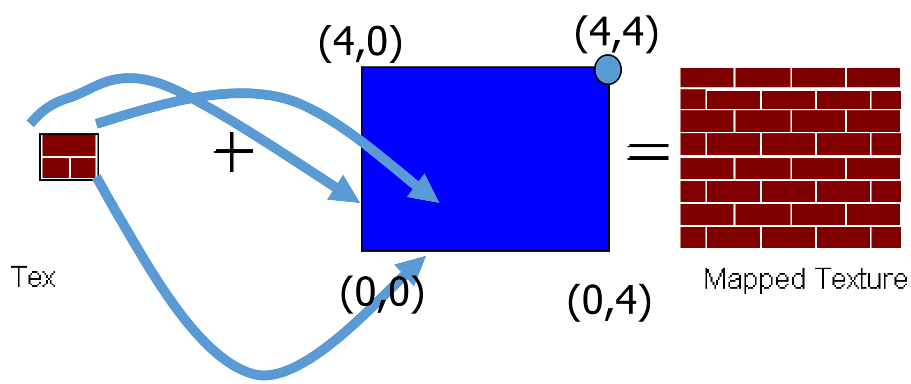


IN THREE.JS

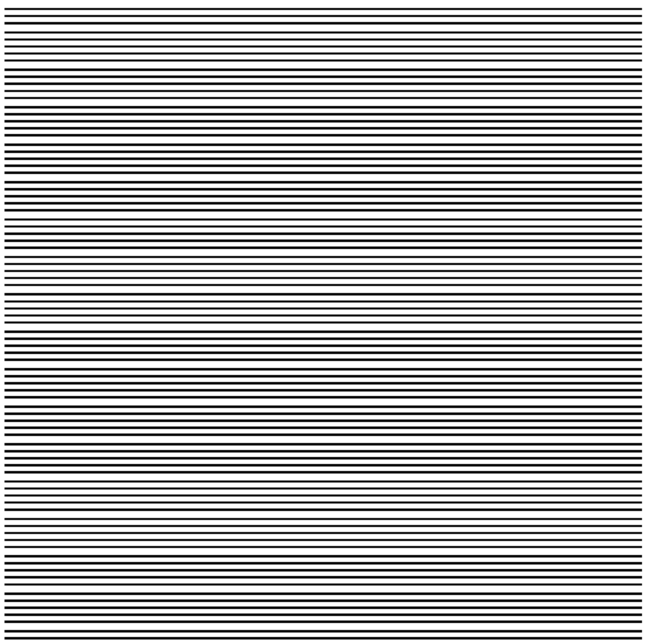
```
var texture = THREE.ImageUtils.loadTexture(
"textures/water.jpg" );
texture.wrapS = THREE.RepeatWrapping;
texture.wrapT = THREE.ClampToEdgeWrapping;
texture.repeat.set( 4, 4 );
```

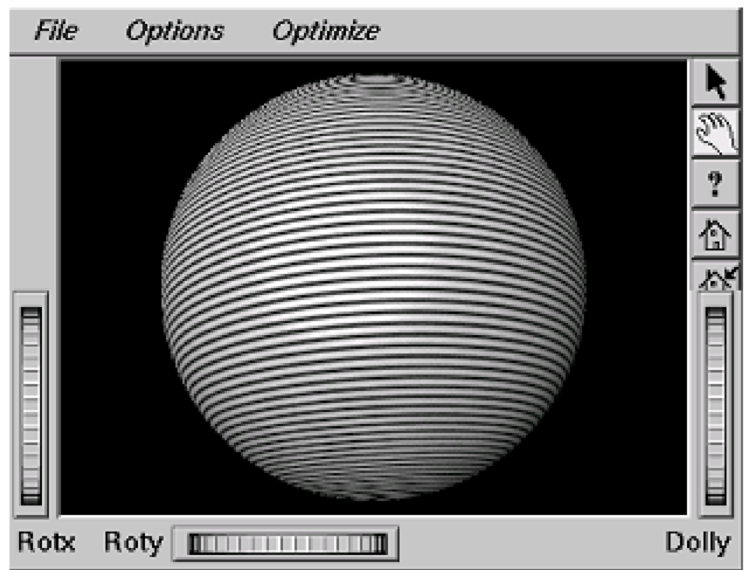
TILED TEXTURE MAP





RECONSTRUCTION

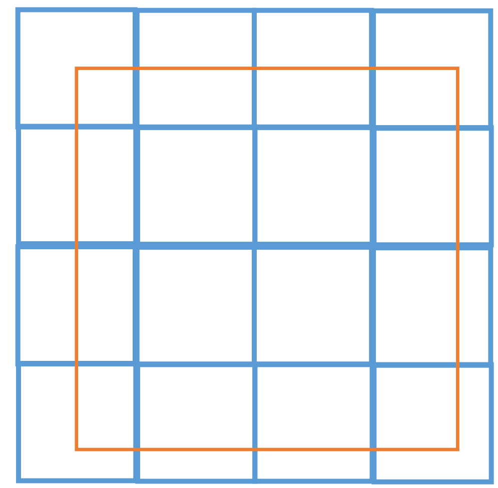




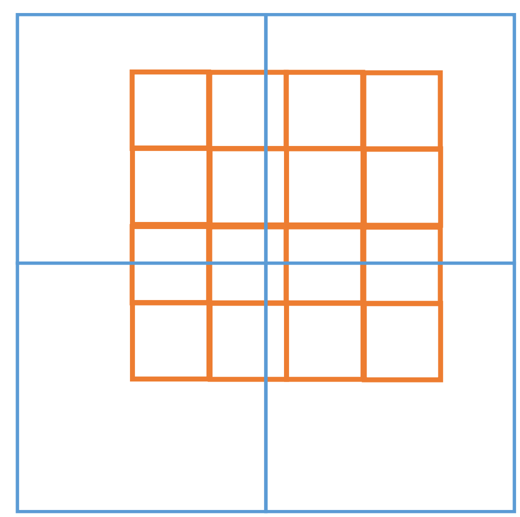
(image courtesy of Kiriakos Kutulakos, U Rochester)

