University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2007

Tamara Munzner

Textures I

Week 9, Wed Mar 14

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2007

Reading for Today and Next Time

- FCG Chap 11 Texture Mapping
  - except 11.8
- RB Chap Texture Mapping
- FCG Sect 16.6 Procedural Techniques
- FCG Sect 16.7 Groups of Objects

News

- Q3 specular color should be (1,1,0)
- P3: bug in sample implementation fixed
  - new reference images and sample binaries posted
  - no change to template

Correction: HSV and RGB

- HSV/HSI conversion from RGB
  - not expressible in matrix

$$I = \frac{R+G+B}{3} \qquad S = 1 - \frac{\boxed{\min(R,G,B)}}{I}$$

$$H = \cos^{-1}\left[\frac{\frac{1}{2}\left[(R-G)+(R-B)\right]}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right]$$

Review: Z-Buffer Algorithm

- augment color framebuffer with Z-buffer or depth buffer which stores Z value at each pixel
  - at frame beginning, initialize all pixel depths to ∞
  - when rasterizing, interpolate depth (Z) across polygon
  - check Z-buffer before storing pixel color in framebuffer and storing depth in Z-buffer
  - don’t write pixel if its Z value is more distant than the Z value already stored there

Clarification/Review: Depth Test Precision

- reminder: projective transformation maps eye-space *z* to generic *z*-range (NDC)

$$\begin{bmatrix} x_N \\ y_N \\ z_N \\ w_N \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_E \\ y_E \\ z_E \\ w_E \end{bmatrix}$$

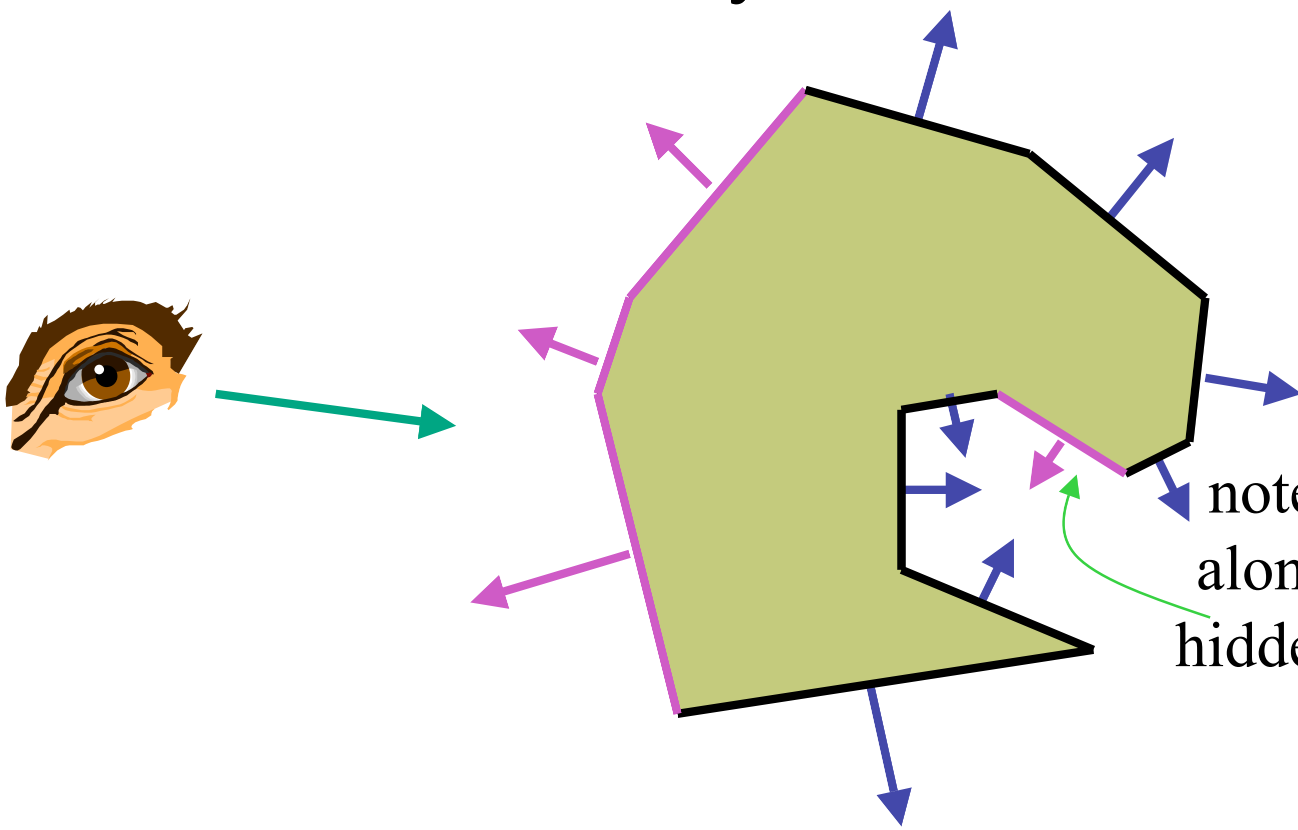
- thus $z_N \sim= 1/z_E$

$$z_N = \frac{-(f+n)}{f-n} z_E + \frac{-2fn}{f-n} w_E, \; w_N = -z_E \qquad \frac{z_N}{w_N} = \frac{f+n}{f-n} + \frac{2fn}{f-n} \frac{w_E}{\boxed{z_E}}$$

Backface Culling

Back-Face Culling

- on the surface of a "solid" object, polygons whose normals point away from the camera are always occluded:

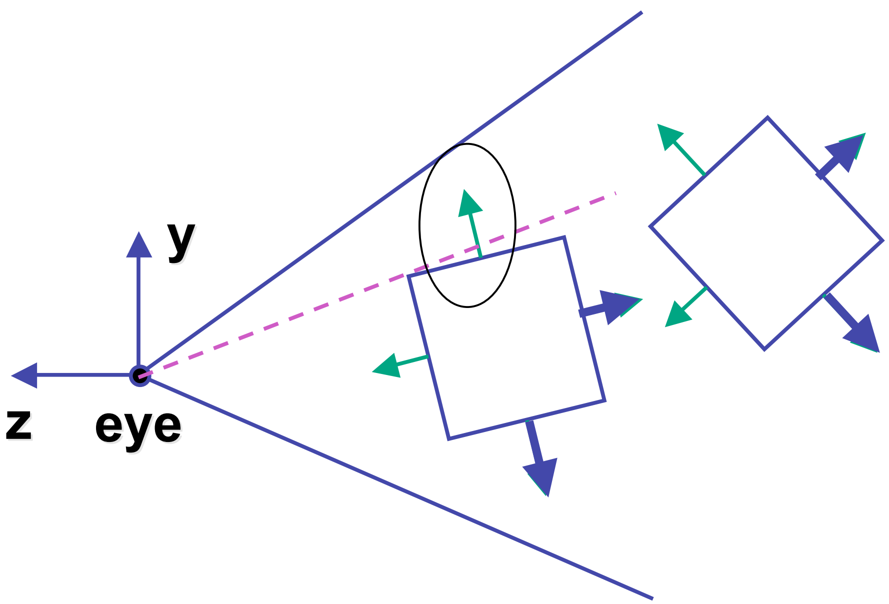


note: backface culling alone doesn't solve the hidden-surface problem!

Back-Face Culling

- not rendering backfacing polygons improves performance
  - by how much?
    - reduces by about half the number of polygons to be considered for each pixel
  - optimization when appropriate

Back-face Culling: VCS



first idea:

cull if $N_Z < 0$

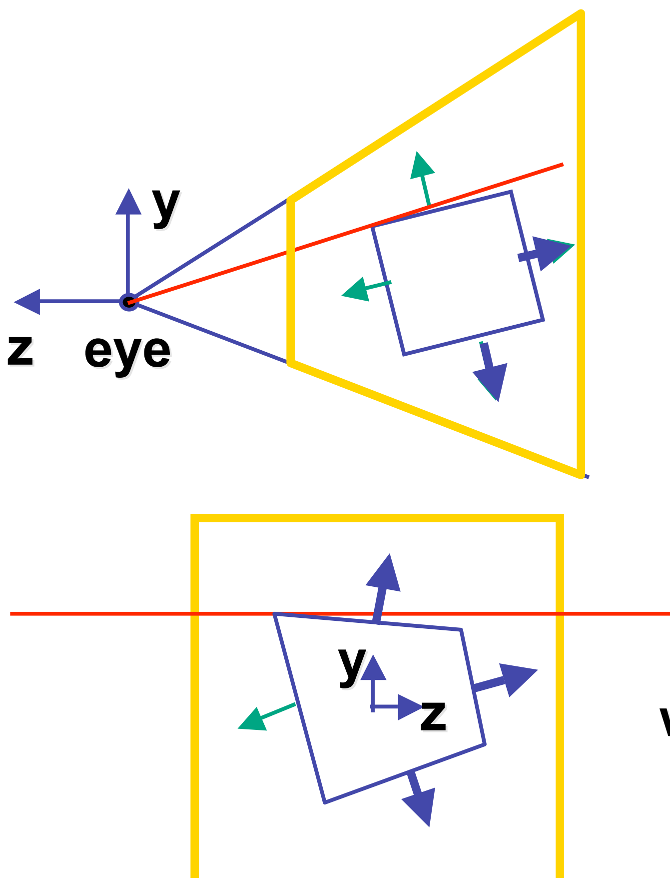
sometimes
misses polygons that
should be culled

Back-face Culling: NDCS

VCS

NDCS

eye



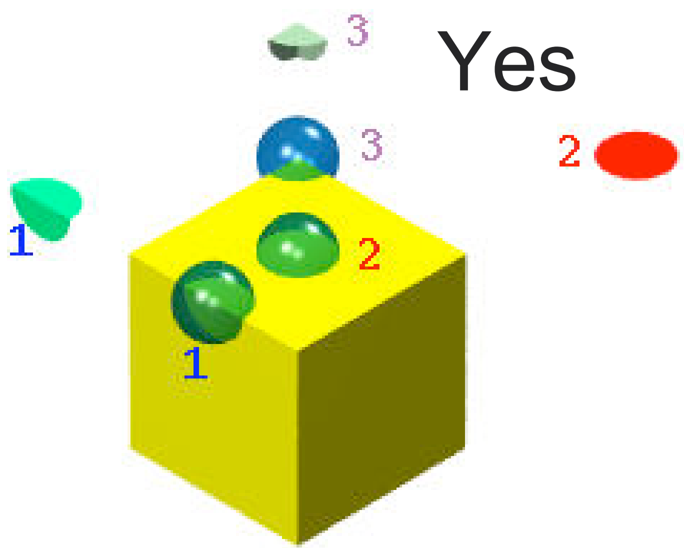
works to cull if $N_Z > 0$

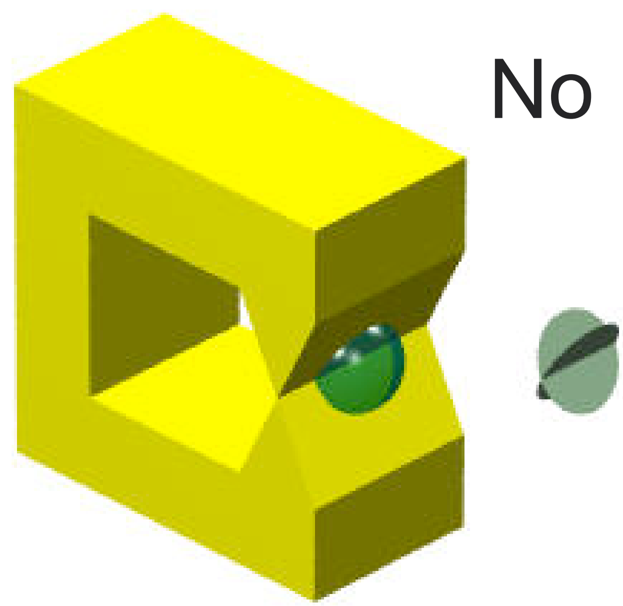
Back-Face Culling: Manifolds

- most objects in scene are typically "solid"
- specifically: orientable closed manifolds
  - orientable: must have two distinct sides
    - cannot self-intersect
    - a sphere is orientable since has two sides, 'inside' and 'outside'.
    - a Mobius strip or a Klein bottle is not orientable
  - closed: cannot "walk" from one side to the other
    - sphere is closed manifold
    - plane is not

