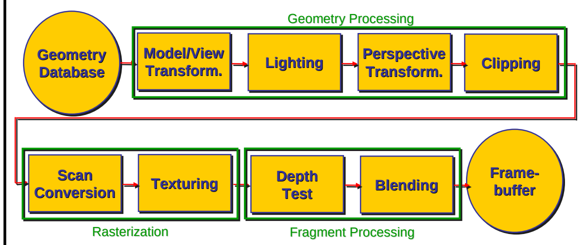
University of British Columbia

Chapter 12

Texture Mapping

The Rendering Pipeline



University of British Columbia

Texture Mapping

- Real life objects non uniform in terms of color & normal
- To generate realistic objects - reproduce coloring & normal variations = **Texture**
- Can often replace complex geometric details

University of British Columbia

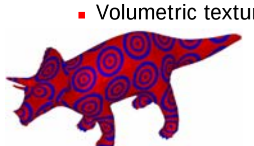
Texture Mapping

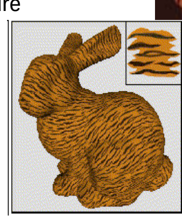
- increase realism
  - lighting/shading models not enough
- hide geometric simplicity
  - images convey illusion of geometry
  - map a brick wall texture on a flat polygon
  - create bumpy effect on surface
- associate 2D information with 3D surface
  - point on surface corresponds to a point in texture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

University of British Columbia

Color Texture Mapping

- Define color (RGB) for each point on object surface
- Two approaches
  - Surface texture map
  - Volumetric texture

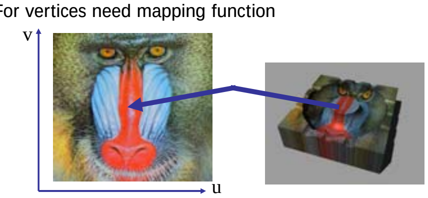




University of British Columbia

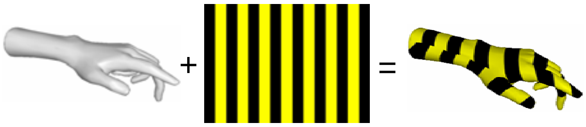
Surface texture

- Define texture pattern over (u,v) domain (Image)
  - Image – 2D array of "texels"
- Assign (u,v) coordinates to each point on object surface
- For free-form – use inverse of surface function
- For polygons (triangle)
  - Inside – use barycentric coordinates
  - For vertices need mapping function



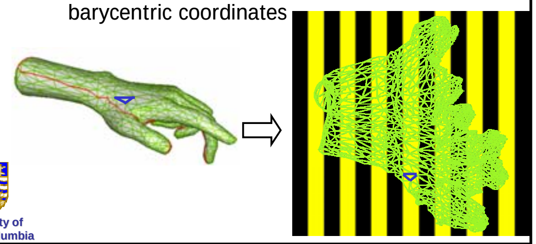
University of British Columbia

Texture Mapping

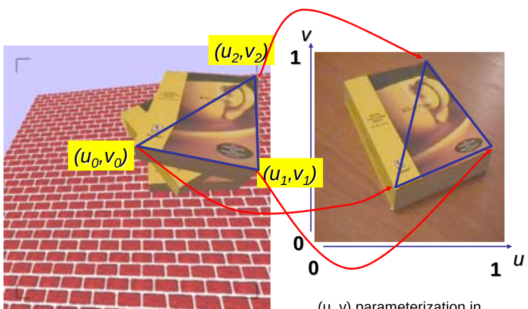


Mapping for Triangular Meshes

- Mapping defined by:
  - Vertices (3D) mapped to specified (u,v) locations in 2D
  - Each interior point mapped to 2D using barycentric coordinates



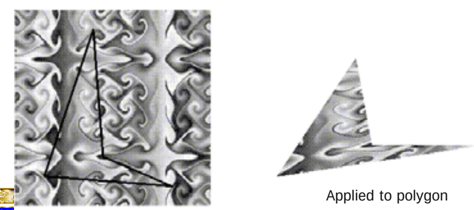
Texture Mapping



(u, v) parameterization in OpenGL

Example Texture Map

- Associate (u,v) with each vertex
  - Not necessarily same proportions as (x,y,z)



Applied to polygon

Texture Coordinates

- every polygon has object coordinates and texture coordinates
  - object coordinates describe where polygon vertices are on the screen
  - texture coordinates describe texel coordinates of each vertex
  - texture coordinates are interpolated along vertex-vertex edges
- glTexCoord2f(TYPE coords)
  - Other versions for different texture dimensions

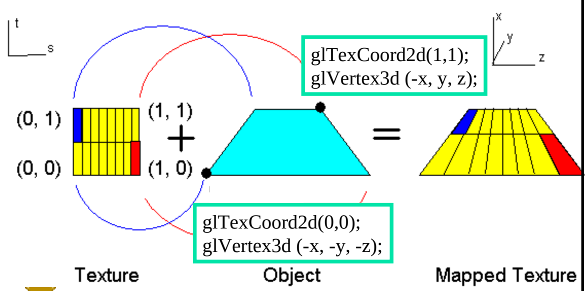
Texture Mapping - OpenGL

- Texture Coordinates
  - Generation/storage at vertices
    - specified by programmer or artist

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

    - generate as a function of vertex coords
  - interpolated across triangle (like R,G,B,Z) (well, not quite...)