RECONSTRUCTION

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

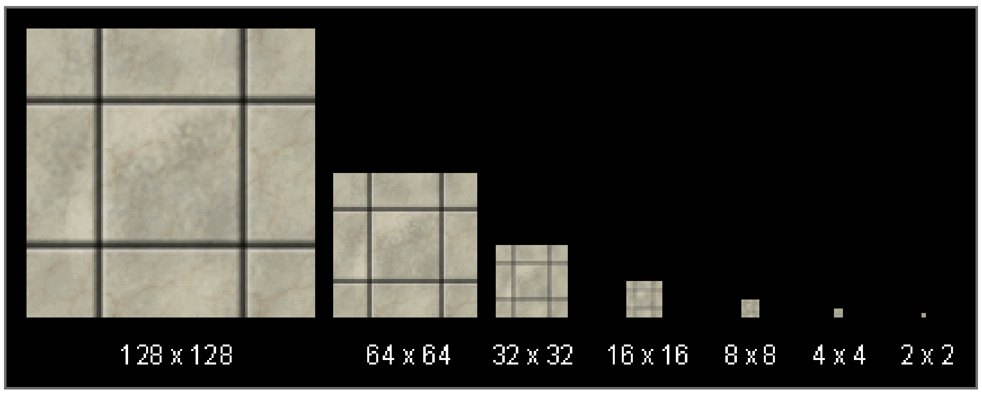


  - pixels that are much smaller than texels ?
    - magnification

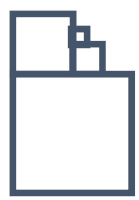


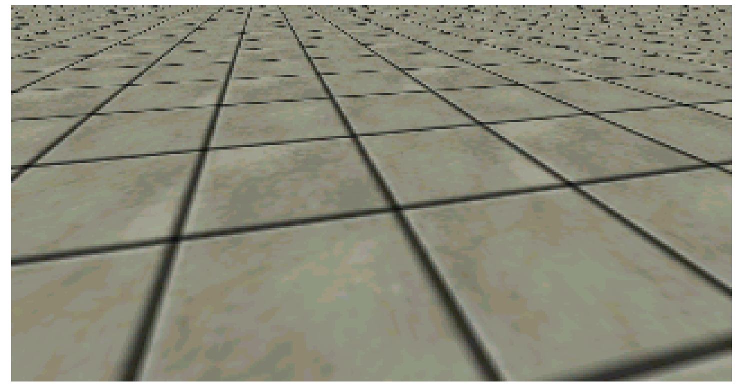
MIPMAPPING

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

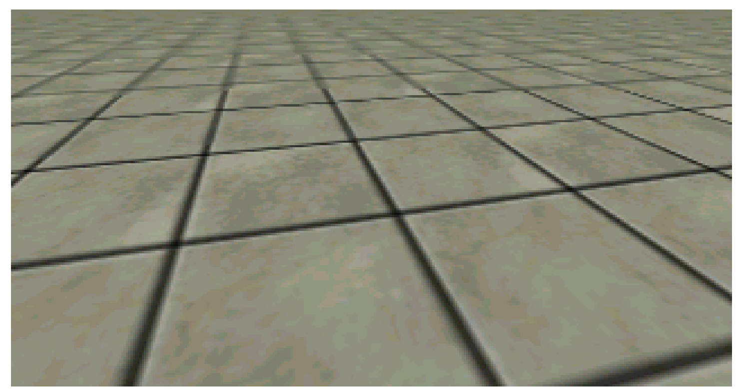


store whole pyramid in single block of memory





Without MIP-mapping

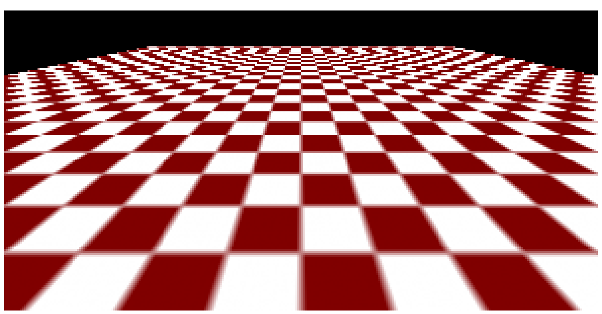


With MIP-mapping

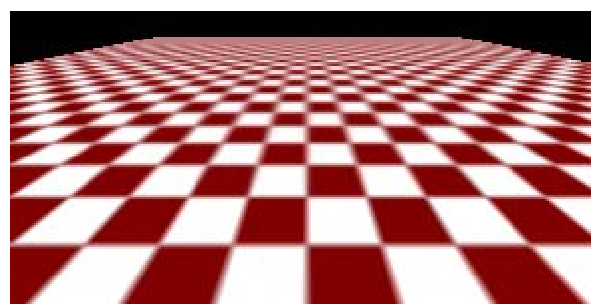
MIPMAPS

- **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

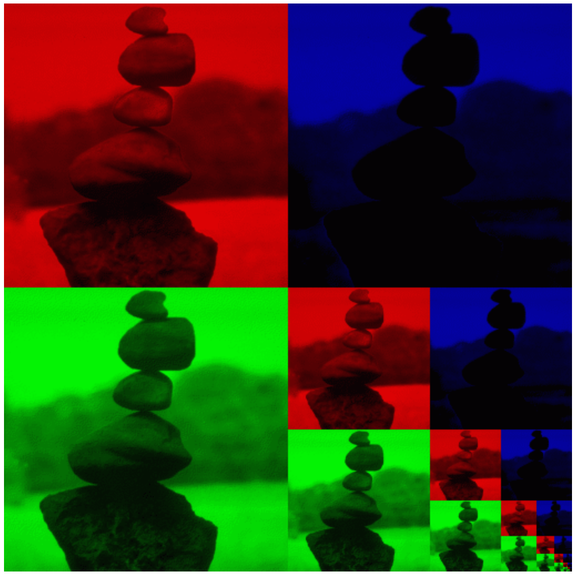


with



MIPMAP STORAGE

- only 1/3 more space required

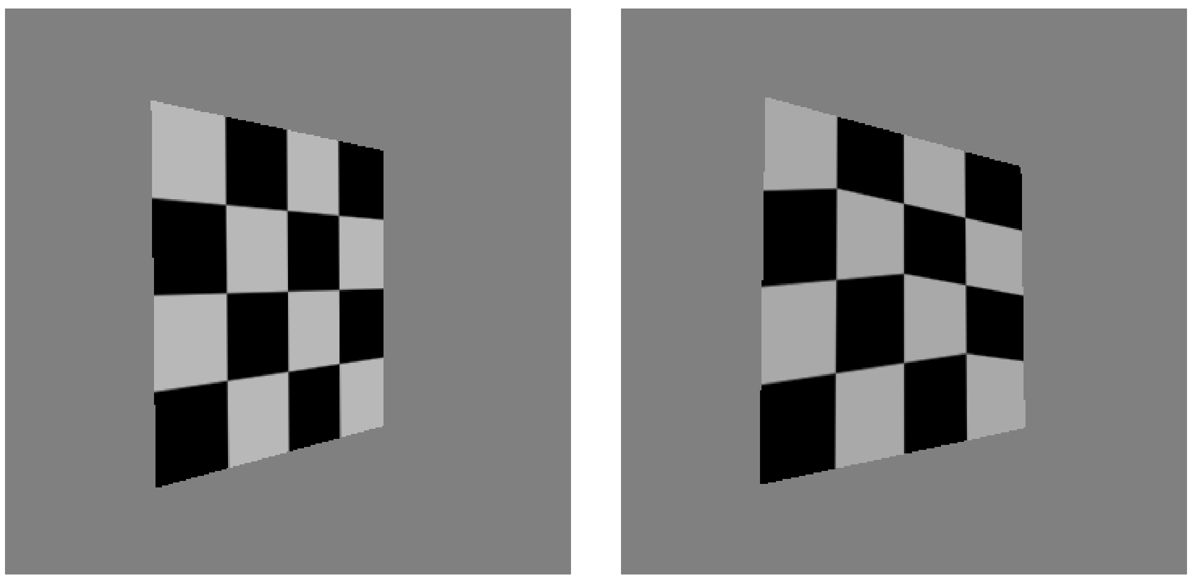


HOW TO INTERPOLATE S,T?

TEXTURE MAPPING

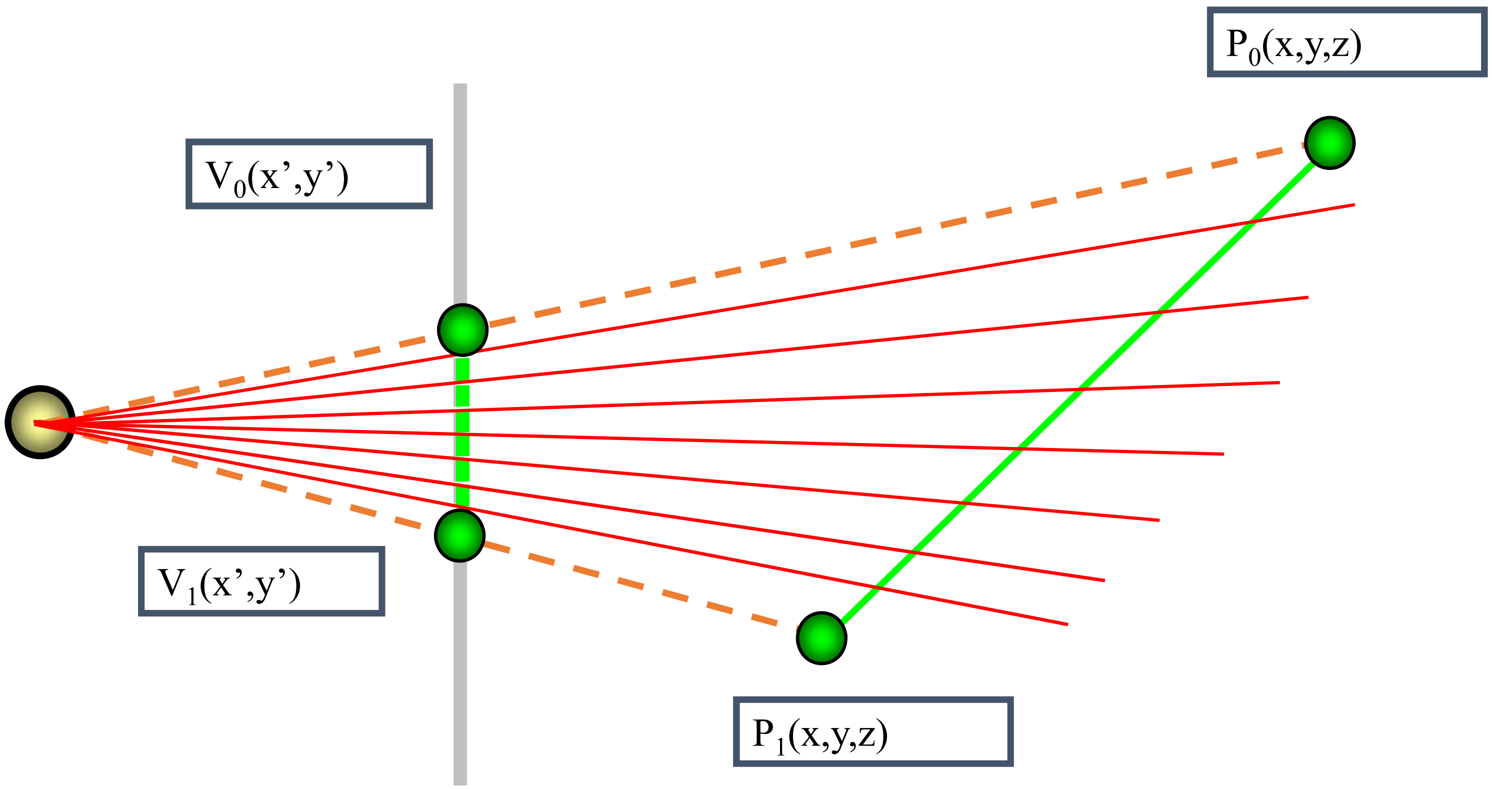
Texture coordinate interpolation

- Perspective foreshortening problem
- Also problematic for color interpolation, etc.