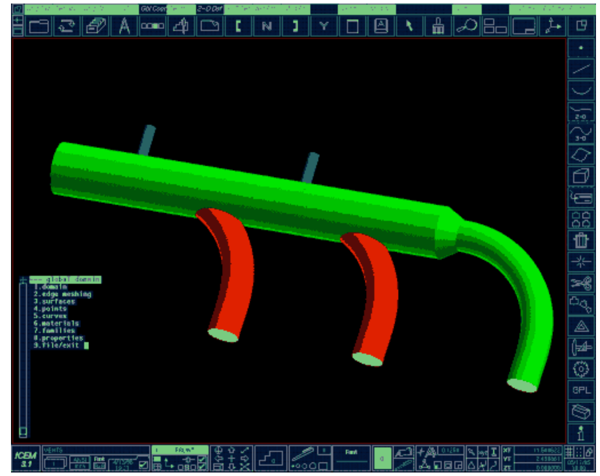
Back-Face Culling: Manifolds

- most objects in scene are typically "solid"
- specifically: orientable closed manifolds
  - manifold: local neighborhood of all points isomorphic to disc
  - boundary partitions space into interior & exterior





Backface Culling: Manifolds



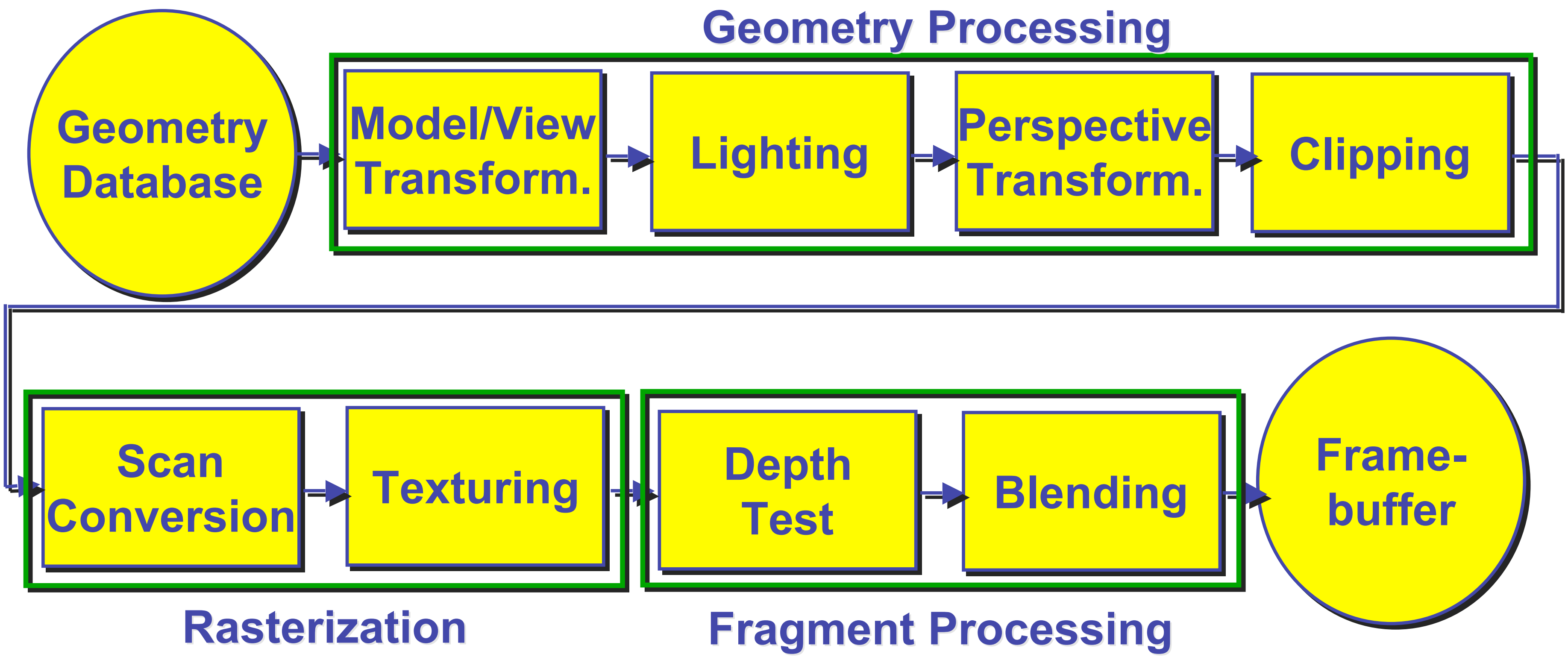
- examples of manifold objects:
  - sphere
  - torus
  - well-formed CAD part
- examples of non-manifold objects:
  - a single polygon
  - a terrain or height field
  - polyhedron w/ missing face
  - anything with cracks or holes in boundary
  - one-polygon thick lampshade

Invisible Primitives

- *why might a polygon be invisible?*
  - polygon outside the *field of view / frustum*
    - solved by clipping
  - polygon is *backfacing*
    - solved by backface culling
  - polygon is *occluded* by object(s) nearer the viewpoint
    - solved by hidden surface removal

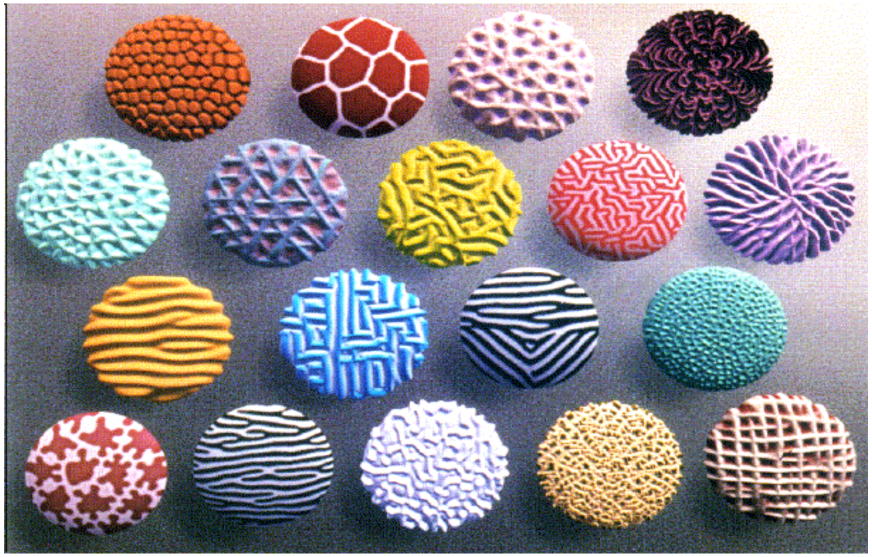
Texturing

Rendering Pipeline



Texture Mapping

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

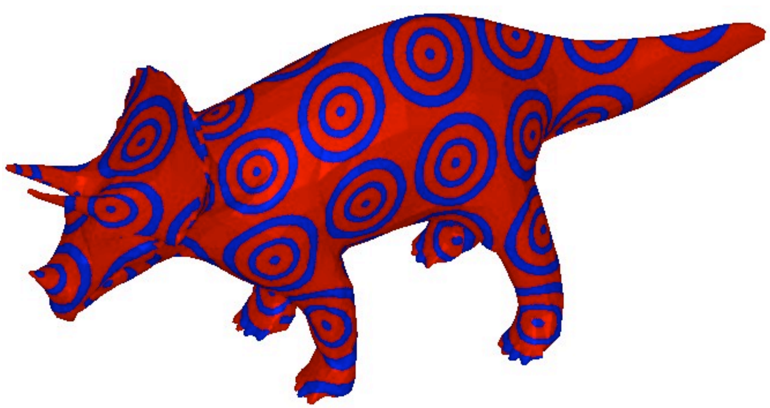


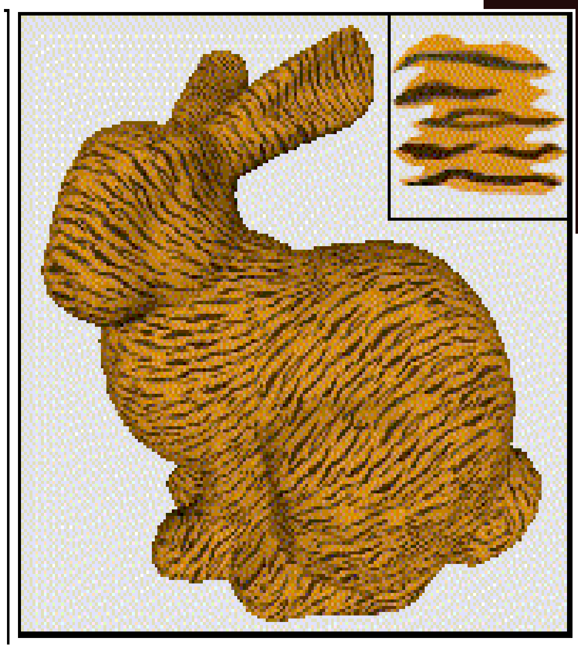
Texture Mapping

- introduced to increase realism
  - lighting/shading models not enough
- hide geometric simplicity
  - images convey illusion of geometry
  - map a brick wall texture on a flat polygon
  - create bumpy effect on surface
- associate 2D information with 3D surface
  - point on surface corresponds to a point in texture
  - “paint” image onto polygon

Color Texture Mapping

- define color (RGB) for each point on object surface
- two approaches
  - surface texture map
  - volumetric texture

