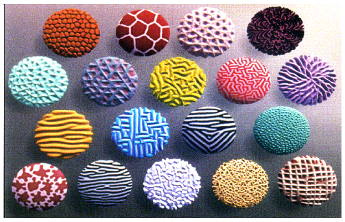
Texture Mapping

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

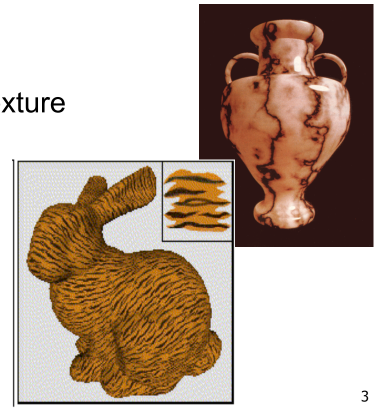


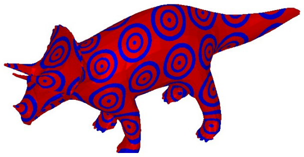
Texture Mapping

- 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

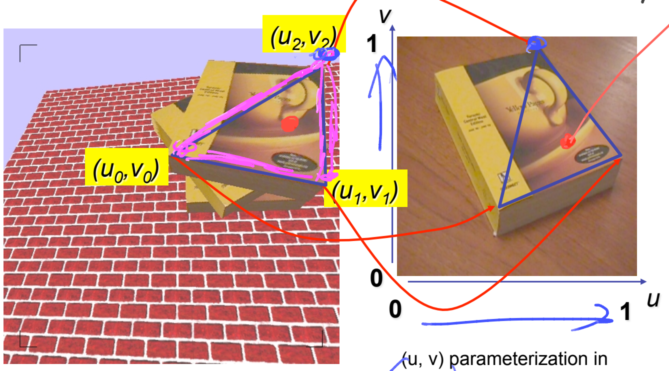
- define color (RGB) for each point on object surface
- from an image:
  - surface texture map
  - affine or projective texture
- other:
  - volumetric texture
  - procedural texture





Texture Mapping

rendered scene. (or t) texture map. (u,v) = ?

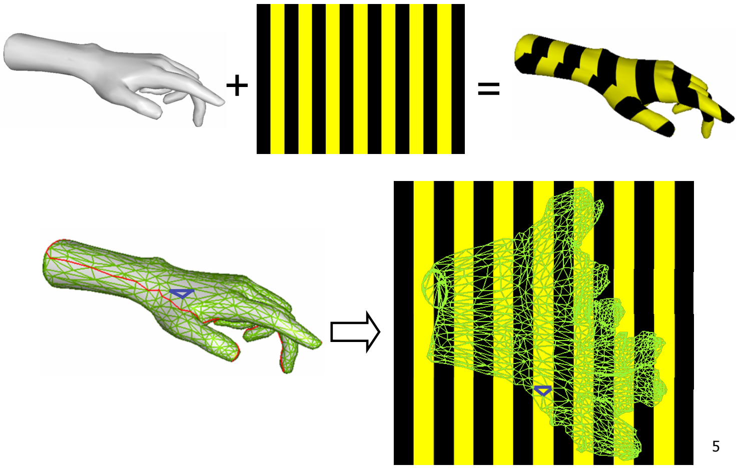


(or s)

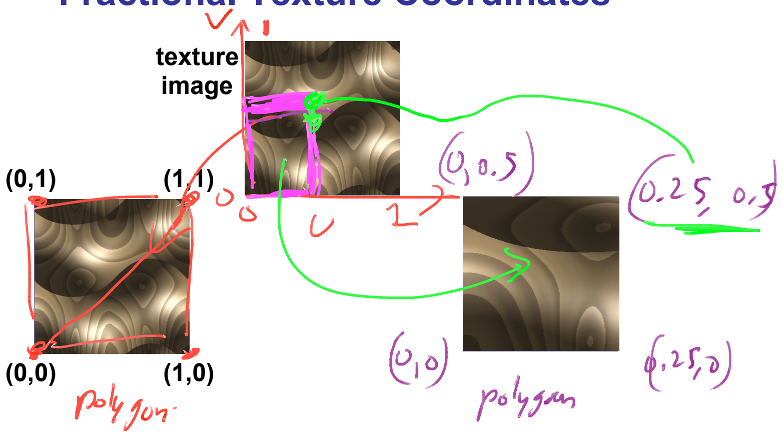
texture coords (u,v) are vertex attributes