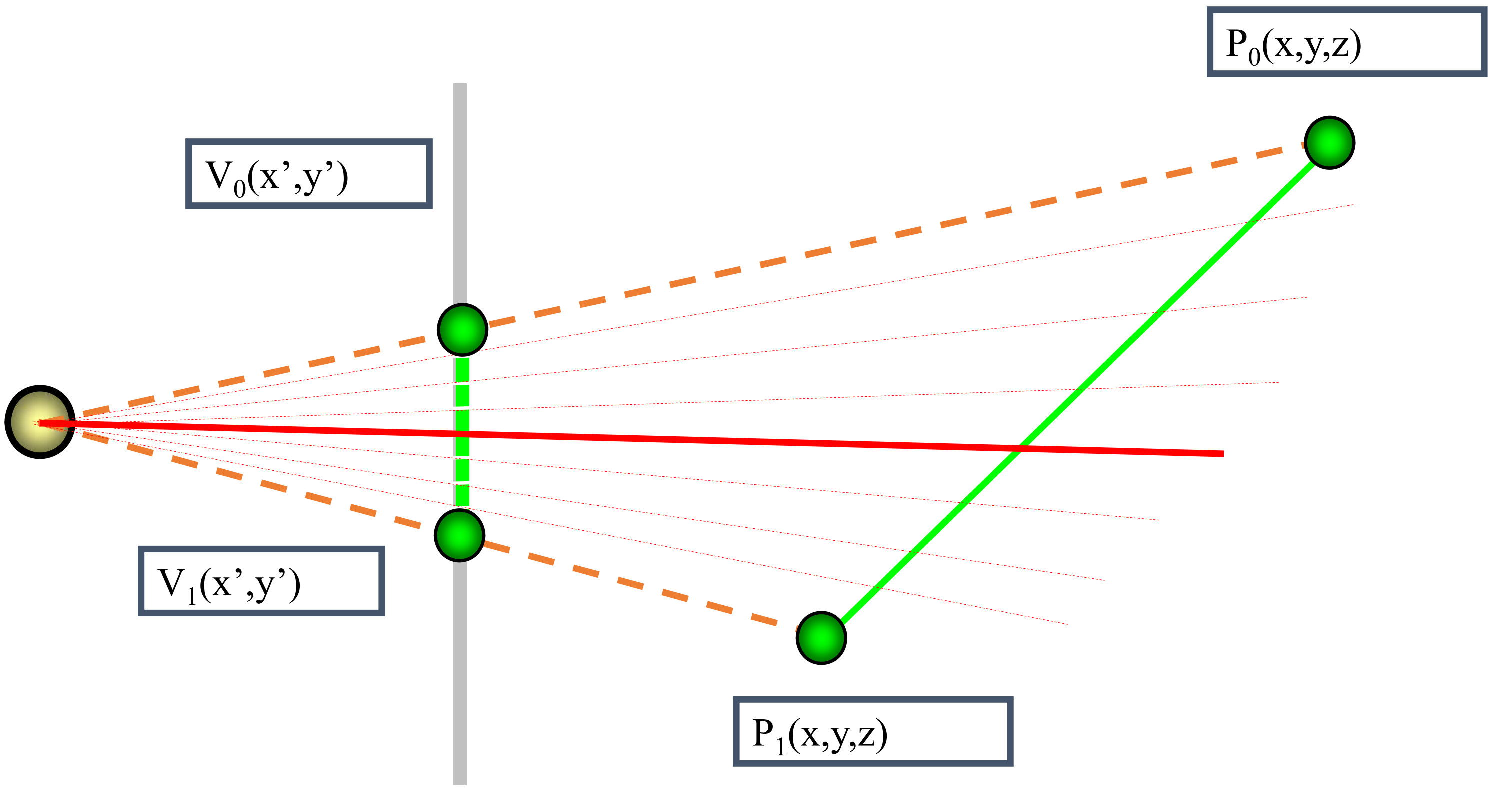
INTERPOLATION: SCREEN VS. WORLD SPACE

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



INTERPOLATION: SCREEN VS. WORLD SPACE

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



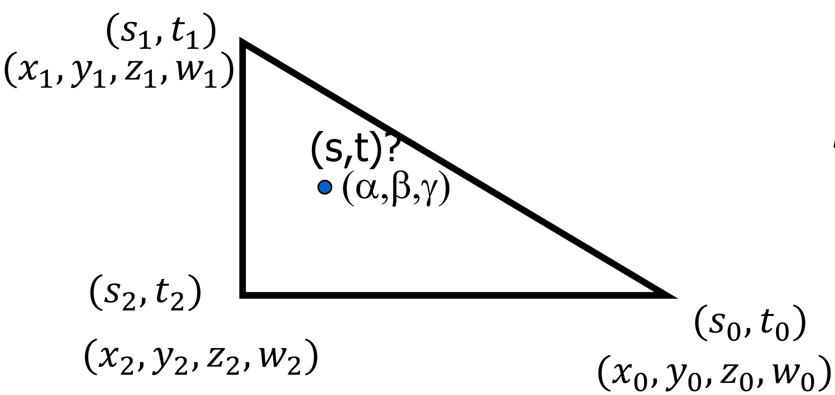
PERSPECTIVE - REMINDER

$$T\left(\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad z_{NDC} = \frac{a \cdot z_{eye} + b}{z_{eye}} = a + \frac{b}{z_{eye}}$$

- Preserves order
  - BUT distorts distances

TEXTURE COORDINATE INTERPOLATION

- Perspective Correct Interpolation
  - α, β, γ : Barycentric coordinates (2D) of point **P**
  - s_0, s_1, s_2 : texture coordinates of vertices
  - w_0, w_1, w_2 : homogenous coordinate of vertices



$$s = \frac{\alpha \cdot s_0 / w_0 + \beta \cdot s_1 / w_1 + \gamma \cdot s_2 / w_2}{\alpha / w_0 + \beta / w_1 + \gamma / w_2}$$

- Similarly for t

Derivation (similar triangles): https://www.comp.nus.edu.sg/~lowkl/publications/lowk_persp_interp_techrep.pdf

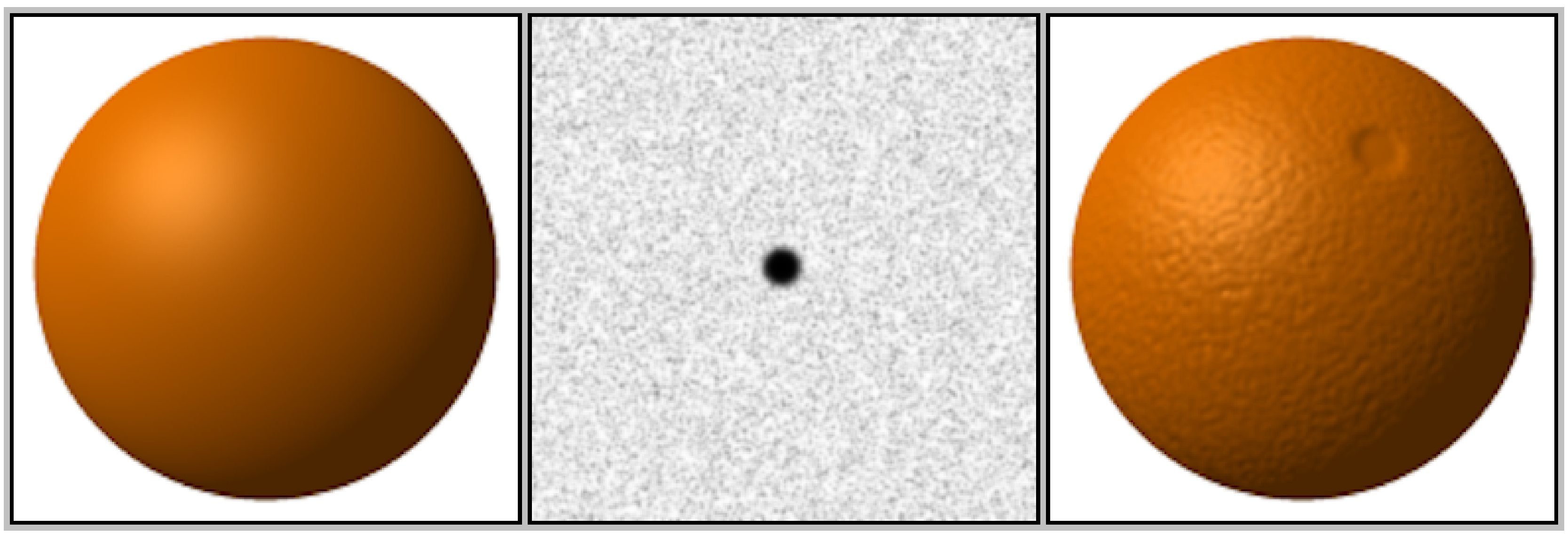
OTHER USES FOR TEXTURES

OTHER USES FOR TEXTURES

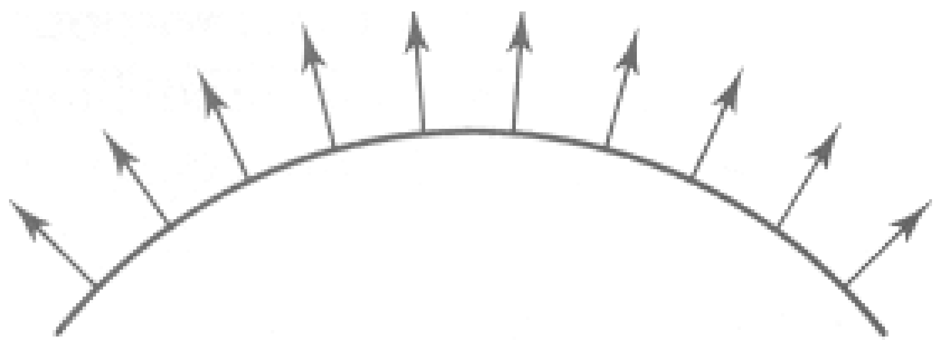
- usually provides colour, but …
- can also use to control other material/object properties
  - surface normal (bump mapping)
  - reflected color (environment mapping)