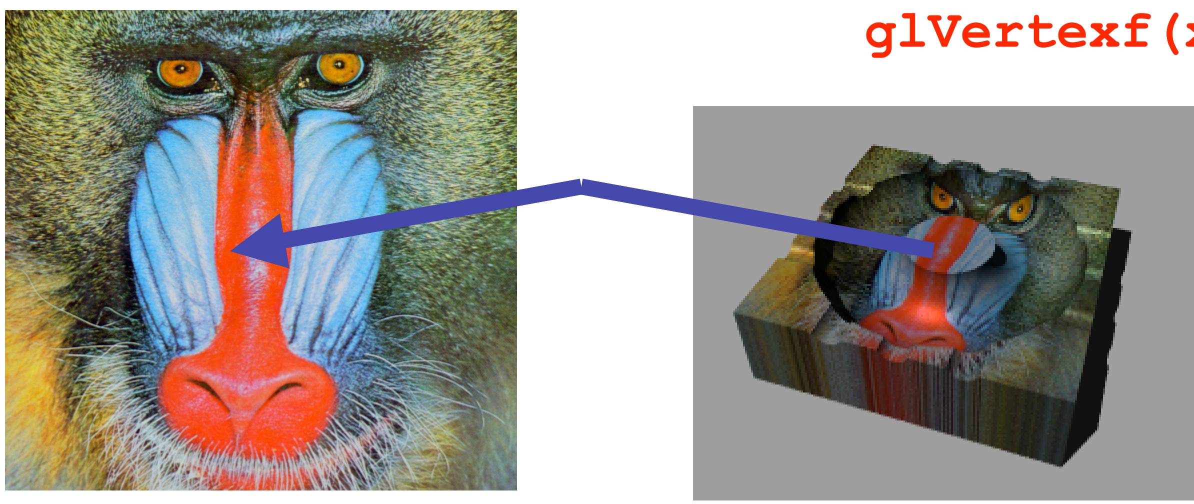




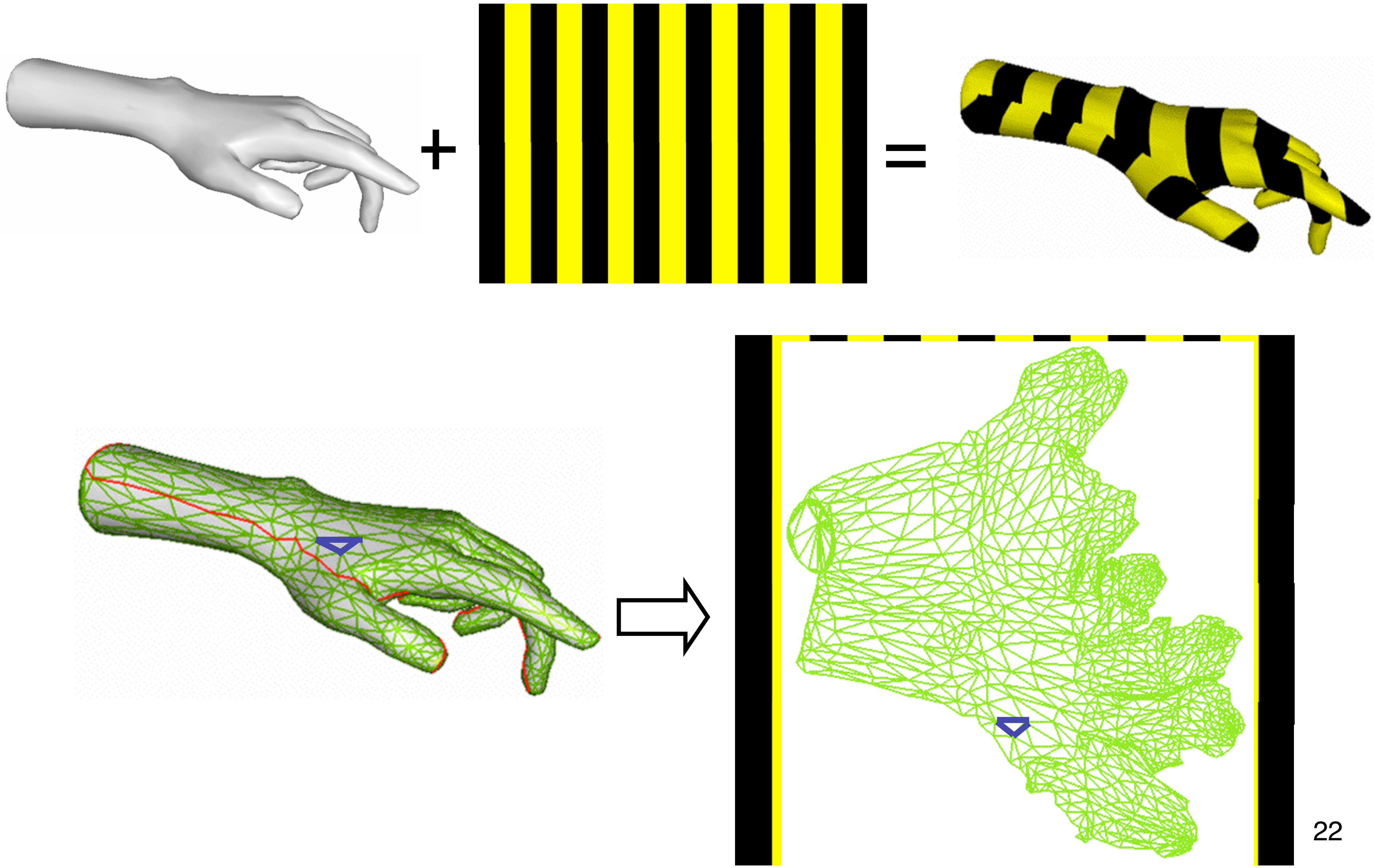
Texture Coordinates

- texture image: 2D array of color values (texels)
- assigning texture coordinates (s,t) at vertex with object coordinates (x,y,z,w)
  - use interpolated (s,t) for texel lookup at each pixel
  - use value to modify a polygon’s color
    - or other surface property
  - specified by programmer or artist

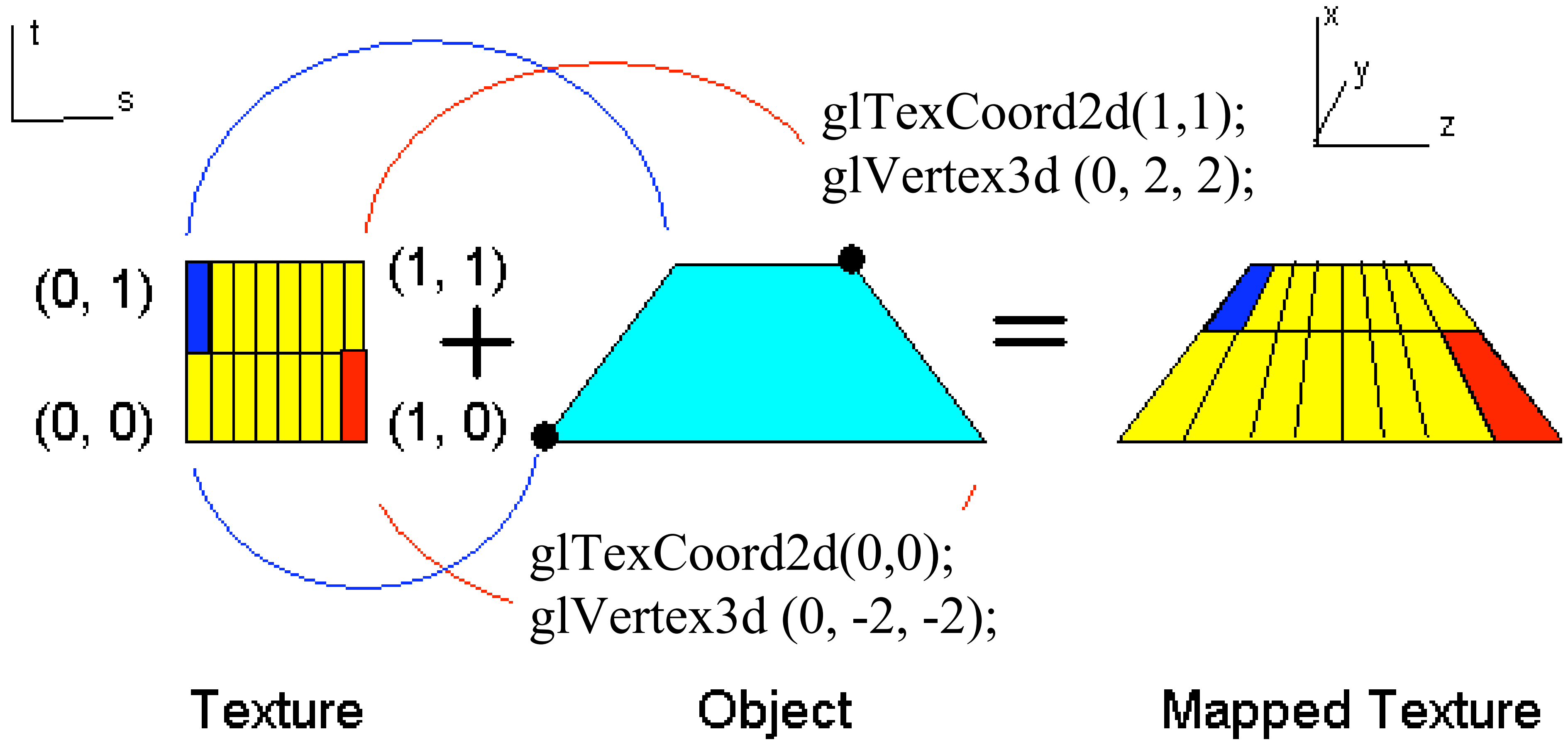
```
glTexCoord2f(s,t)
glVertexf(x,y,z,w)
```



Texture Mapping Example

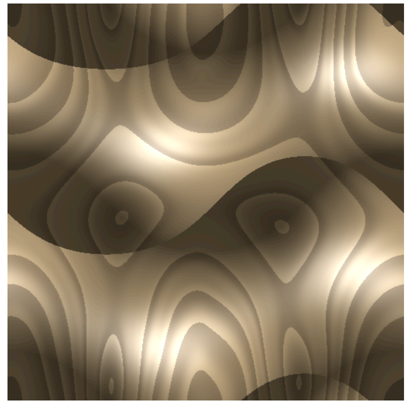


Example Texture Map

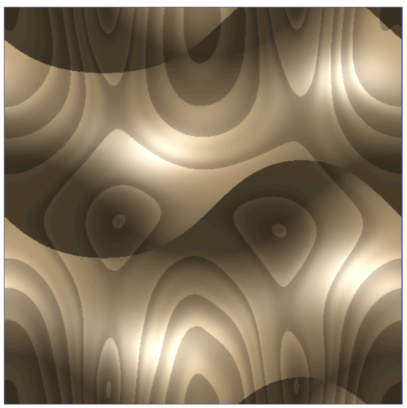


Fractional Texture Coordinates

texture
image

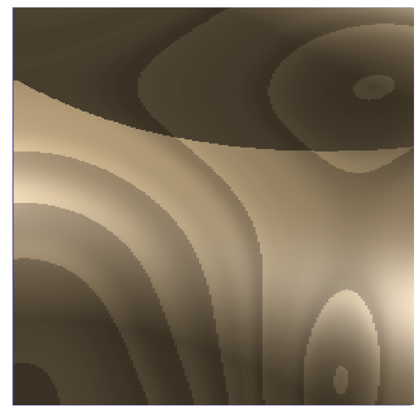


(0,1) (1,1)



(0,0) (1,0)

(0,.5) (.25,.5)