(u, v) parameterization in OpenGL
(s,t)

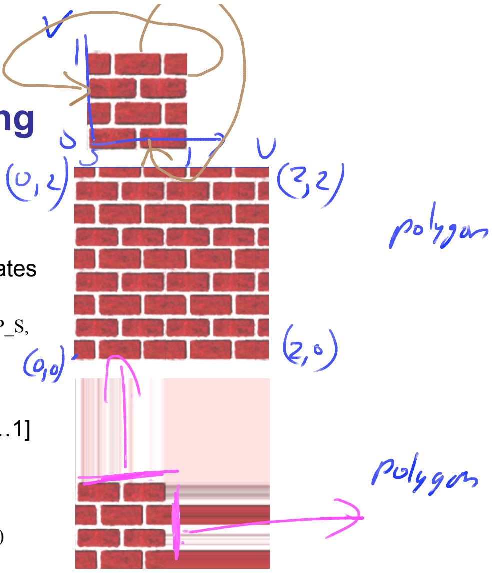
Texture Mapping Example



Fractional Texture Coordinates



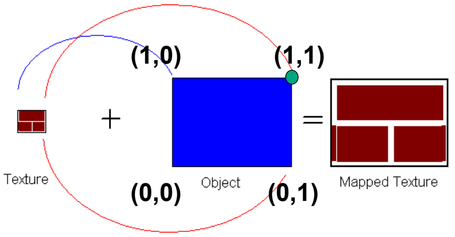
Texture Lookup: Tiling and Clamping

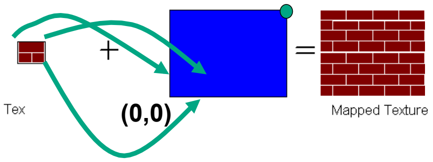


- What if s or t is outside [0…1] ?
- Multiple choices
  - Use fractional part of texture coordinates
    - Cyclic repetition glTexParameteri(…, GL_TEXTURE_WRAP_S, GL_REPEAT, GL_TEXTURE_WRAP_T, GL_REPEAT, ...)
  - Clamp every component to range [0…1]
    - Re-use color values from texture image border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

Specified in "Sampler"

Tiled Texture Map



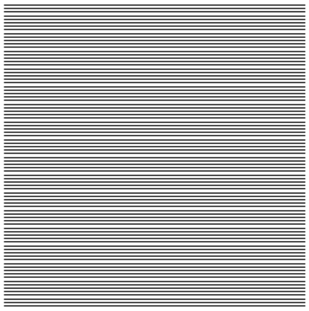


Texture Objects and Binding

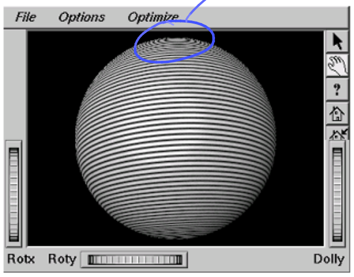
- texture object
  - an OpenGL data type that keeps textures resident in memory and provides identifiers to easily access them
  - provides efficiency gains over having to repeatedly load and reload a texture
  - various strategies for managing texture memory and texture cache
- texture binding
  - which texture to use right now
  - switch between preloaded textures

Reconstruction

texture map



aliasing artifacts
"moire" pattern.