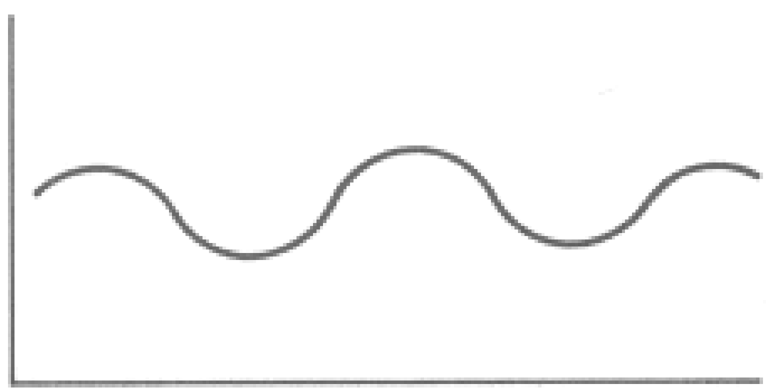
BUMP MAPPING: NORMALS AS TEXTURE

- 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

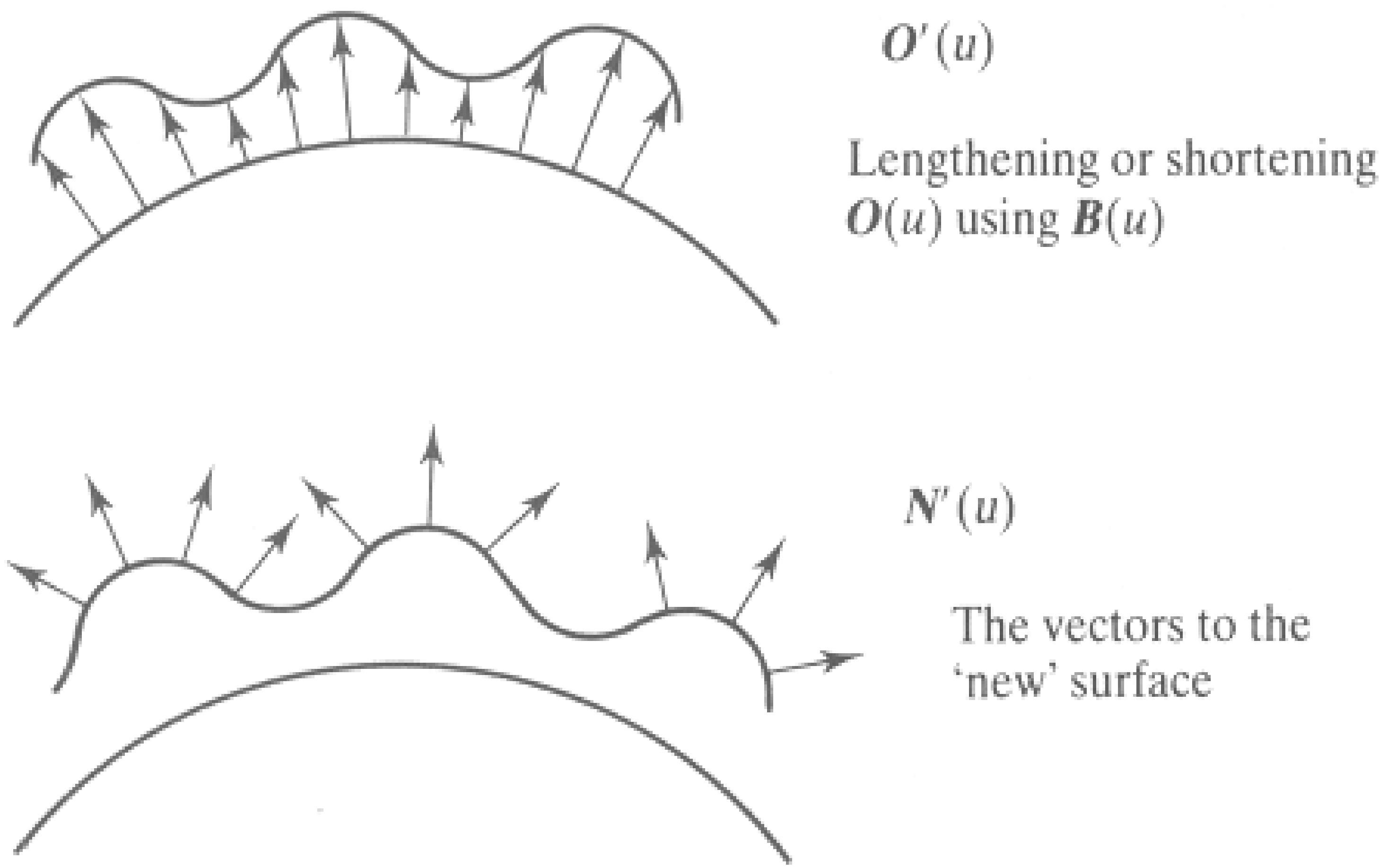


BUMP MAPPING

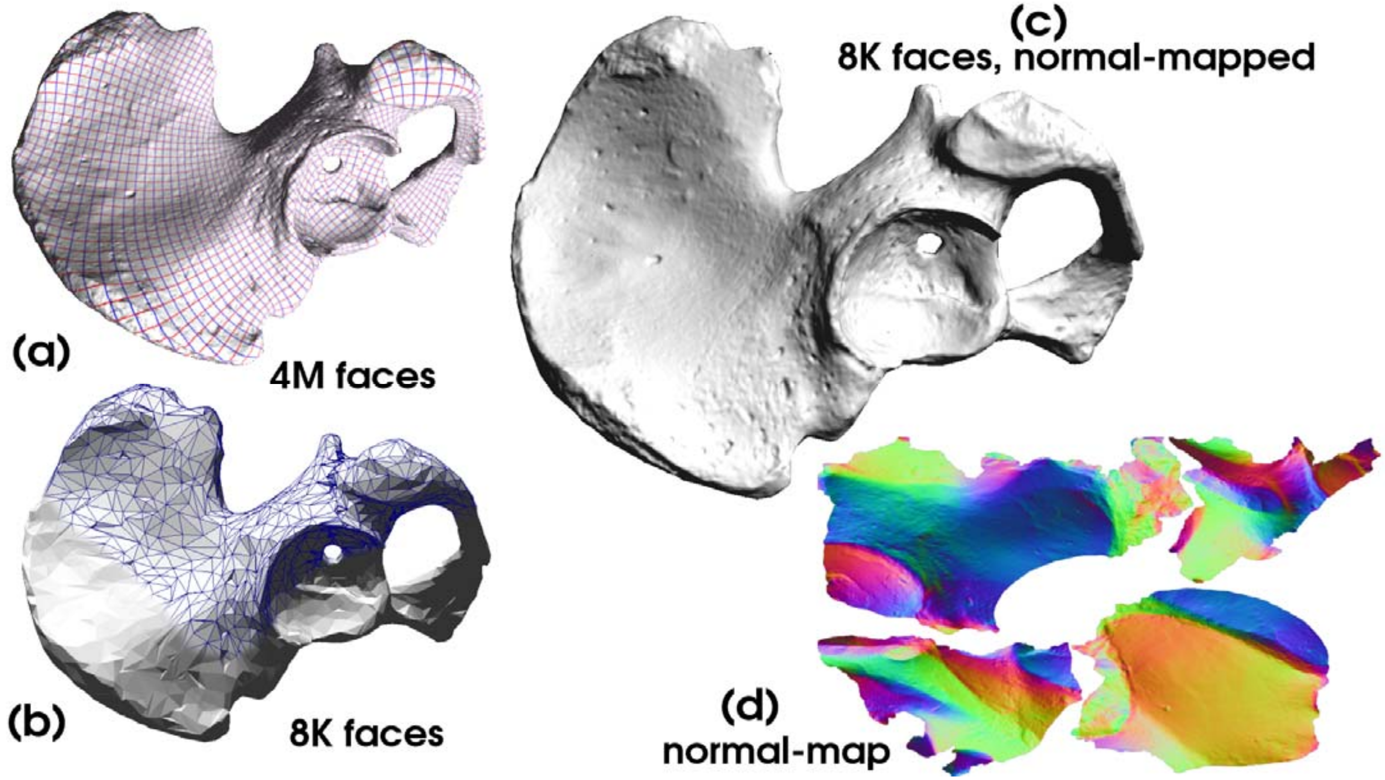




BUMP MAPPING

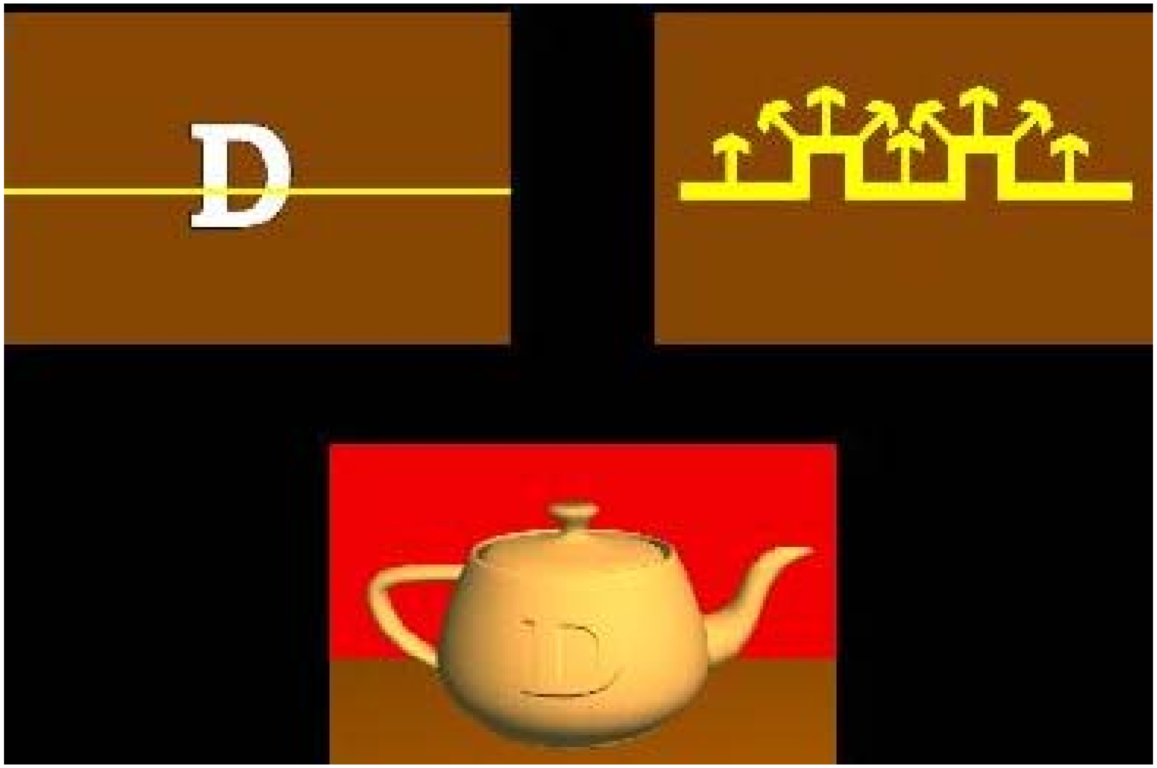


Normal/Bump mapping

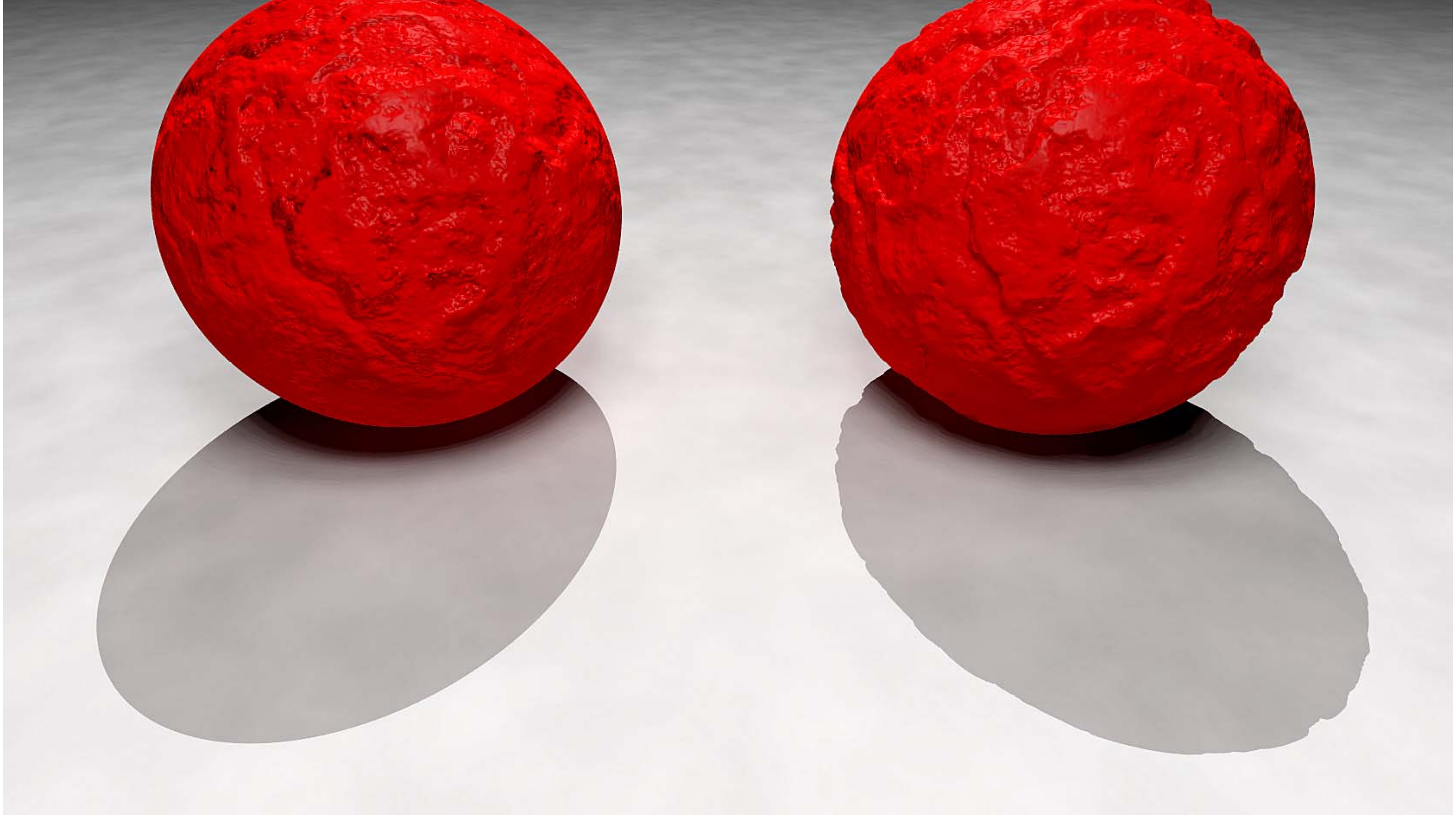


EMBOSSING

- at transitions
  - rotate point's surface normal by θ or - θ

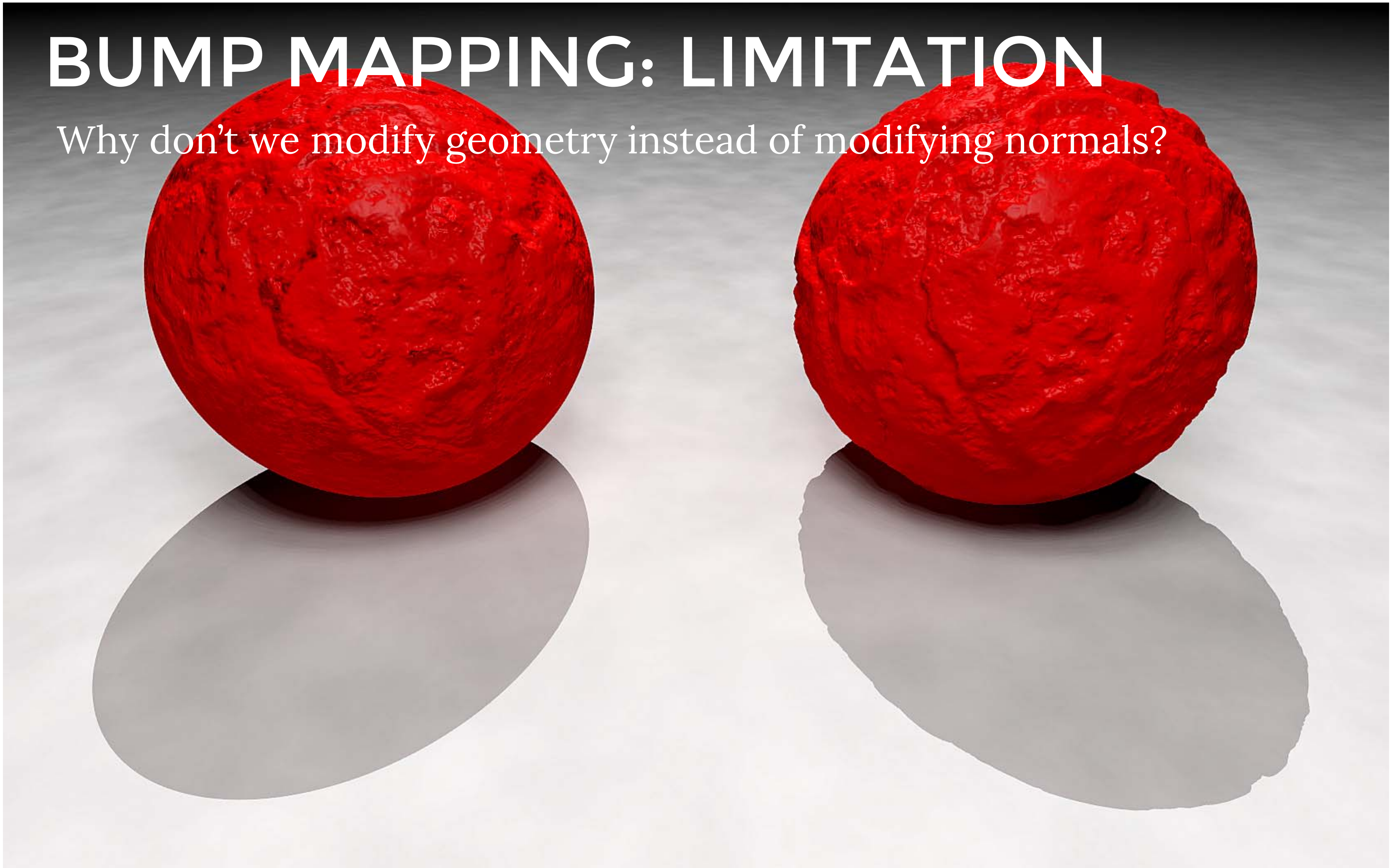


BUMP MAPPING: LIMITATION



BUMP MAPPING: LIMITATION

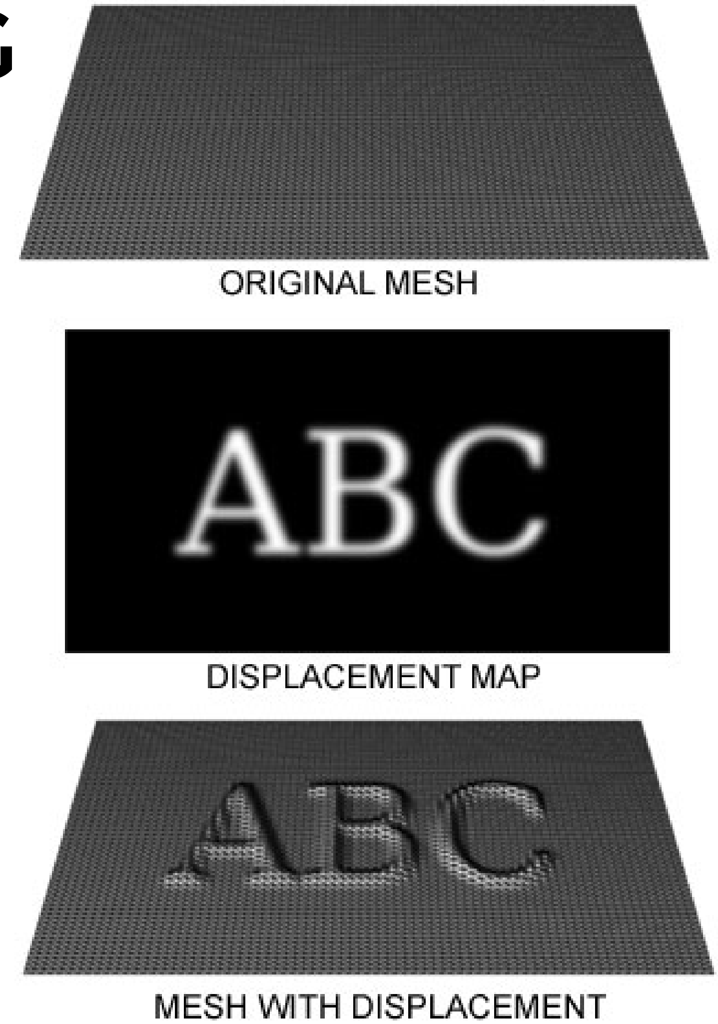
Why don’t we modify geometry instead of modifying normals?



DISPLACEMENT MAPPING