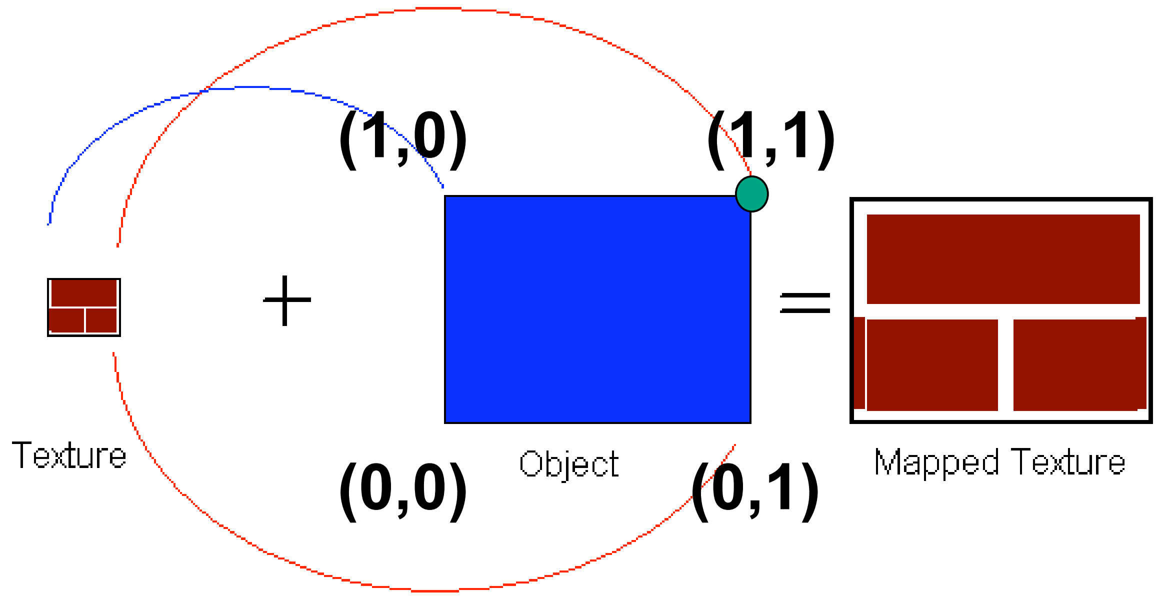
(0,0) (.25,0)

Texture Lookup: Tiling and Clamping

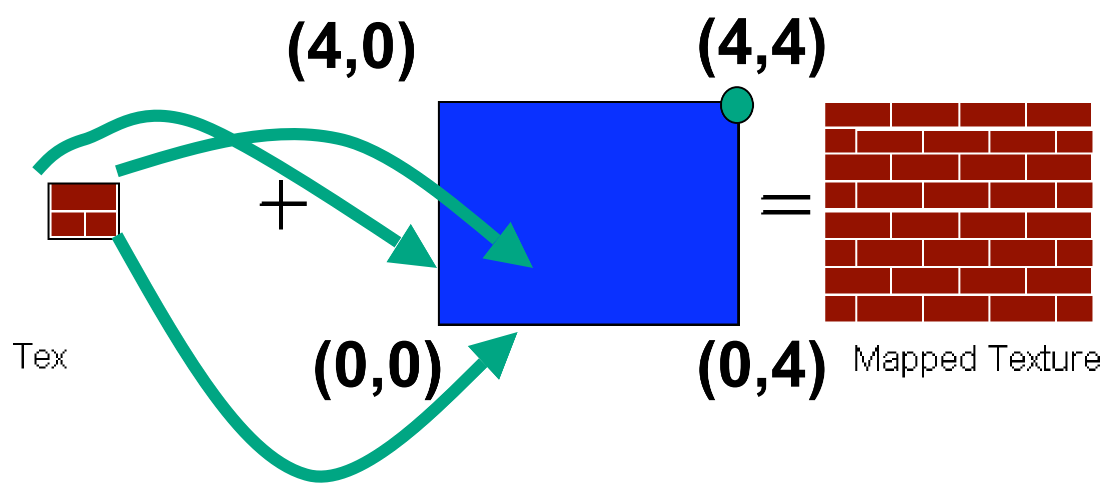
- what if s or t is outside the interval [0…1]?
- multiple choices
  - use fractional part of texture coordinates
    - cyclic repetition of texture to tile whole surface
      glTexParameteri(…, GL_TEXTURE_WRAP_S, GL_REPEAT, GL_TEXTURE_WRAP_T, GL_REPEAT, ...)
  - clamp every component to range [0…1]
    - re-use color values from texture image border
      glTexParameteri(…, GL_TEXTURE_WRAP_S, GL_CLAMP, GL_TEXTURE_WRAP_T, GL_CLAMP, ...)

Tiled Texture Map

glTexCoord2d(1, 1);
glVertex3d (x, y, z);



glTexCoord2d(4, 4);
glVertex3d (x, y, z);



Demo

- Nate Robbins tutors
  - texture

Texture Coordinate Transformation

- motivation
  - change scale, orientation of texture on an object
- approach
  - *texture matrix stack*
  - transforms specified (or generated) tex coords

```
glMatrixMode( GL_TEXTURE );
glLoadIdentity();
glRotate();
```

    ...

  - more flexible than changing (s,t) coordinates
- [demo]

Texture Functions

- once have value from the texture map, can:
  - directly use as surface color: `GL_REPLACE`
    - throw away old color, lose lighting effects
  - modulate surface color: `GL_MODULATE`
    - multiply old color by new value, keep lighting info
    - texturing happens **after** lighting, not relit
  - use as surface color, modulate alpha: `GL_DECAL`
    - like replace, but supports texture transparency
  - blend surface color with another: `GL_BLEND`
    - new value controls which of 2 colors to use
    - indirection, new value not used directly for coloring
- specify with `glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, <mode>)`
- [demo]

Texture Pipeline

(x, y, z)

Object position

(-2.3, 7.1, 17.7)

(s, t)

Parameter space

(0.32, 0.29)

(s', t')

Transformed parameter space

(0.52, 0.49)

Texel space

(81, 74)

Texel color

(0.9,0.8,0.7)

Object color

(0.5,0.5,0.5)

Final color

(0.45,0.4,0.35)

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
  - you can prioritize textures to keep in memory
  - OpenGL uses least recently used (LRU) if no priority is assigned
- texture binding
  - which texture to use right now
  - switch between preloaded textures

Basic OpenGL Texturing

- create a texture object and fill it with texture data:
  - `glGenTextures(num, &indices)` to get identifiers for the objects
  - `glBindTexture(GL_TEXTURE_2D, identifier)` to bind
    - following texture commands refer to the bound texture
  - `glTexParameteri(GL_TEXTURE_2D, …, …)` to specify parameters for use when applying the texture
  - `glTexImage2D(GL_TEXTURE_2D, ….)` to specify the texture data (the image itself)
- enable texturing: `glEnable(GL_TEXTURE_2D)`
- state how the texture will be used:
  - `glTexEnvf(…)`
- specify texture coordinates for the polygon:
  - use `glTexCoord2f(s,t)` before each vertex:
    - `glTexCoord2f(0,0); glVertex3f(x,y,z);`

Low-Level Details

- large range of functions for controlling layout of texture data
  - state how the data in your image is arranged
  - e.g.: `glPixelStorei(GL_UNPACK_ALIGNMENT, 1)` tells OpenGL not to skip bytes at the end of a row
  - you must state how you want the texture to be put in memory: how many bits per “pixel”, which channels,…
- textures must be square and size a power of 2
  - common sizes are 32x32, 64x64, 256x256
  - smaller uses less memory, and there is a finite amount of texture memory on graphics cards
- ok to use texture template sample code for project 4
  - http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=09

Texture Mapping

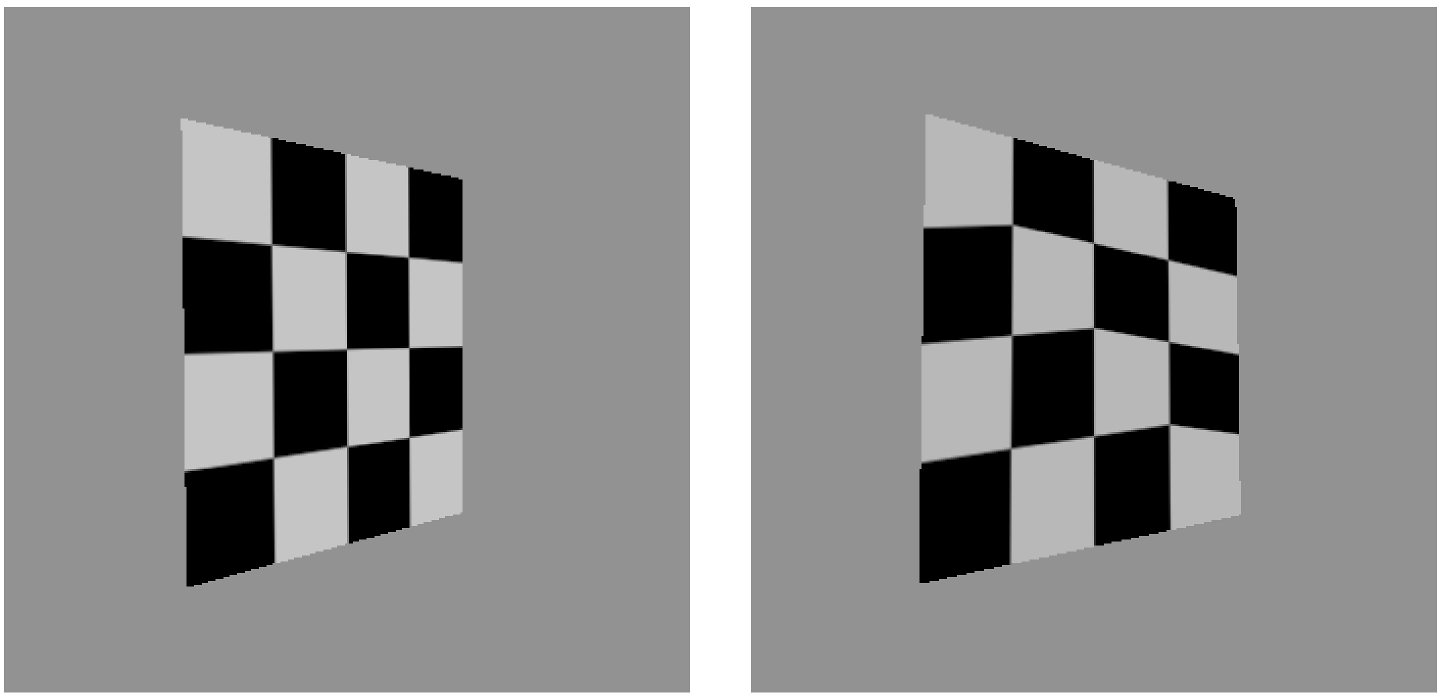
- texture coordinates
  - specified at vertices

```
glTexCoord2f(s,t);
glVertexf(x,y,z);
```

  - interpolated across triangle (like R,G,B,Z)
    - …well not quite!