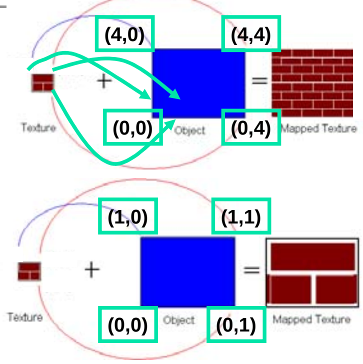
Example Texture Map



Example Texture Map

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

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



Texture Lookup

- issue:
  - what happens to fragments with u or v outside the interval $[0...1]$?

multiple choices:

- cyclic repetition of texture to tile whole surface

  glTexParameteri(..., GL_TEXTURE_WRAP_S, GL_REPEAT)
- clamp every component to range [0...1] - re-use color values from border of texture image

  glTexParameteri(..., GL_TEXTURE_WRAP_S, GL_CLAMP)

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

Texture Mapping

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

Perspective - Reminder

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{d}{d-\alpha} & \frac{-\alpha d}{d-\alpha} \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ (z-\alpha)d/(d-\alpha) \\ z/d \end{bmatrix} \qquad \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} = \begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ \frac{d^2}{d-\alpha}\left(1-\frac{\alpha}{z}\right) \end{bmatrix}$$

- Preserves order
  - BUT distorts distances

Texture Coordinate Interpolation

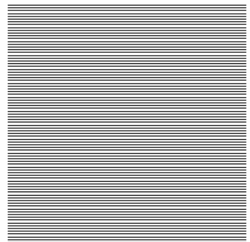
- Perspective Correct Interpolation
  - α, β, γ :
    Barycentric coordinates of point **P**
  - u_0, u_1, u_2 : texture coordinates of vertices
  - w_0, w_1, w_2 : homogenous coordinate of vertices

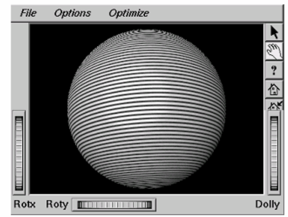
$$u = \frac{\alpha \cdot u_0/w_0 + \beta \cdot u_1/w_1 + \gamma \cdot u_2/w_2}{\alpha/w_0 + \beta/w_1 + \gamma/w_2}$$

  - Similarly for v

University of
British Columbia

Reconstruction



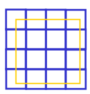


(image courtesy of Kiriakos Kutulakos, U Rochester)

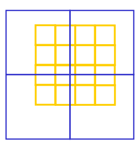
University of
British Columbia

Reconstruction

- How to deal with:
  - pixels that are much larger than texels ?
    (apply filtering, "averaging")



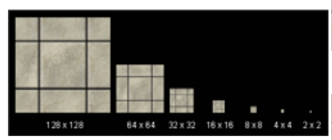
  - pixels that are much smaller than texels ?
    (interpolate)



University of
British Columbia

MIP-mapping

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



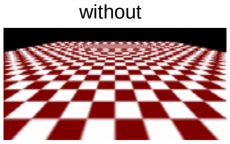
Without MIP-mapping

With MIP-mapping

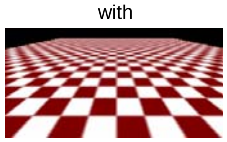
University of
British Columbia

MIP-mapping

without

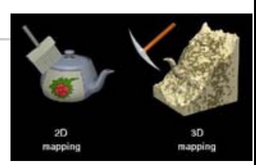


with



University of
British Columbia

Volumetric Texture



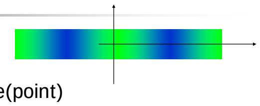
- Define texture pattern over 3D domain - 3D space containing the object
  - Texture function can be digitized or procedural
  - For each point on object compute texture from point location in space
- Common for natural material/irregular textures (stone, wood,etc...)

University of
British Columbia

Principles

- 3D function ρ
  - $\rho = \rho(x,y,z)$
- 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

Effects



- Boring Marble
  - function boring_marble(point)

    x = point.x;

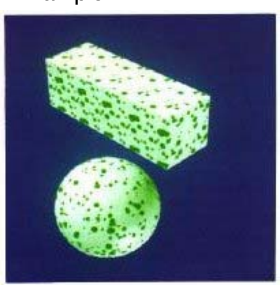
    return marble_color(sin(x));

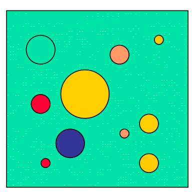
    // marble_color maps scalars to colors
- Bombing
  - Randomly drop bombs of various shapes, sizes and orientation into texture space (store data in table)
  - For point P search table and determine if inside shape
    - if so, color by shape

University of British Columbia

Effects (cont.)