- bump mapping gets silhouettes wrong
  - shadows wrong too

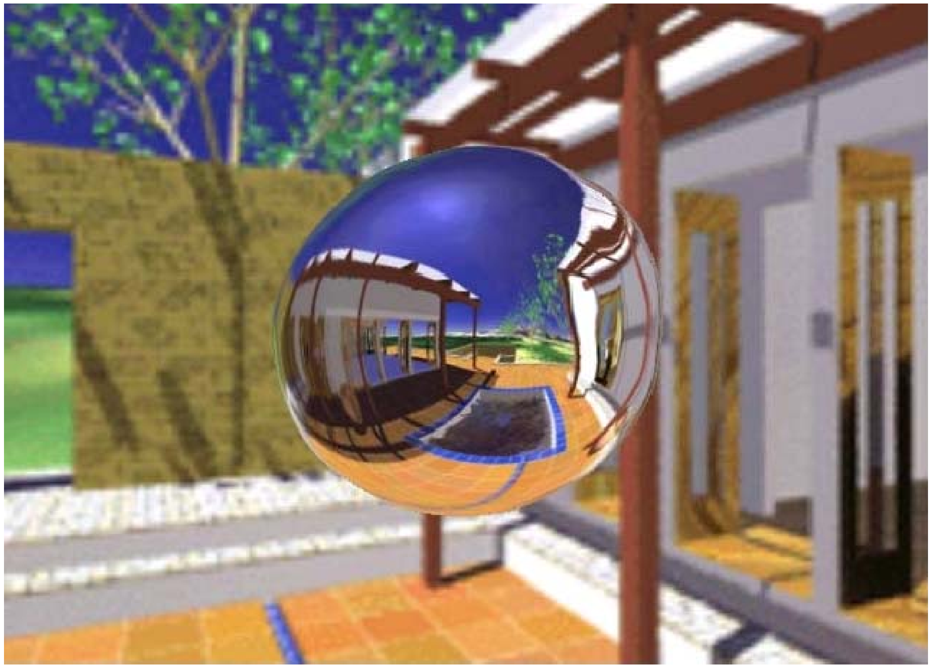
- change surface geometry instead
  - only recently available with realtime graphics
  - need to subdivide surface



https://en.wikipedia.org/wiki/Displacement_mapping#/media/File:Displacement.jpg

ENVIRONMENT MAPPING

- cheap way to achieve reflective effect
  - generate image of surrounding
  - map to object as texture



ENVIRONMENT MAPPING

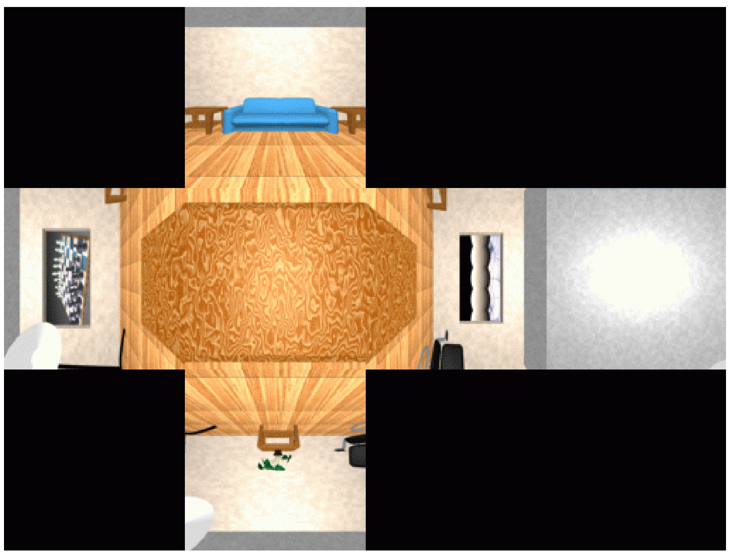
- used to model object that reflects surrounding textures to the eye
  - movie example: cyborg in Terminator 2
- different approaches
  - sphere, cube most popular
  - others possible too