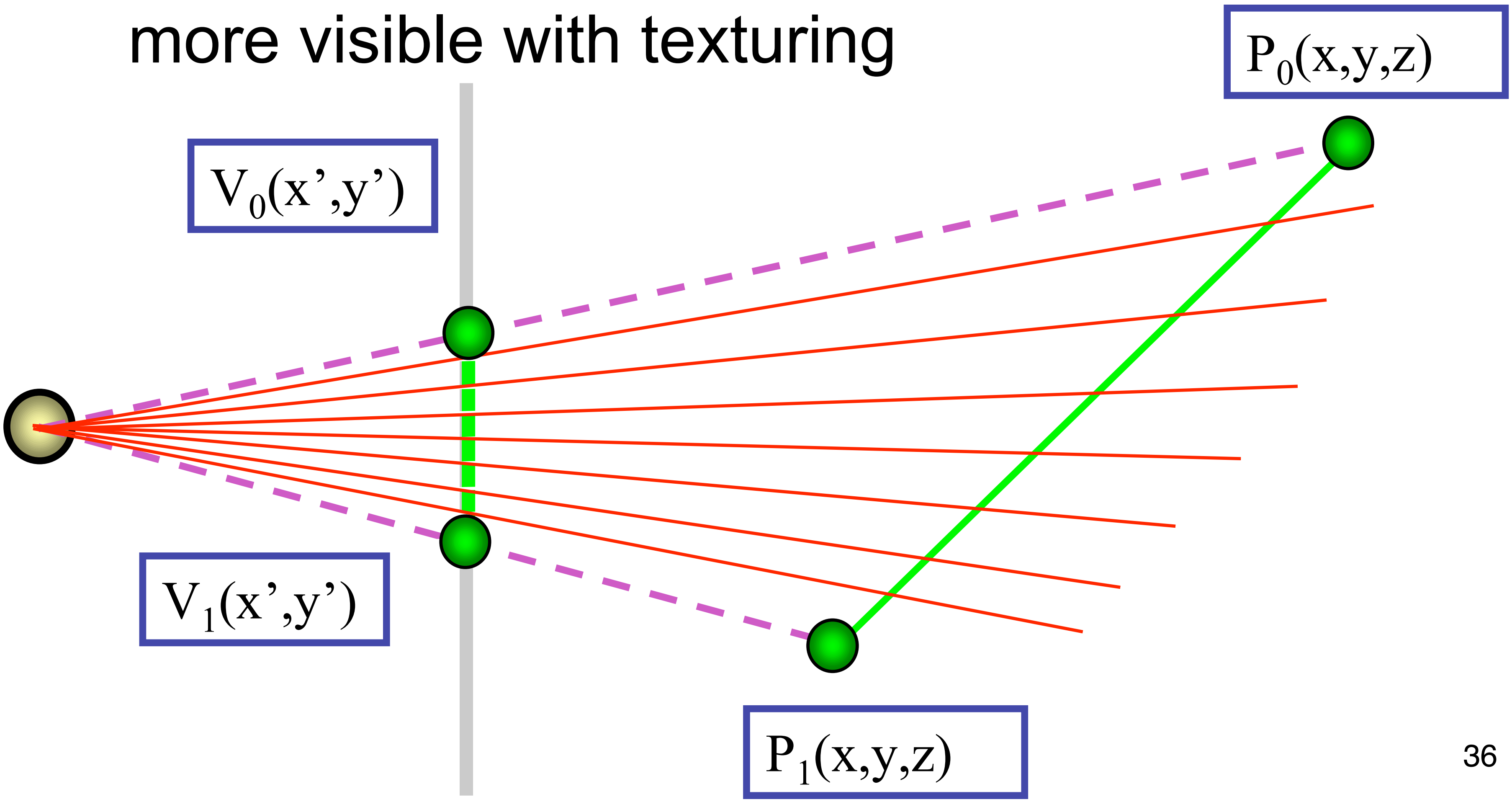
Texture Mapping

- texture coordinate interpolation
  - perspective foreshortening problem



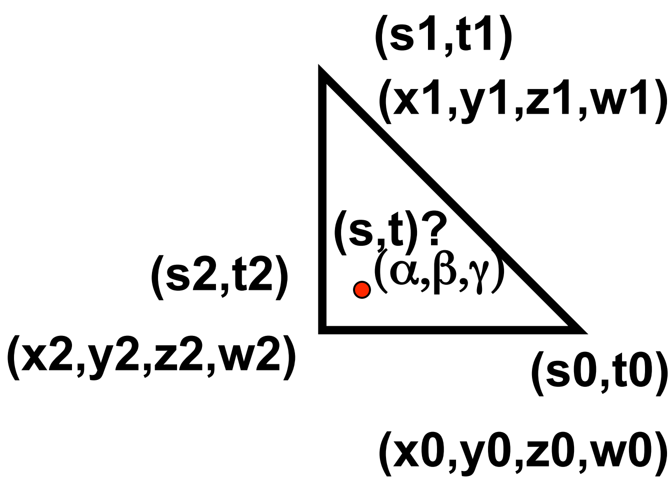
Interpolation: Screen vs. World Space

- screen space interpolation incorrect
  - problem ignored with shading, but artifacts more visible with texturing



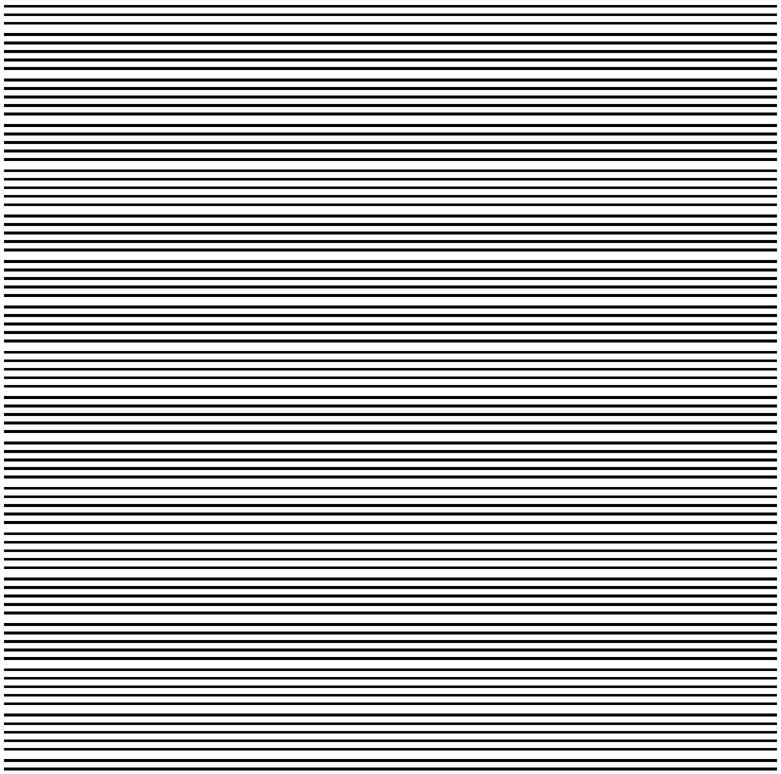
Texture Coordinate Interpolation

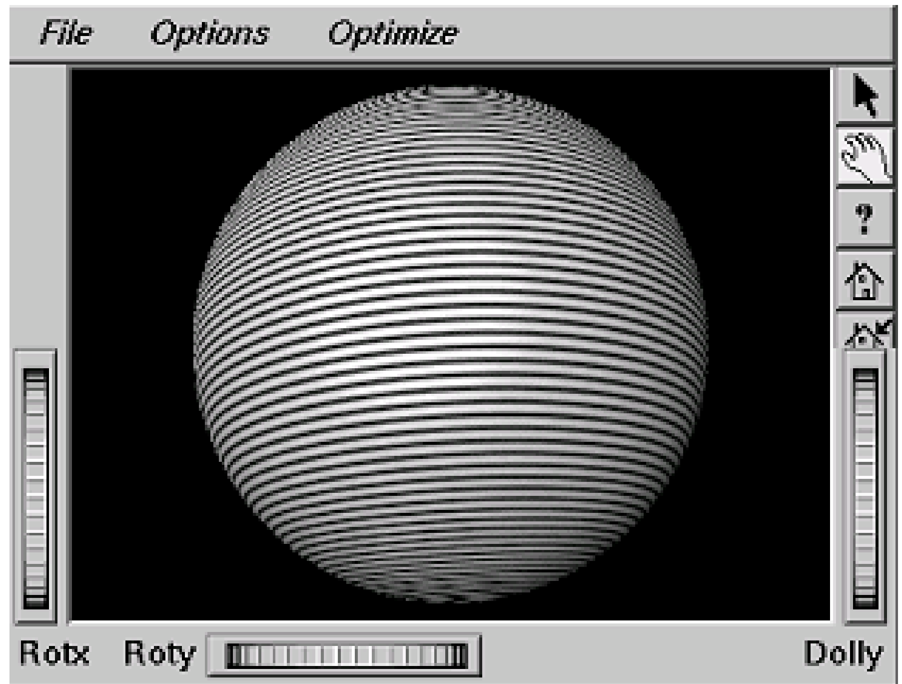
- perspective correct interpolation
  - **α, β, γ** :
    - barycentric coordinates of a point **P** in a triangle
  - *s0*, *s1*, *s2* :
    - texture coordinates of vertices
  - *w0*, *w1*,*w2* :
    - homogeneous coordinates of vertices



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

Reconstruction

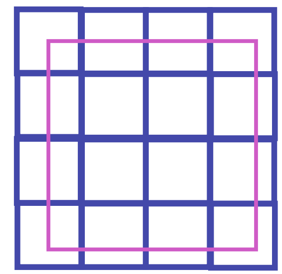




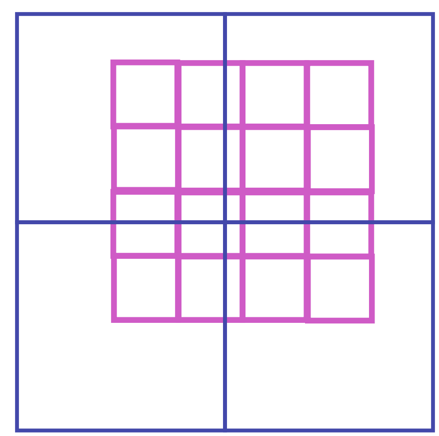
(image courtesy of Kiriakos Kutulakos, U Rochester)

Reconstruction

- how to deal with:
  - pixels that are much larger than texels?
    - apply filtering, “averaging”