- Otherwise, color by objects color
- Example:





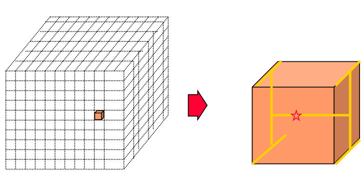
University of British Columbia

Function Noise

- Noise – return scalar for each P(x,y,z)
- Defined as:
  - Initially, for each x,y,z in Z (x, y, z $\in$ N): H(x,y,z) = d (d - randomly chosen value)
  - Retrieval:
  - If (x,y,z) are all integers:
    - Noise(x,y,z) = H(x,y,z)
  - Otherwise:
    - Noise(x,y,z) = interpolation of neighboring H(x,y,z)

University of British Columbia

Function Noise (cont.)



University of British Columbia

Function Turbulence

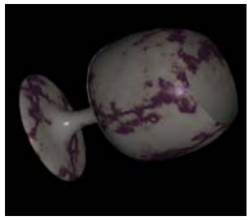
```
function turbulence(p)
    t = 0;
    scale = 1;
    while (scale > pixelsize) {
            t += abs(Noise(p/scale)*scale);
            scale/=2;
    }
return t;
```

University of British Columbia

More Effects

- Marble effect (using turbulence):

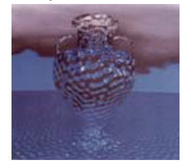
```
function marble(point)
  x = point.x + turbulence(point);
  return marble_color(sin(x))
```

University of British Columbia

Texture Parameters

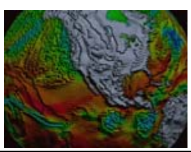
- In addition to color can control other material/object properties
  - Reflectance (either diffuse or specular)
  - Surface normal (bump mapping)
  - Transparency
  - Reflected color (environment mapping)



University of British Columbia

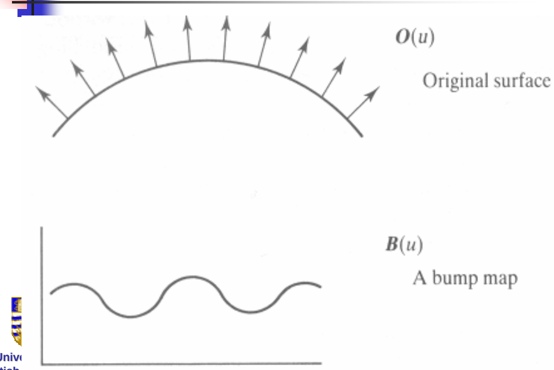
Normal – Bump Mapping

- 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

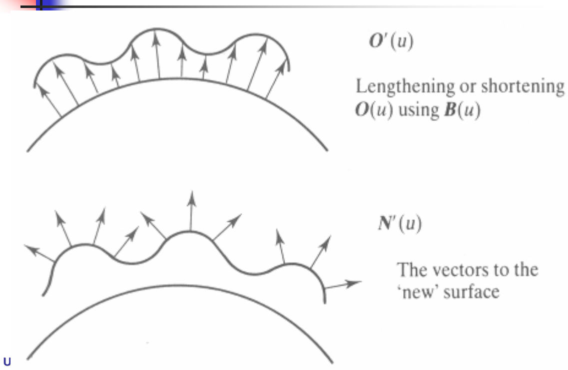
University of British Columbia

Bump Mapping



University of British Columbia

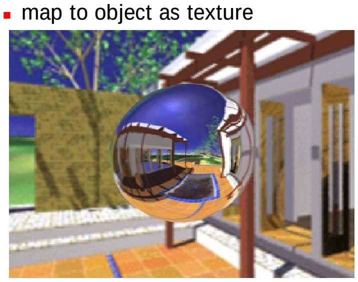
Bump Mapping



University of British Columbia

Environment Mapping

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



University of British Columbia

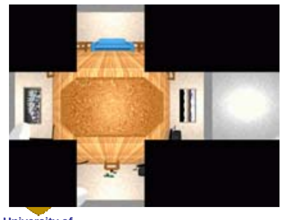
Environment Mapping

- 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

University of British Columbia

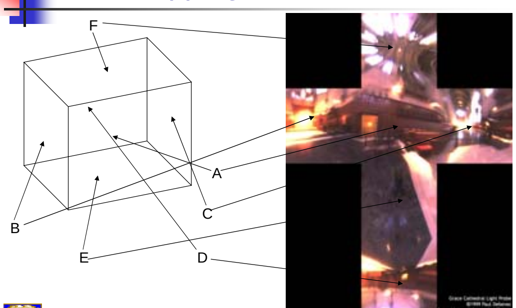
Cube Mapping

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



University of British Columbia

Cube Mapping



University of British Columbia

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