SPHERE MAPPING

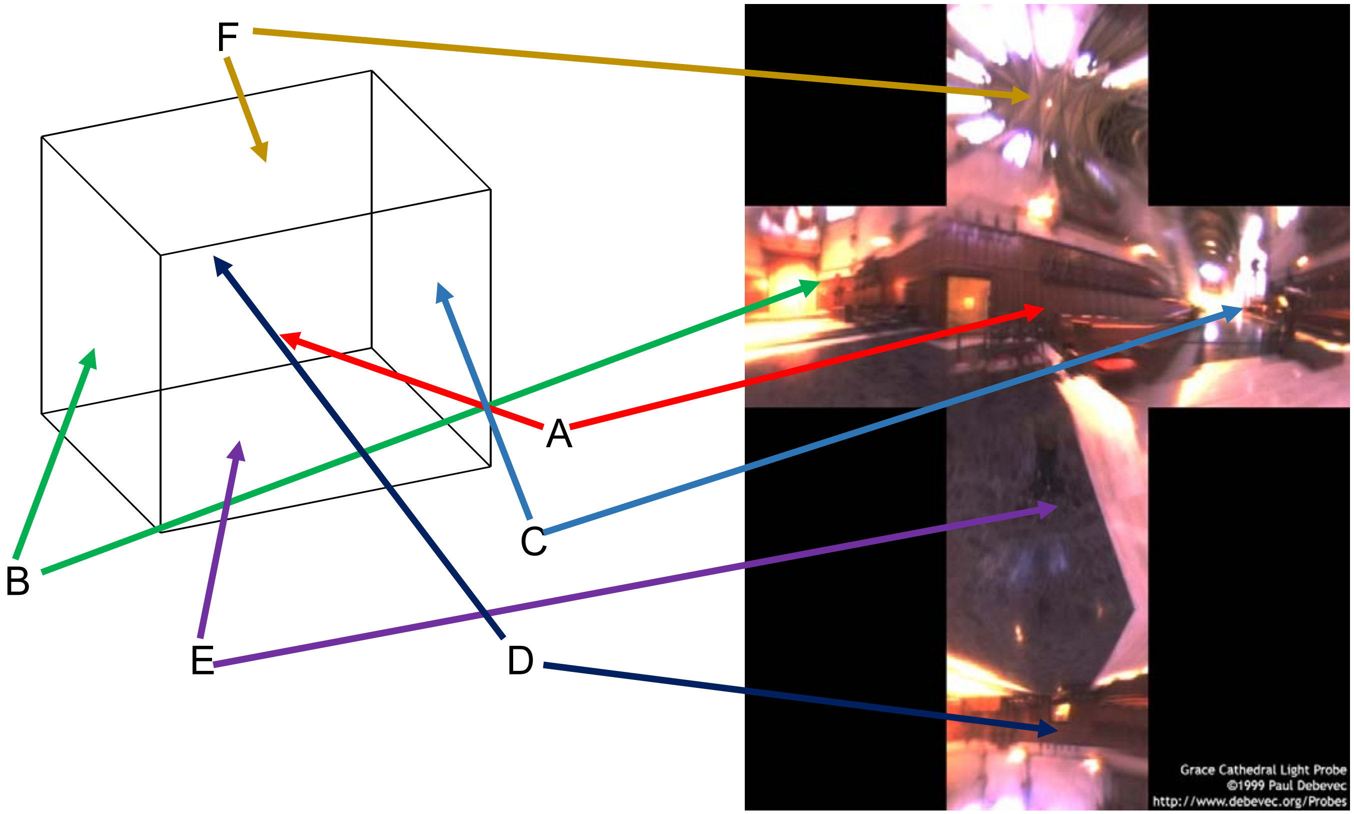
- texture is distorted fish-eye view
  - point camera at mirrored sphere
  - spherical texture mapping creates texture coordinates that correctly index into this texture map

CUBE MAPPING

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



CUBE MAPPING



CUBE MAPPING

- direction of reflection vector *r* 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.
- difficulty in interpolating across faces

CUBE MAPPING

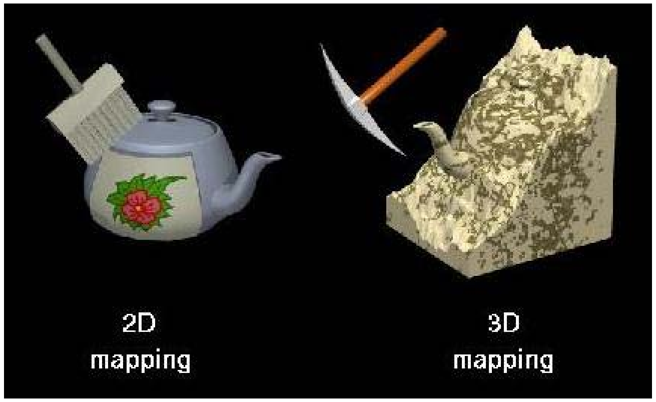
how to calculate?

- direction of reflection vector r 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.
- difficulty in interpolating across faces

ENVIRONMENT MAPS (EM)

- in theory, every object should have a separate EM
- in theory, every time something moves, you should re-compute EM
- "you'll be surprised at what you can get away with"

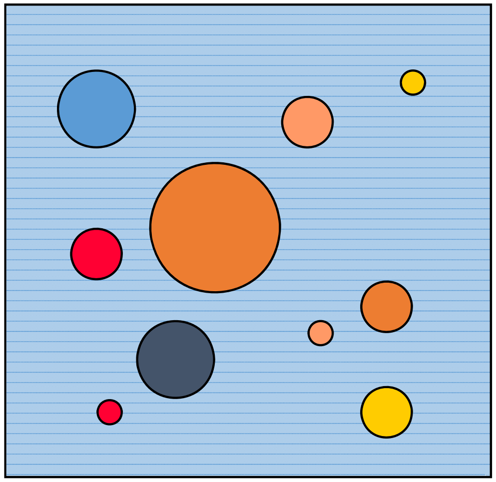
VOLUMETRIC TEXTURE

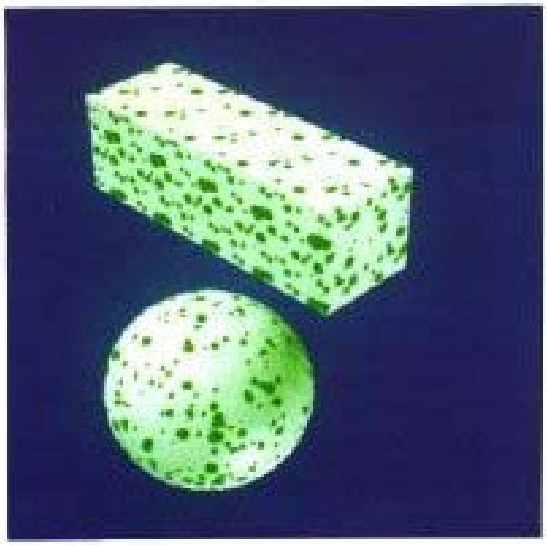


- define texture pattern over 3D domain - 3D space containing the object
- texture function can be digitized or procedural
- for each point on object compute texture from point location in space
- e.g., ShaderToy
- computing is cheap, memory access not as much

PROCEDURAL TEXTURE EFFECTS: BOMBING

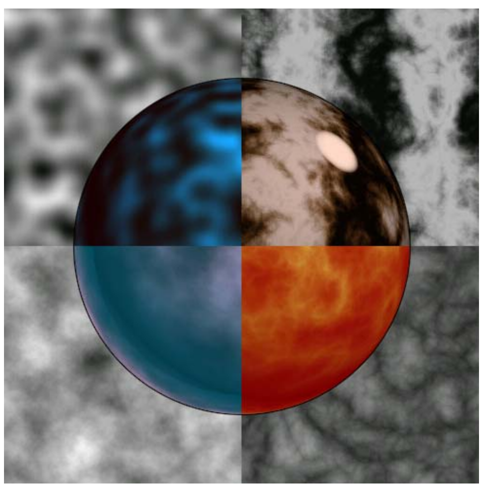
- randomly drop bombs of various shapes, sizes and orientation into texture space (store data in table)
  - for point P search table and determine if inside shape
    - if so, color by shape's color
    - otherwise, color by object's color