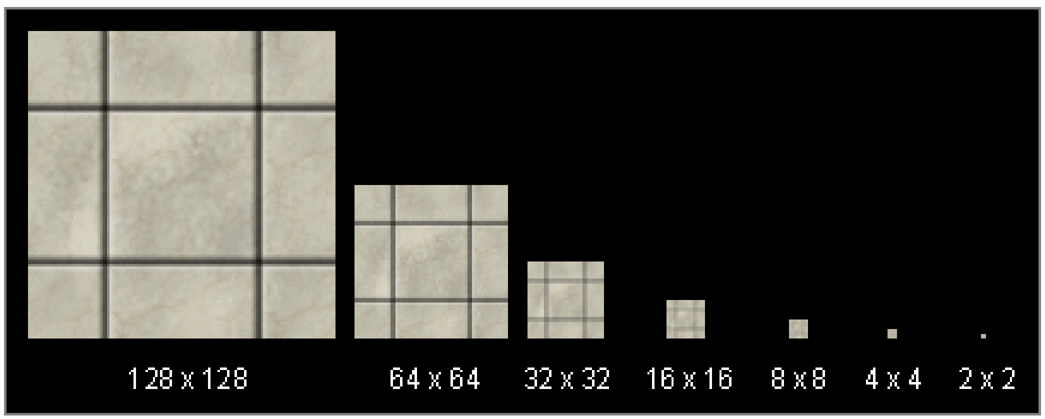


  - pixels that are much smaller than texels ?
    - interpolate

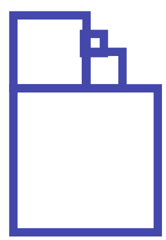


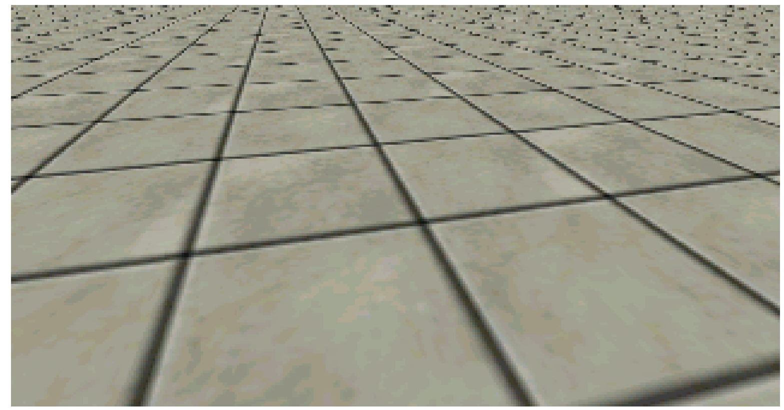
MIPmapping

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

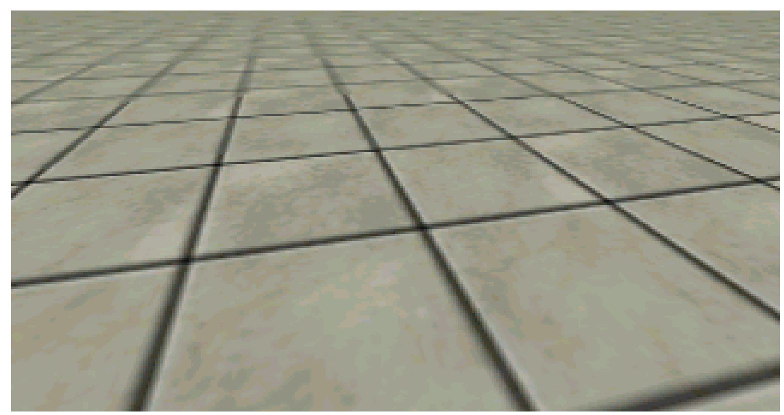


store whole pyramid in single block of memory





Without MIP-mapping

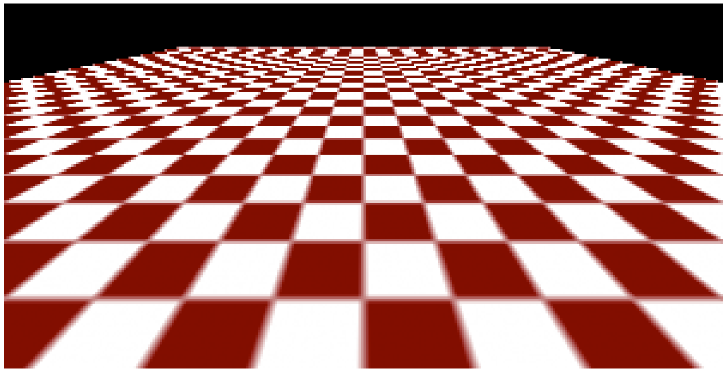


With MIP-mapping

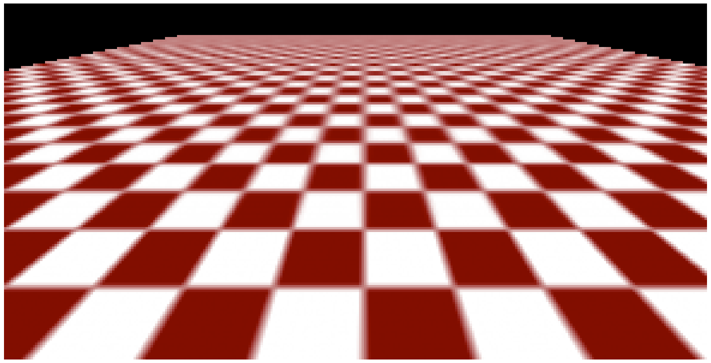
MIPmaps

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

without



with

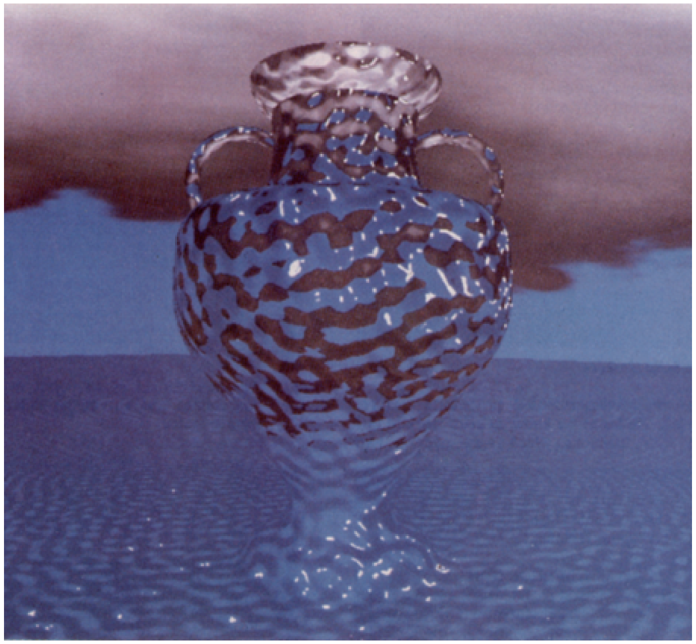


MIPmap storage

- only 1/3 more space required

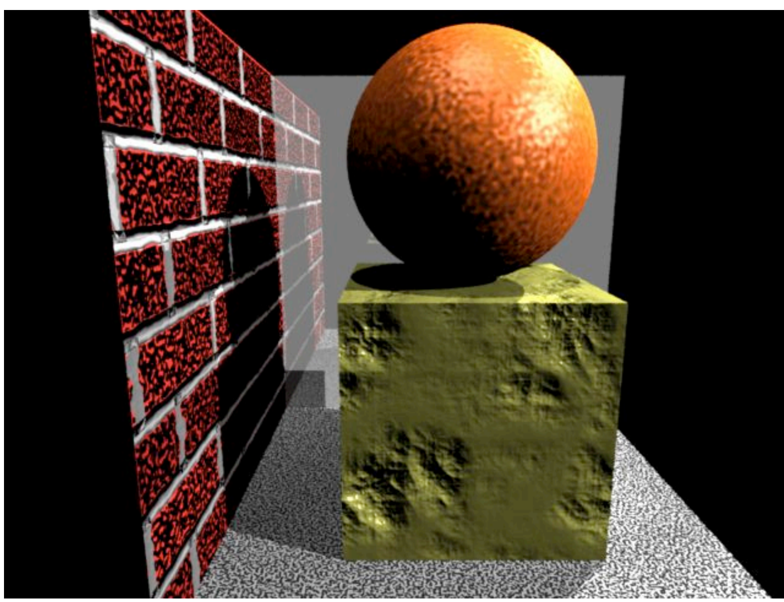
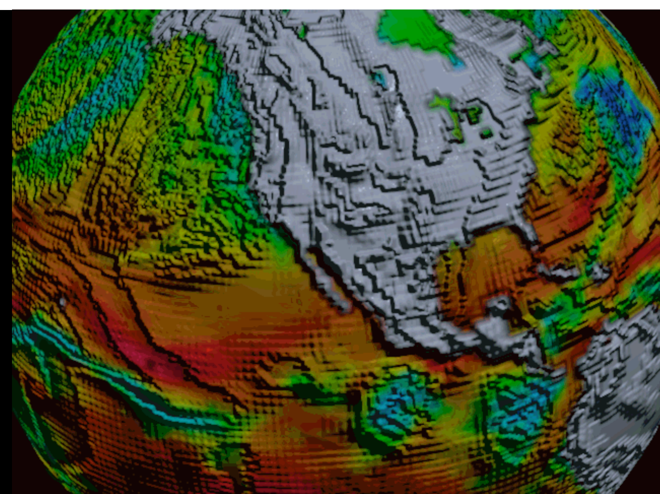
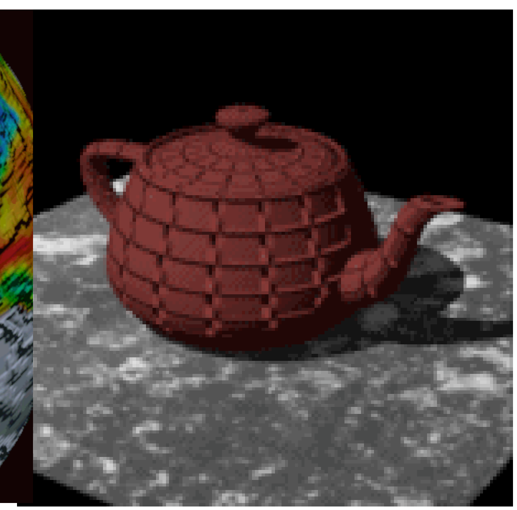
Texture Parameters

- in addition to color can control other material/object properties
  - surface normal (bump mapping)
  - reflected color (environment mapping)