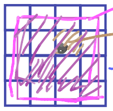


(image courtesy of Kiriakos Kutulakos, U Rochester)

Reconstruction

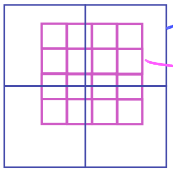
- how to deal with:
  - pixels that are much larger than texels?

"minification"
pixels are larger than texels



  - pixels that are much smaller than texels ?

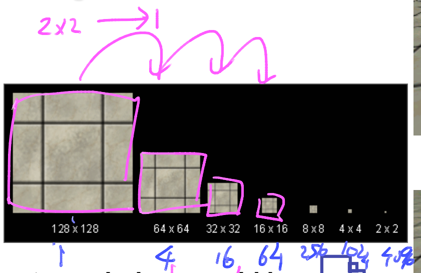
"magnification"
pixels are smaller than texels



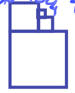
MIPmapping

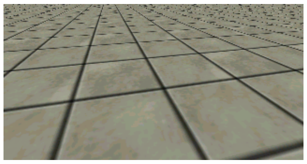
multim in parvo : latin for many in one place

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

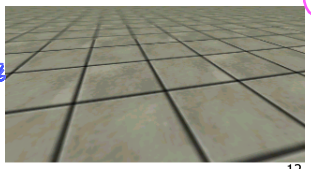


store whole pyramid in single block of memory





Without MIP-mapping



With MIP-mapping

Using the MIPMAP

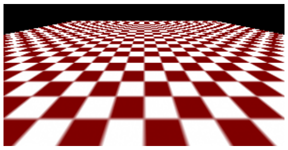
Strategy A: NEAREST_MIPMAP
- compute how many texels a pixel covers
- choose best MIPMAP level
- grab texel from that level

B LINEAR MIPMAP
- compute # texels a pixel covers
- ~~choose~~ find 2 neighboring MIPMAP levels
- lookup texture & interpolate

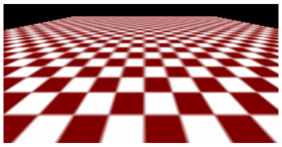
MIPmaps

- **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
- `gluBuild2DMipmaps`
  - automatically constructs a family of textures from original texture size down to 1x1

without

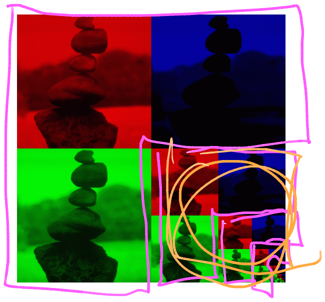


with



MIPmap storage

- only $\frac{1}{3}$ more space required



$$\boxed{1} + \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \cdots$$

$$1 + r + r^2 + r^3 + \cdots$$

where $r = \frac{1}{4}$

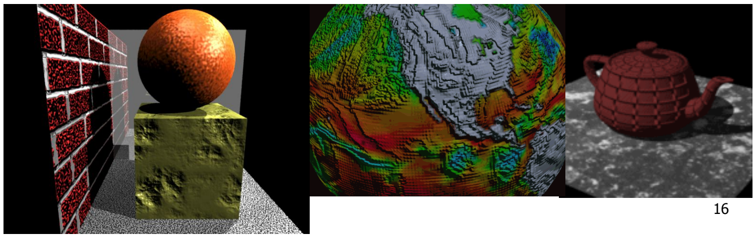
$$\text{sum} = \frac{1}{1-r} = \frac{1}{1-\frac{1}{4}} = \frac{1}{\frac{3}{4}} = \boxed{\frac{4}{3}}$$

Other uses for Textures

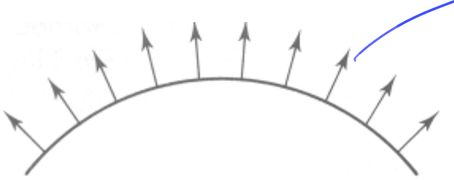
- usually provides colour, but …
- can also use to control other material/object properties
  - surface normal (bump mapping)
  - reflected color (environment mapping)

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



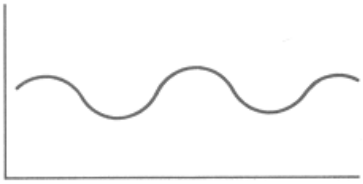
Bump Mapping



original surface normals

$O(u)$

Original surface

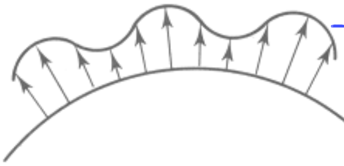


$B(u)$: Scalar height field.