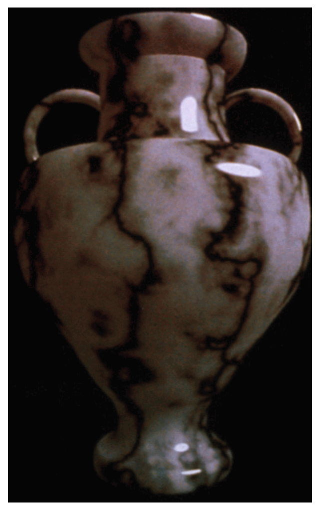


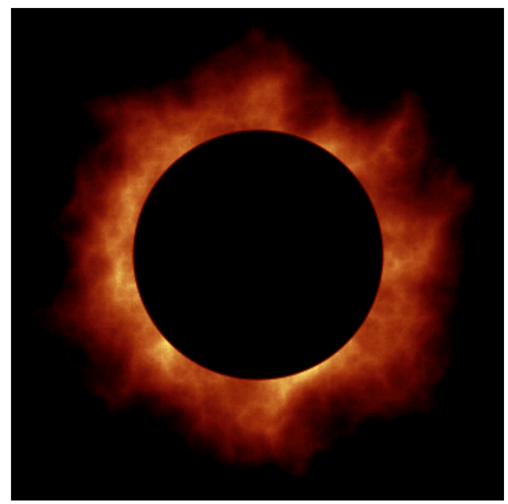


PERLIN NOISE: PROCEDURAL TEXTURES

- 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

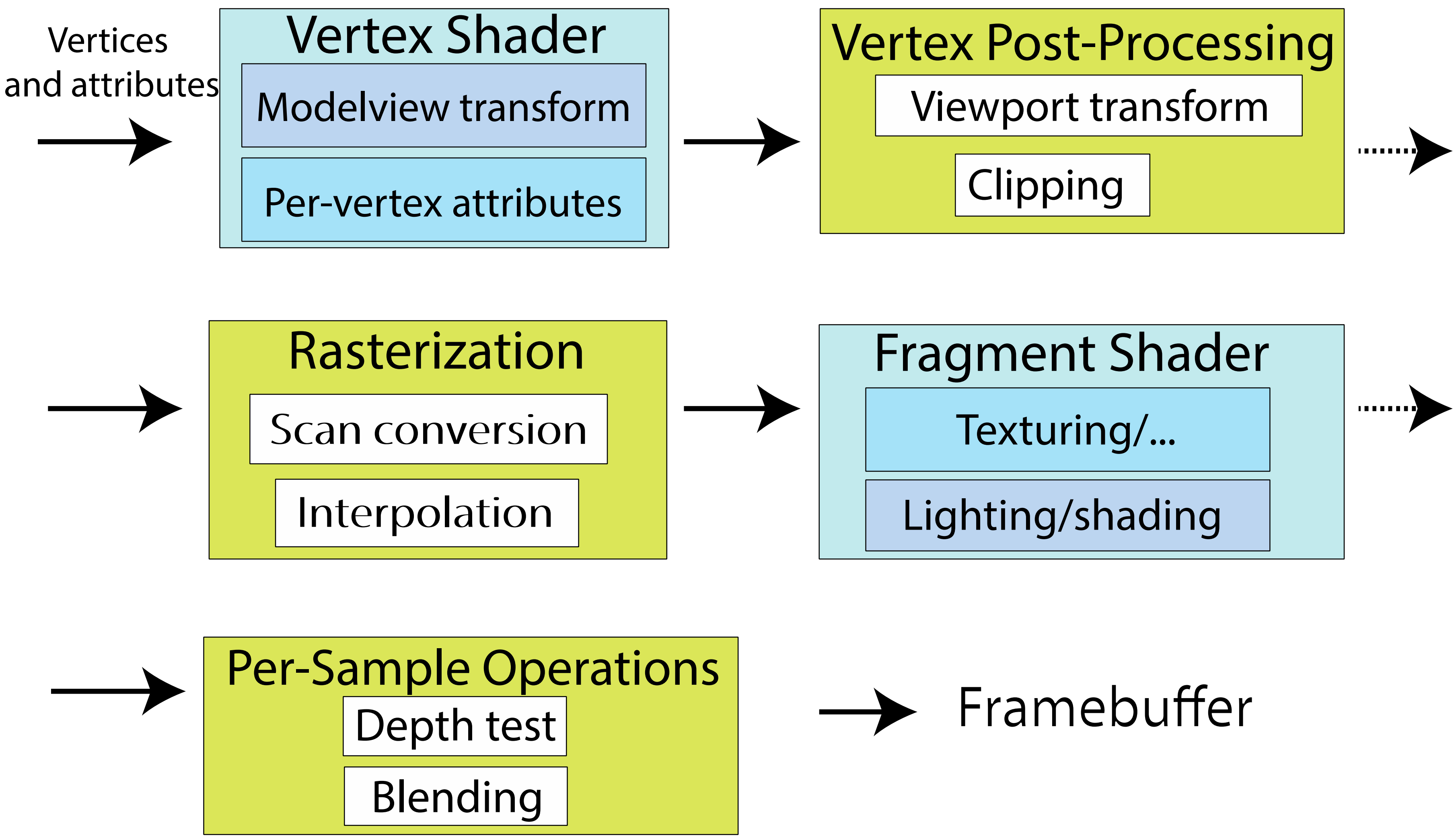




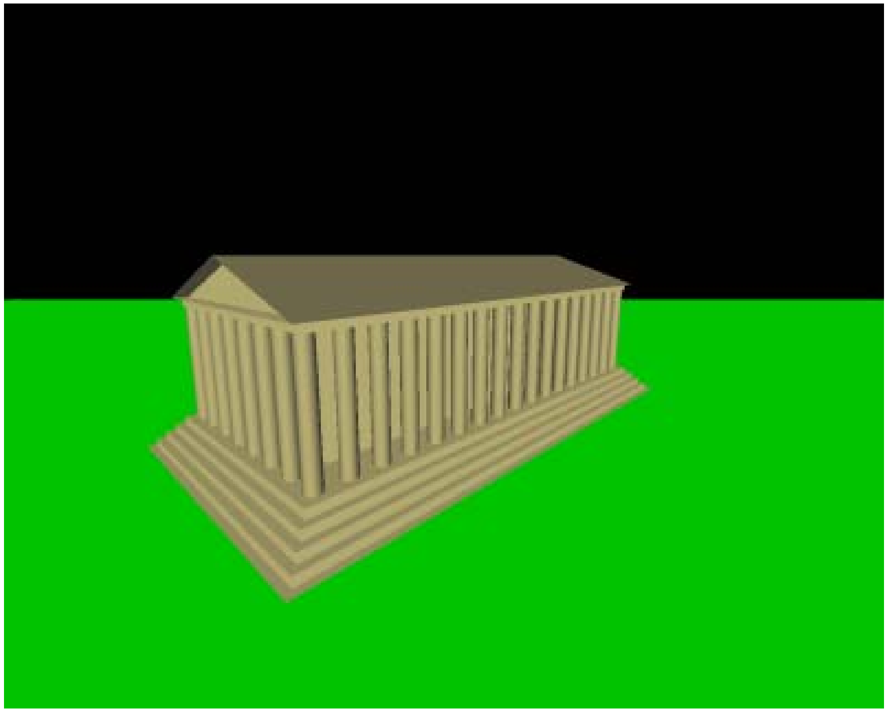


http://mrl.nyu.edu/~perlin/planet/

THE RENDERING PIPELINE



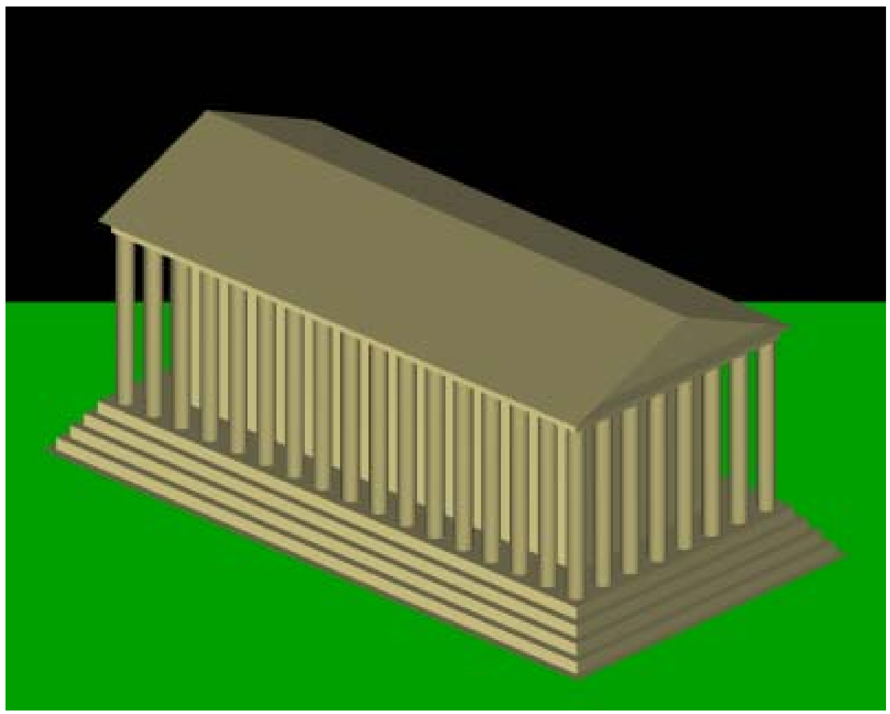
SHADOWS



SHADOWS

Need at least 2 shader passes:

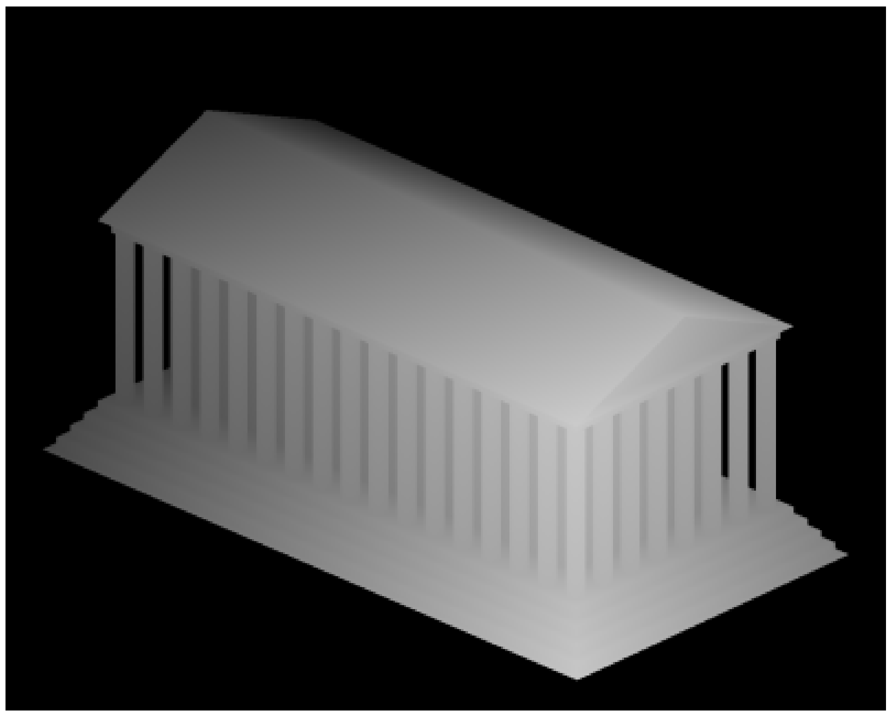
1. Draw everything as it's viewed from the LIGHT SOURCE



SHADOW MAPPING

Need at least 2 shader passes:

1. Draw everything as it’s viewed from the LIGHT SOURCE
   Depth per pixel (‘depth map’)

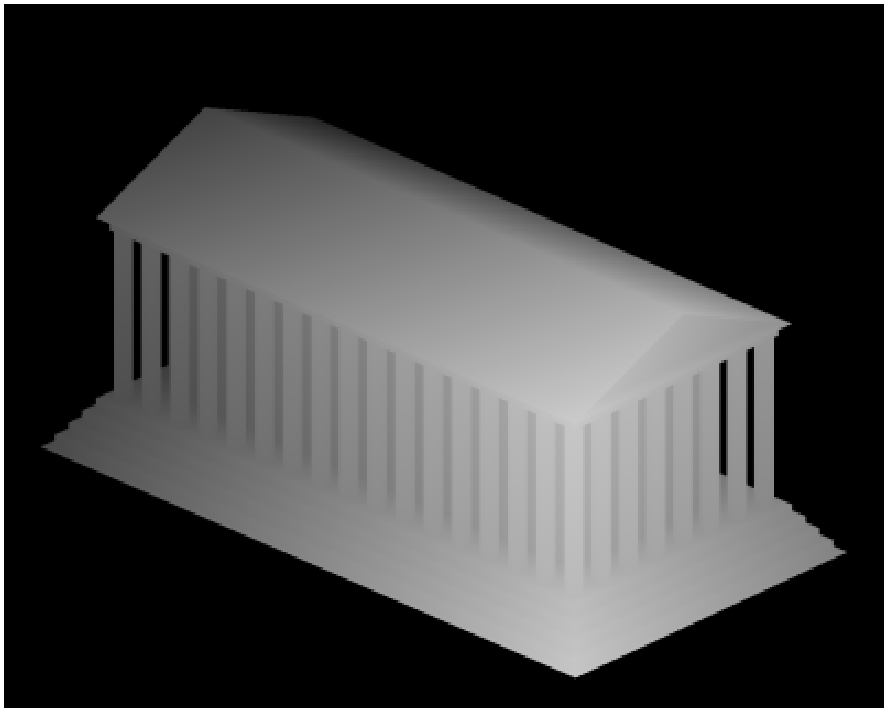


SHADOW MAPPING

Need at least 2 shader passes:

1. Draw everything as it's viewed from the LIGHT SOURCE
   Depth per pixel ('depth map')

How?