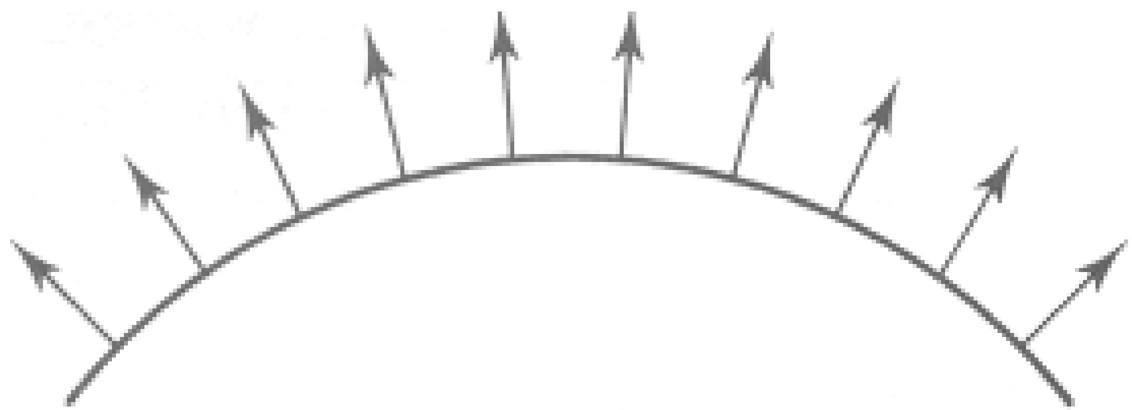


Bump Mapping: Normals As Texture

- 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

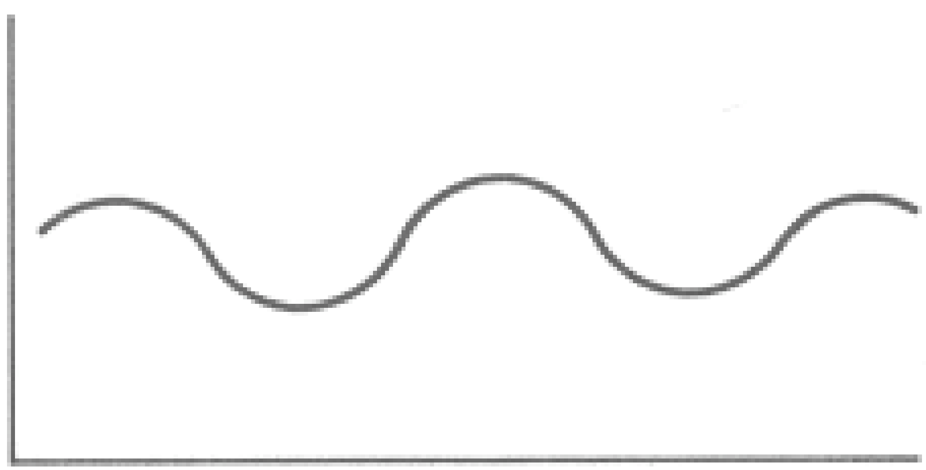




Bump Mapping



$O(u)$

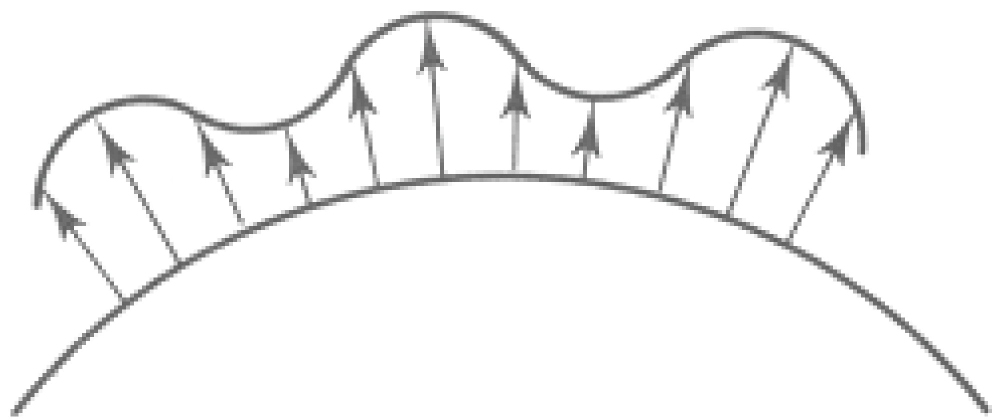
Original surface



$B(u)$

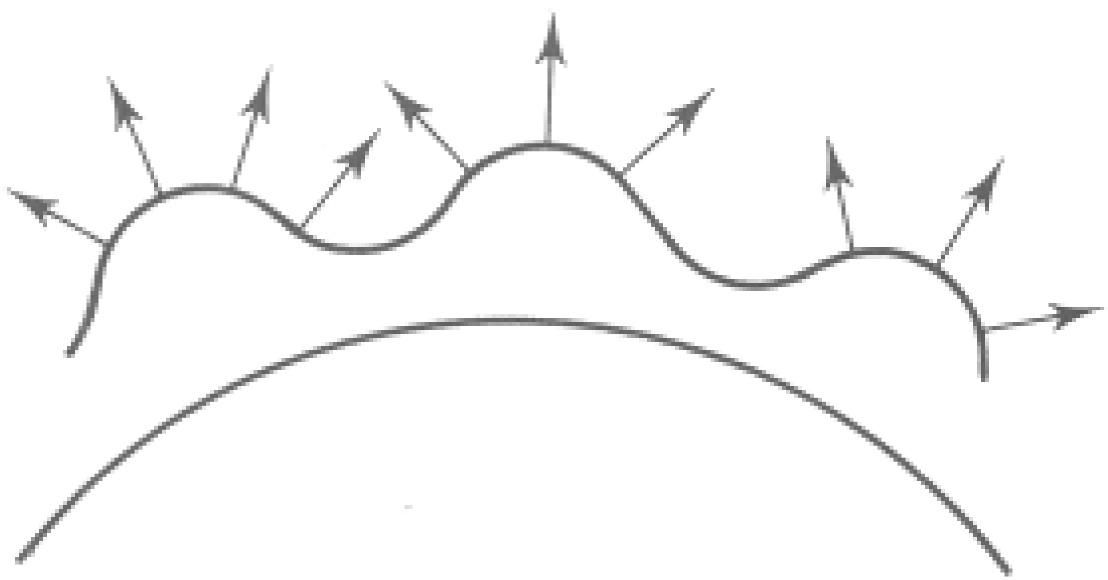
A bump map

Bump Mapping



$O'(u)$

Lengthening or shortening $\boldsymbol{O}(u)$ using $\boldsymbol{B}(u)$

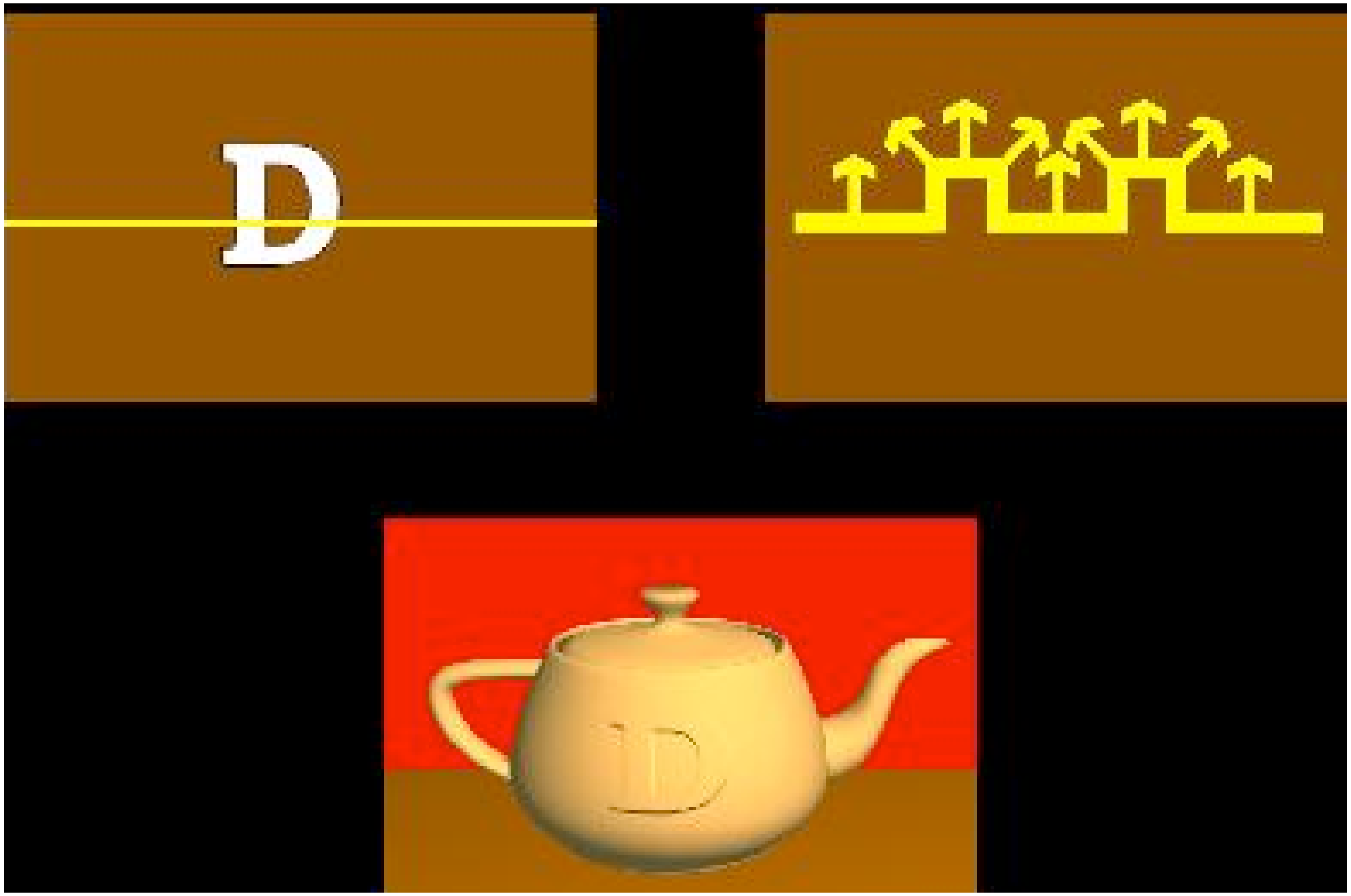


$N'(u)$

The vectors to the 'new' surface

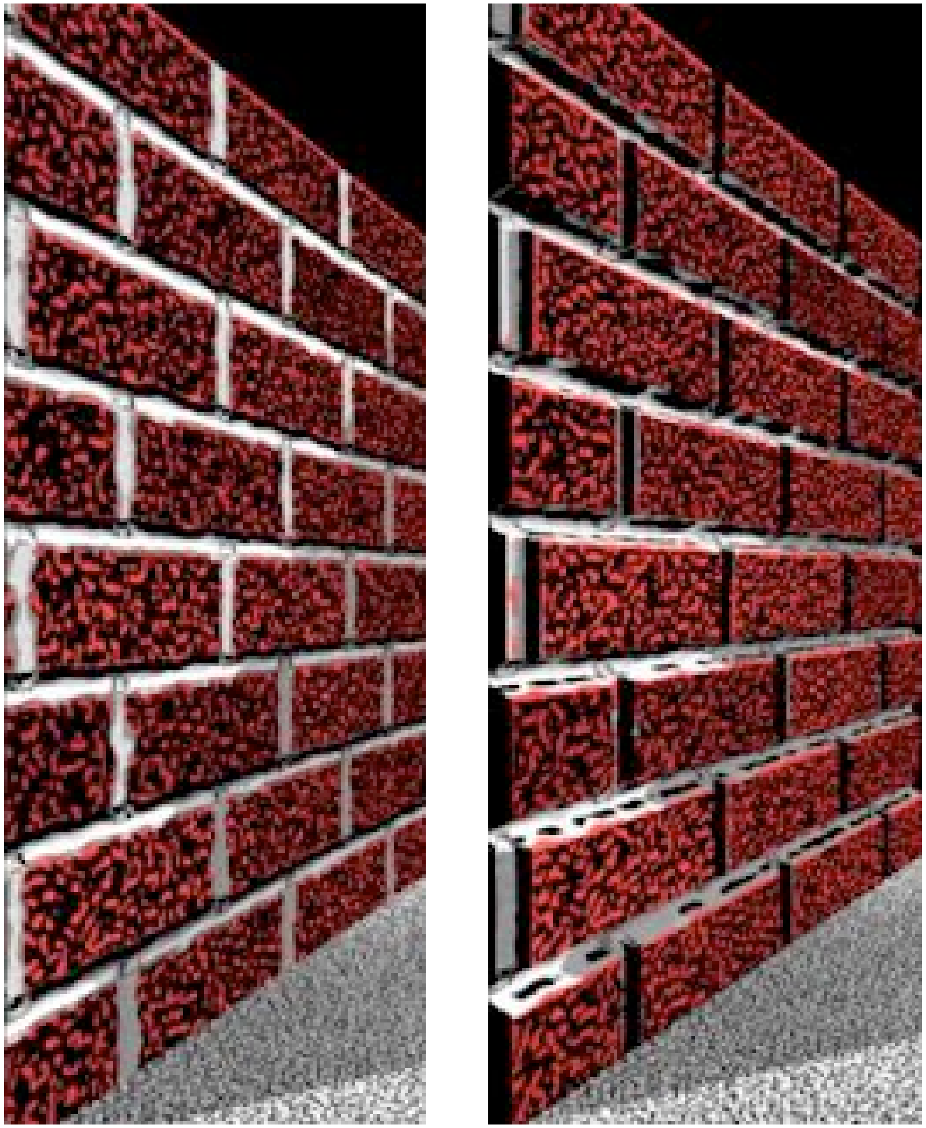
Embossing

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



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

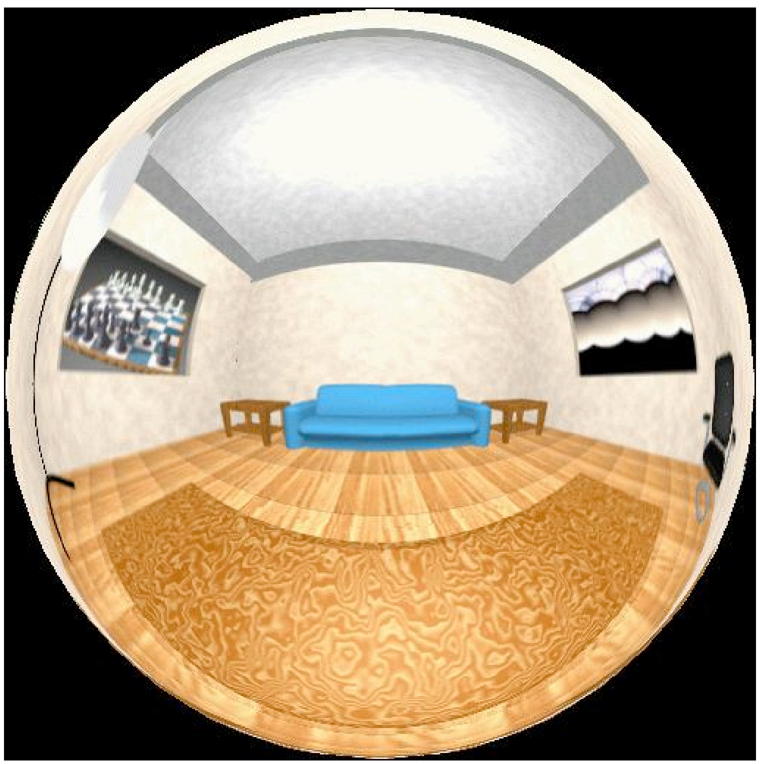
- cheap way to achieve reflective effect
  - generate image of surrounding
  - map to object as texture

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
    - OpenGL support
      - `GL_SPHERE_MAP, GL_CUBE_MAP`
  - others possible too

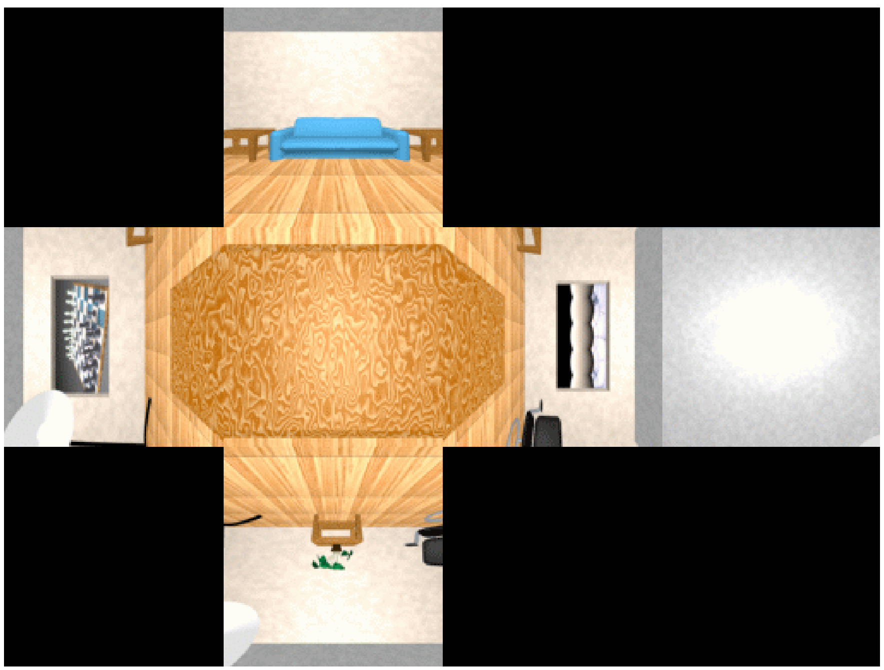
Sphere Mapping

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

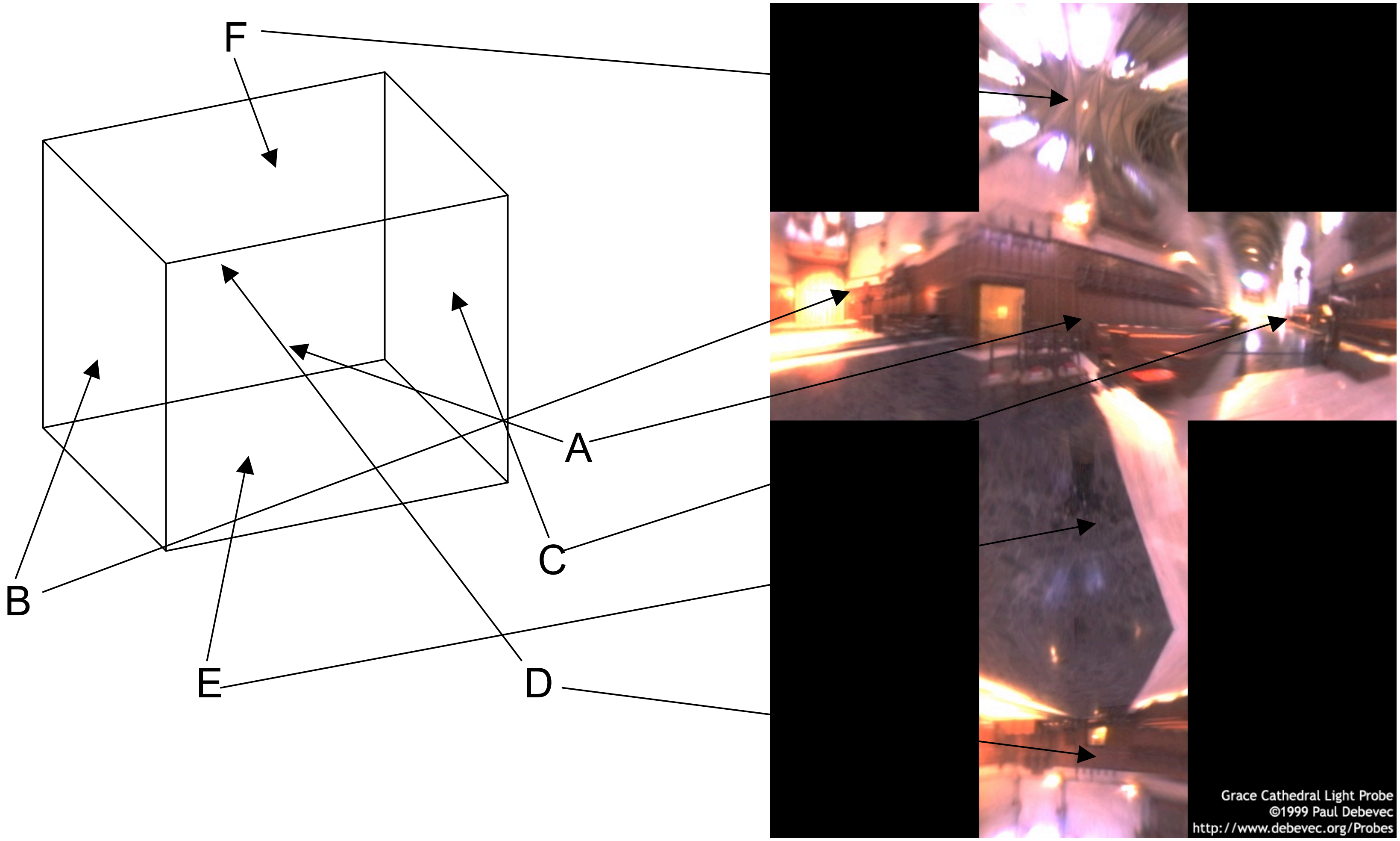



Cube Mapping

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



Cube Mapping



Cube Mapping