A bump map stored as an image

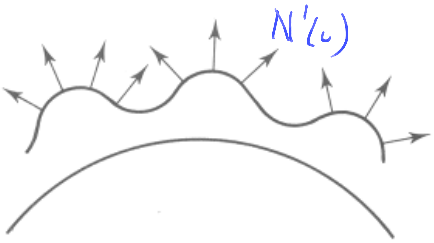
Idea: use the bump map to compute alterations to the surface normals.

Bump Mapping



$O'(u) = O(u) + \vec{N}\ \underline{B(u)}$

Lengthening or shortening $O(u)$ using $B(u)$

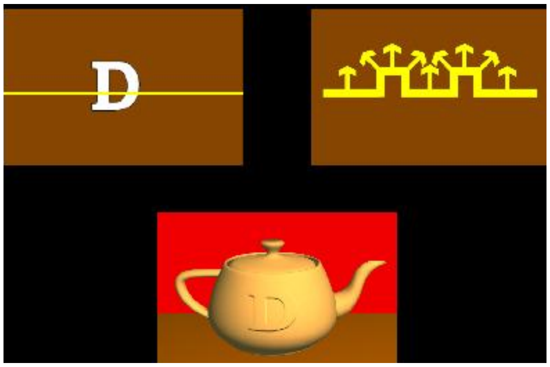


$N'(u)$ = normals for the new surface $O'(u)$

The vectors to the 'new' surface

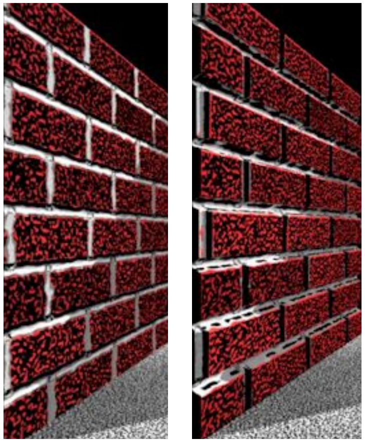
Embossing

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



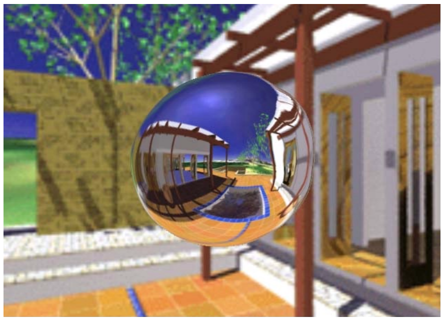
Displacement Mapping

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



Environment Mapping

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

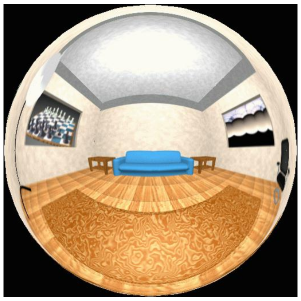


Environment Mapping

- 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

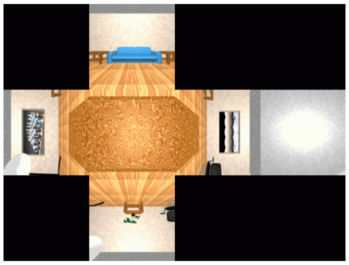
Sphere Mapping

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

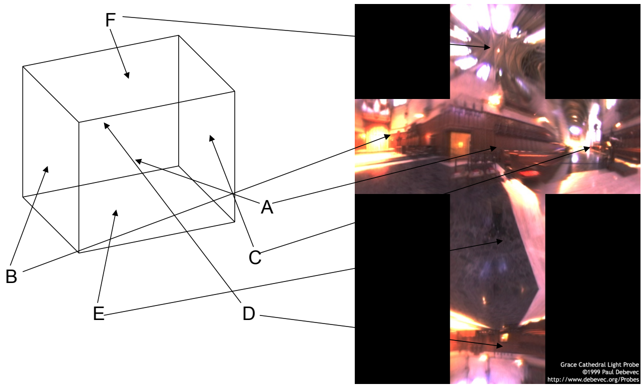


Cube Mapping

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



Cube Mapping

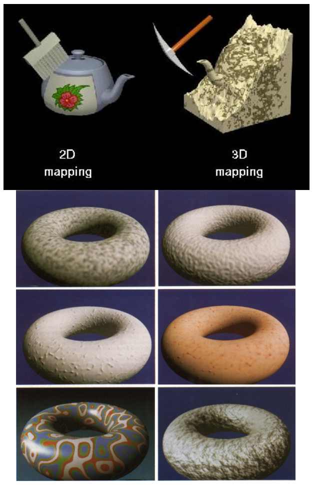


Cube Mapping

- 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 (normalized by the 3rd coordinate) selects the pixel from the face.