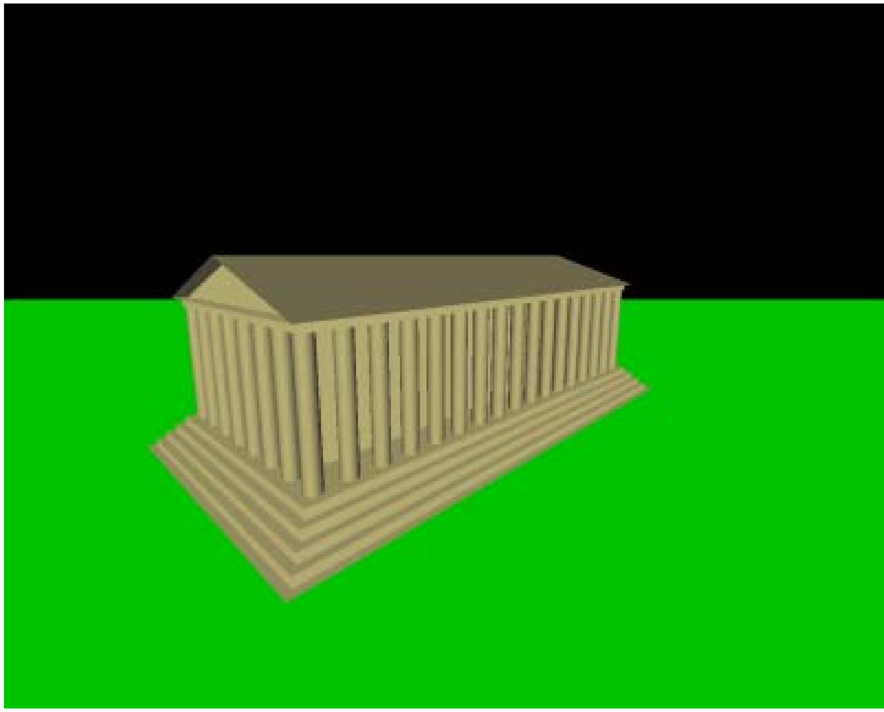
SHADOWS (IDEA)

Need at least 2 shader passes:

1. Draw everything as it's viewed from the LIGHT SOURCE
   Depth per pixel ('depth map').
2. Now draw everything from CAMERA
   When computing color per pixel:

   - Find corresponding depth map pixel:
     D - distance from light source

   - Is distance from me to the camera > D?
     - Yes: I am occluded! I'm in SHADOW.
     - No: I'm LIT!



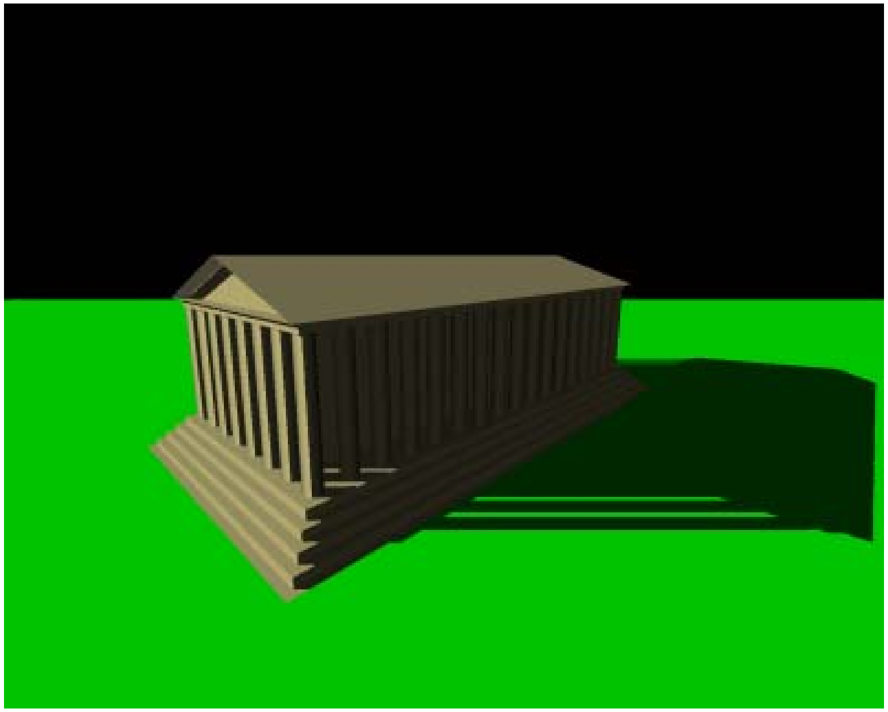
SHADOWS (IDEA)

Need at least 2 shader passes:

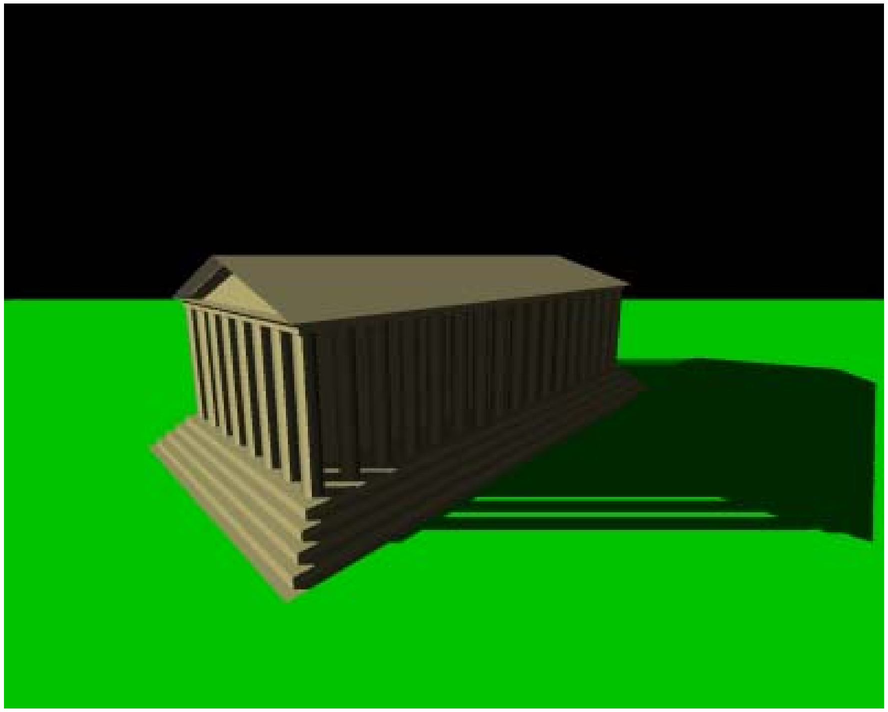
1. Draw everything as it's viewed from the LIGHT SOURCE
   Depth per pixel ('depth map').
2. Now draw everything from CAMERA
   When computing color per pixel:

   - Find corresponding depth map pixel:
     D - distance from light source

   - Is distance from me to the camera > D?
     - Yes: I am occluded! I'm in SHADOW.
     - No: I'm LIT!

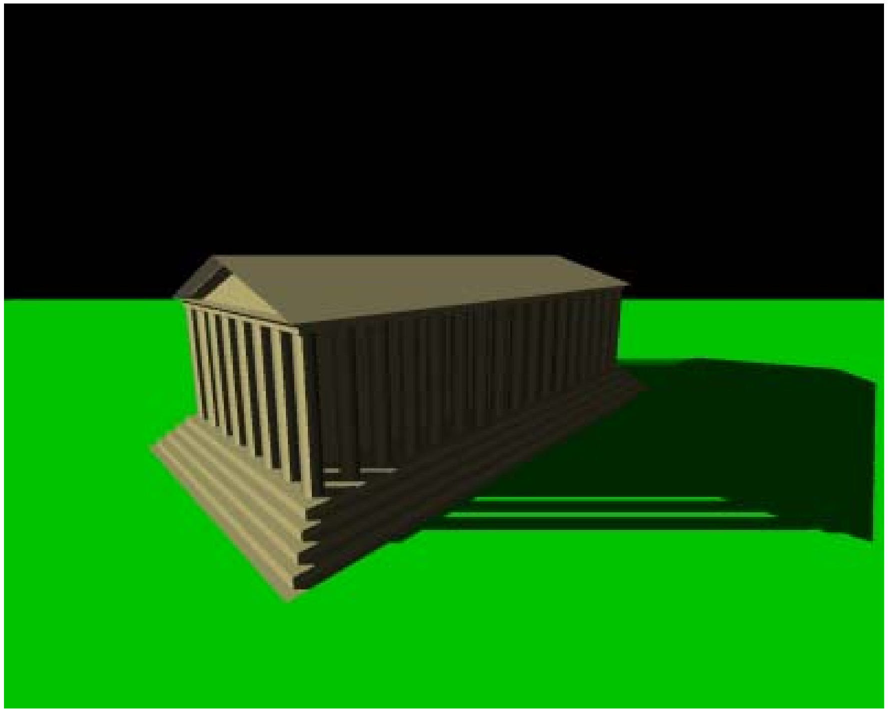


PROBLEMS OF SHADOW MAPPING



PROBLEMS OF SHADOW MAPPING

- Hard shadow edges
  - Can be solved by several shadow map lookups



PROBLEMS OF SHADOW MAPPING

- Hard shadow edges
  - Do several shadow map lookups
- Shadow aliasing
  - Increase shadow map resolution
- Many variations of shadow mapping try to solve those problems