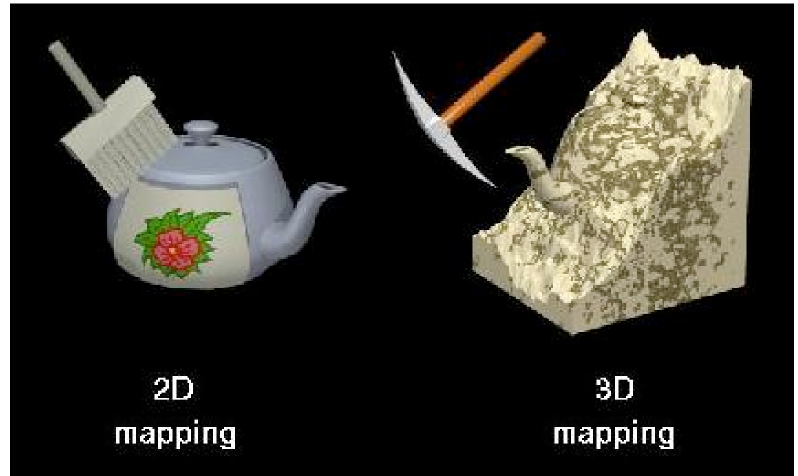
- direction of reflection vector *r* selects the face of the cube to be indexed
  - co-ordinate with largest magnitude
    - e.g., the vector (-0.2, 0.5, -0.84) selects the –Z face
  - remaining two coordinates (normalized by the 3rd coordinate) selects the pixel from the face.
    - e.g., (-0.2, 0.5) gets mapped to (0.38, 0.80).
- difficulty in interpolating across faces

Review: Texture Objects and Binding

- texture objects
  - texture management: switch with bind, not reloading
  - can prioritize textures to keep in memory
  - Q: what happens to textures kicked out of memory?
    - A: resident memory (on graphics card) vs. nonresident (on CPU)
    - details hidden from developers by OpenGL

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
- common for natural material/irregular textures (stone, wood,etc…)



Volumetric Bump Mapping

Marble

Bump

Volumetric Texture Principles

- 3D function ρ

  $\forall \rho = \rho(x,y,z)$

- texture space – 3D space that holds the texture (discrete or continuous)
- rendering: for each rendered point P(x,y,z) compute $\rho(x,y,z)$
- volumetric texture mapping function/space transformed with objects