    - e.g., (-0.2, 0.5) gets mapped to (0.38, 0.80).
- difficulty in interpolating across faces

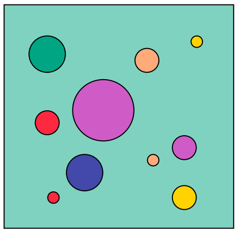
Volumetric Texture

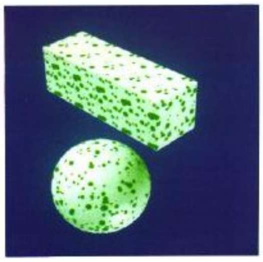
- define texture pattern over 3D domain - 3D space containing the object
  - texture function can be digitized or procedural
  - for each point on object compute texture from point location in space
- e.g., ShaderToy
- computing is cheap, memory access is expensive !



Procedural Texture Effects: Bombing

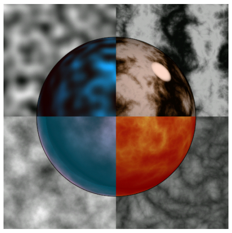
- randomly drop bombs of various shapes, sizes and orientation into texture space (store data in table)
  - for point P search table and determine if inside shape
    - if so, color by shape
    - otherwise, color by objects color

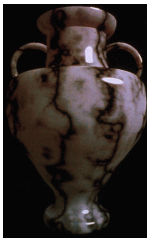




Perlin Noise: Procedural Textures

- 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

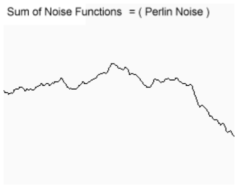


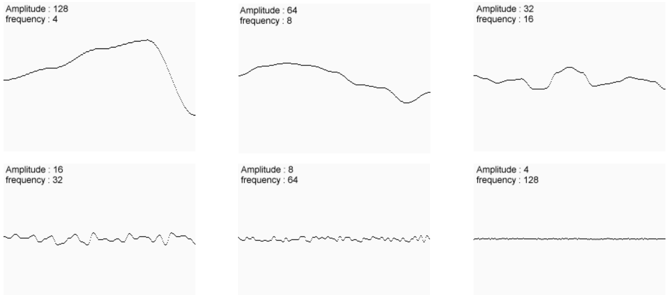


http://mrl.nyu.edu/~perlin/planet/

Perlin Noise: Turbulence

- multiple feature sizes
  - add scaled copies of noise





Perlin Noise: Turbulence

- multiple feature sizes
    - add scaled copies of noise